Chapter 18
Computational Semantics

"Then you should say what you mean," the March Hare went on.
"I do," Alice hastily replied; "at least–at least I mean what I say–that's the same thing, you know."
"Not the same thing a bit!" said the Hatter. "You might just as well say that 'I see what I eat' is the same thing as 'I eat what I see'!"

Lewis Carroll, *Alice in Wonderland*

Semantic analysis

This chapter presents a principled computational approach to the problem of **semantic analysis**, the process whereby meaning representations of the kind discussed in the last chapter are composed for linguistic expressions. The automated creation of accurate and expressive meaning representations necessarily involves a wide range of knowledge sources and inference techniques. Among the sources of knowledge that are typically involved are the meanings of words, the conventional meanings associated with grammatical constructions, knowledge about the structure of the discourse, common-sense knowledge about the topic at hand, and knowledge about the state of affairs in which the discourse is occurring.

Syntax-driven semantic analysis

The focus of this chapter is a kind of **syntax-driven semantic analysis** that is fairly modest in its scope. In this approach, meaning representations are assigned to sentences solely on the basis of knowledge gleaned from the lexicon and the grammar. When we refer to an expression's meaning or meaning representation, we have in mind a representation that is both context independent and free of inference. Representations of this type correspond to the traditional notion of literal meaning discussed in the previous chapter.

There are two motivations for proceeding along these lines: first, to provide useful input to those domains, including question answering, in which such primitive representations suffice to produce useful results, and second to use these impoverished representations as inputs to subsequent processes that produce richer, more complete, meaning representations. Chapters 21 and 24 will discuss how these meaning representations can be used in processing extended discourses and dialogs.

18.1 Syntax-Driven Semantic Analysis

Principle of compositionality

The approach detailed in this section is based on the **principle of compositionality**. The key idea behind this approach is that the meaning of a sentence can be constructed from the meanings of its parts. When interpreted superficially, this principle is somewhat less than useful. We know that sentences are composed of words and that words

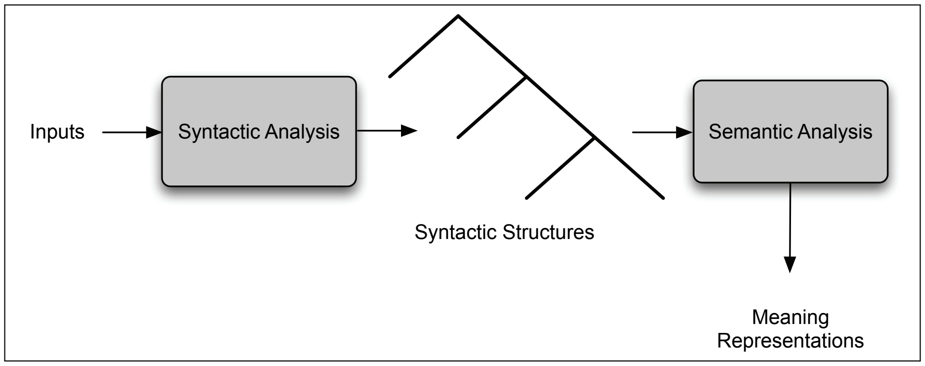


Figure 18.1 A simple pipeline approach to semantic analysis.

are the primary carriers of meaning in language. It would seem then that all this principle tells us is that we should compose the meaning representation for sentences from the meanings of the words that make them up.

Fortunately, the Mad Hatter has provided us with a hint as to how to make this principle useful. The meaning of a sentence is not based solely on the words that make it up, but also on the ordering and grouping of words and on the relations among the words in the sentence. This is just another way of saying that the meaning of a sentence is partially based on its syntactic structure. Therefore, in syntax-driven semantic analysis, the composition of meaning representations is guided by the syntactic *components* and *relations* provided by the kind of grammars discussed in Chapter 12.

Let's begin by assuming that the syntactic analysis of an input sentence serves as the input to a semantic analyzer. Figure 18.1 illustrates an obvious pipeline-oriented approach that follows directly from this assumption. An input is first passed through a parser to derive its syntactic analysis. This analysis is then passed as input to a **semantic analyzer** to produce a meaning representation. Note that although this diagram shows a parse tree as input, other syntactic representations such as flat chunks, feature structures, or dependency structures can also be used. For the remainder of this chapter we assume tree-like inputs.

Semantic analyzer

Before moving on, we should touch on the role of ambiguity in this story. As we've seen, ambiguous representations can arise from numerous sources, including competing syntactic analyses, ambiguous lexical items, competing anaphoric references, and as we show later in this chapter, ambiguous quantifier scopes. In the syntax-driven approach presented here, we assume that syntactic, lexical, and anaphoric ambiguities are not a problem. That is, we'll assume that some larger system is capable of iterating through the possible ambiguous interpretations and passing them individually to the kind of semantic analyzer described here.

Let's consider how such an analysis might proceed with the following example:

(18.1) Franco likes Frasca.

Figure 18.2 shows a simplified parse tree (lacking any feature attachments), along with a plausible meaning representation for this example. As suggested by the dashed arrows, a semantic analyzer given this tree as input might fruitfully proceed by first re-

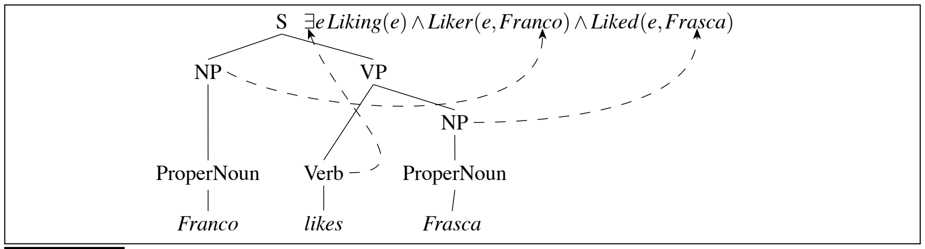


Figure 18.2 Parse tree for the sentence *Franco likes Frasca.*

trieving a skeletal meaning representation from the subtree corresponding to the verb *likes*. The analyzer would then retrieve or compose meaning representations corresponding to the two noun phrases in the sentence. Then, using the representation acquired from the verb as a kind of template, the analyzer would use the noun phrase meaning representations to bind the appropriate variables in the verb representation, thus producing the meaning representation for the sentence as a whole.

Unfortunately, there are a number of serious difficulties with this simplified story. As described, the function used to interpret the tree in Fig. 18.2 must know, among other things, that the verb carries the template upon which the final representation is based, where its corresponding arguments are, and which argument fills which role in the verb's meaning representation. In other words, the function requires a good deal of specific knowledge about *this particular example and its parse tree* to create the required meaning representation. Given that there are an infinite number of such trees for any reasonable grammar, any approach based on one semantic function for every possible tree is in serious trouble.

Fortunately, we have faced this problem before. We do not define languages by enumerating the strings or trees that are permitted, but rather by specifying finite devices that are capable of generating the desired set of outputs. It would seem, therefore, that the right place for semantic knowledge in a syntax-directed approach is with the finite set of devices that generate trees in the first place: the grammar rules and the lexical entries. This is known as the **rule-to-rule hypothesis** (Bach, 1976).

Rule-to-rule hypothesis

Designing an analyzer based on this approach brings us back to the notion of parts and what it means for them to have meanings. The following section is an attempt to answer two questions:

- What does it mean for a syntactic constituent to have a meaning?
- What characteristics must these meanings have so that they can be composed into larger meanings?

18.2 Semantic Augmentations to Syntactic Rules

Semantic attachments

In keeping with the approach used in Chapter 15, we will begin by augmenting our context-free grammar rules with **semantic attachments**. These attachments are instructions that specify how to compute the meaning representation of a construction from the meanings of its constituent parts. Abstractly, our augmented rules have the following structure:

$$A \rightarrow \alpha_1 \ldots \alpha_n \qquad \{f(\alpha_j.sem, \ldots, \alpha_k.sem)\} \tag{18.2}$$

The semantic attachment to the basic context-free rule is shown in the $\{\ldots\}$ to the right of the rule's syntactic constituents. This notation states that the meaning representation assigned to the construction A, which we denote as $A.sem$, can be computed by running the function f on some subset of the semantic attachments of As constituents.

There are myriad ways to instantiate this style of rule-to-rule approach. Our semantic attachments could, for example, take the form of arbitrary programming language fragments. We could then construct a meaning representation for a given derivation by passing the appropriate fragments to an interpreter in a bottom-up fashion and then storing the resulting representations as the value for the associated non-terminals.[1] Such an approach would allow us to create any meaning representation we might like. Unfortunately, the unrestricted power of this approach would also allow us to create representations that have no correspondence at all with the kind of formal logical expressions described in the last chapter. Moreover, this approach would afford us very little guidance as to how to go about designing the semantic attachments to our grammar rules.

For these reasons, more principled approaches are typically used to instantiate the rule-to-rule approach. We introduce two such constrained approaches in this chapter. The first makes direct use of FOL and the λ-calculus notation introduced in Chapter 17. This approach essentially uses a logical notation to guide the creation of logical forms in a principled fashion. The second approach, described later in Section 18.4 is based on the feature-structure and unification formalisms introduced in Chapter 15.

To get started, let's take a look at a basic example along with a simplified target semantic representation.

(18.3) Maharani closed.

$$Closed(Maharani) \tag{18.4}$$

Let's work our way bottom-up through the rules involved in this example's derivation. Starting with the proper noun, the simplest possible approach is to assign a unique FOL constant to it, as in the following.

$$ProperNoun \rightarrow Maharani \qquad \{Maharani\} \tag{18.5}$$

The non-branching *NP* rule that dominates this one doesn't add anything semantically, so we just copy the semantics of the *ProperNoun* up unchanged to the NP.

$$NP \rightarrow ProperNoun \qquad \{ProperNoun.sem\} \tag{18.6}$$

Moving on to the *VP*, the semantic attachment for the verb needs to provide the name of the predicate, specify its arity, and provide the means to incorporate an argument

[1] Those familiar with compiler tools such as YACC and Bison will recognize this approach.

once it's discovered. We can make use of a λ-expression to accomplish these tasks.

$$VP \rightarrow Verb \qquad \{Verb.sem\} \tag{18.7}$$

$$Verb \rightarrow closed \qquad \{\lambda x.Closed(x)\} \tag{18.8}$$

This attachment stipulates that the verb *closed* has a unary predicate *Closed* as its representation. The λ-notation lets us leave unspecified, as the x variable, the entity that is closing. As with our earlier *NP* rule, the intransitive *VP* rule that dominates the verb simply copies upward the semantics of the verb below it.

Proceeding upward, it remains for the semantic attachment for the *S* rule to bring things together by inserting the semantic representation of the subject *NP* as the first argument to the predicate.

$$S \rightarrow NP\ VP \qquad \{VP.sem(NP.sem)\} \tag{18.9}$$

Since the value of *VP.sem* is a λ-expression and the value of *NP.sem* is a simply a FOL constant, we can create our desired final meaning representation by using λ-reduction to apply the *VP.sem* to the *NP.sem*.

$$\lambda x.Closed(x)(Maharani) \tag{18.10}$$

$$Closed(Maharani) \tag{18.11}$$

This example illustrates a general pattern which repeats itself throughout this chapter. The semantic attachments to our grammar rules consist primarily of λ-reductions, whereby one element of an attachment serves as a functor and the rest serve as arguments to it. As we show, the real work resides in the lexicon where the bulk of the meaning representations are introduced.

Although this example illustrates the basic approach, the full story is a bit more complex. Let's begin by replacing our earlier target representation with one that is more in keeping with the neo-Davidsonian representations introduced in the last chapter, and by considering an example with a more complex noun phrase as its subject.

(18.12) Every restaurant closed.

The target representation for this example should be the following.

$$\forall x\, Restaurant(x) \Rightarrow \exists e\, Closed(e) \wedge ClosedThing(e,x) \tag{18.13}$$

Clearly, the semantic contribution of the subject noun phrase in this example is much more extensive than in our previous one. In our earlier example, the FOL constant representing the subject was simply plugged into the correct place in the *Closed* predicate by a single λ-reduction. Here, the final result involves a complex intertwining of the content provided by the *NP* and the content provided by the *VP*. We'll have to do some work if we want to rely on λ-reduction to produce what we want here.

The first step is to determine exactly what we'd like the meaning representation of *Every restaurant* to be. Let's start by assuming that *Every* invokes the $\forall$ quantifier and that *restaurant* specifies the category of concept that we're quantifying over, which

Restriction

we call the **restriction** of the noun phrase. Putting these together, we might expect the meaning representation to be something like $\forall x\, Restaurant(x)$. Although this is a valid FOL formula, its not a terribly useful one, since it says that everything is a restaurant. What's missing from it is the notion that noun phrases like *every restaurant* are normally embedded in expressions that stipulate something about the universally quantified variable. That is, we're probably trying to *say something* about all restaurants. This notion is traditionally referred to as the *NP*'s **nuclear scope**. In this case, the nuclear scope of this noun phrase is *closed*.

Nuclear scope

We can capture these notions in our target representation by adding a dummy predicate, Q, representing the scope and attaching that predicate to the restriction predicate with an $\Rightarrow$ logical connective, leaving us with the following expression:

$$\forall x\, Restaurant(x) \Rightarrow Q(x)$$

Ultimately, what we need to do to make this expression meaningful is to replace Q with the logical expression corresponding to the nuclear scope. Fortunately, the λ-calculus can come to our rescue again. All we need do is permit λ-variables to range over FOL predicates as well as terms. The following expression captures exactly what we need.

$$\lambda Q.\forall x\, Restaurant(x) \Rightarrow Q(x)$$

The following series of grammar rules with their semantic attachments serve to produce this desired meaning representation for this kind of *NP*.

$$NP \rightarrow Det\, Nominal \qquad \{Det.Sem(Nominal.Sem)\} \tag{18.14}$$

$$Det \rightarrow every \qquad \{\lambda P.\lambda Q.\forall x\, P(x) \Rightarrow Q(x)\} \tag{18.15}$$

$$Nominal \rightarrow Noun \qquad \{Noun.sem\} \tag{18.16}$$

$$Noun \rightarrow restaurant \qquad \{\lambda x.Restaurant(x)\} \tag{18.17}$$

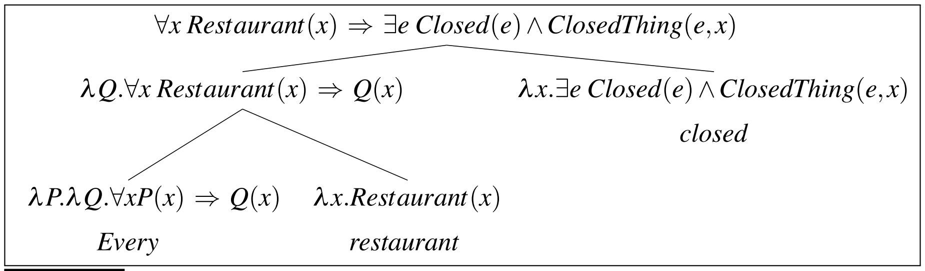
The critical step in this sequence involves the λ-reduction in the *NP* rule. This rule applies the λ-expression attached to the *Det* to the semantic attachment of the *Nominal*, which is itself a λ-expression. The following are the intermediate steps in this process.

$$\lambda P.\lambda Q.\forall x\, P(x) \Rightarrow Q(x)(\lambda x.Restaurant(x))$$

$$\lambda Q.\forall x\, \lambda x.Restaurant(x)(x) \Rightarrow Q(x)$$

$$\lambda Q.\forall x\, Restaurant(x) \Rightarrow Q(x)$$

The first expression is the expansion of the $Det.Sem(Nominal.Sem)$ semantic attachment to the *NP* rule. The second formula is the result of this λ-reduction. Note that this second formula has a λ-application embedded in it. Reducing this expression in place gives us the final form.



$\forall x\, Restaurant(x) \Rightarrow \exists e\, Closed(e) \wedge ClosedThing(e,x)$
$\lambda Q.\forall x\, Restaurant(x) \Rightarrow Q(x)$
$\lambda x.\exists e\, Closed(e) \wedge ClosedThing(e,x)$
closed
$\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)$
$\lambda x.Restaurant(x)$
Every
restaurant

Figure 18.3 Intermediate steps in the semantic interpretation of *Every restaurant closed*.

Having revised our semantic attachment for the subject noun phrase portion of our example, let's move to the *S* and *VP* and *Verb* rules to see how they need to change to accommodate these revisions. Let's start with the *S* rule and work our way down. Since the meaning of the subject *NP* is now a λ-expression, it makes sense to consider it as a functor to be called with the meaning of the *VP* as its argument. The following attachment accomplishes this.

$$S \rightarrow NP\ VP \qquad \{NP.sem(VP.sem)\} \tag{18.18}$$

Note that we've flipped the role of functor and argument from our original proposal for this *S* rule.

The last attachment to revisit is the one for the verb *close*. We need to update it to provide a proper event-oriented representation and to make sure that it interfaces well with the new *S* and *NP* rules. The following attachment accomplishes both goals.

$$Verb \rightarrow close \qquad \{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\} \tag{18.19}$$

This attachment is passed unchanged to the *VP* constituent through the intransitive *VP* rule. It is then combined with the meaning representation of *Every restaurant* as dictated by the semantic attachment for the *S* given earlier. The following expressions illustrate the intermediate steps in this process.

$$\lambda Q.\forall x Restaurant(x) \Rightarrow Q(x)(\lambda y.\exists e Closed(e) \wedge ClosedThing(e,y))$$

$$\forall x Restaurant(x) \Rightarrow \lambda y.\exists e Closed(e) \wedge ClosedThing(e,y)(x)$$

$$\forall x Restaurant(x) \Rightarrow \exists e Closed(e) \wedge ClosedThing(e,x)$$

These steps achieve our goal of getting the *VP*'s meaning representation spliced in as the nuclear scope in the *NP*'s representation. Figure 18.3 shows these steps in the context of the parse tree underlying (18.12).

As is always the case with any kind of grammar engineering effort, we now need to make sure that our earlier simpler examples still work. One area that we need to revisit is our representation of proper nouns. Let's consider them in the context of our earlier example.

(18.20) Maharani closed.

The *S* rule now expects the subject *NP*'s semantic attachment to be a functor applied to the semantics of the *VP*, so our earlier representation of proper nouns as FOL constants won't do. Fortunately, we can once again exploit the flexibility of the λ-calculus to accomplish what we need:

$$\lambda x.x(Maharani)$$

This trick turns a simple FOL constant into a λ-expression, which when reduced serves to inject the constant into a larger expression. You should work through our original example with all of the new semantic rules to make sure that you can come up with the following intended representation:

$$\exists e\, Closed(e) \wedge ClosedThing(Maharani)$$

As one final exercise, let's see how this approach extends to an expression involving a transitive verb phrase, as in the following:

(18.21) Matthew opened a restaurant.

If we've done things correctly we ought to be able to specify the semantic attachments for transitive verb phrases, for the verb *open* and for the determiner *a*, while leaving the rest of our rules alone.

Let's start by modeling the semantics for the determiner *a* on our earlier attachment for *every*.

$$Det \rightarrow a \qquad \{\lambda P.\lambda Q.\exists x\, P(x) \wedge Q(x)\} \tag{18.22}$$

This rule differs from the attachment for *every* in two ways. First, we're using the existential quantifier $\exists$ to capture the semantics of *a*. And second, we've replaced the implies $\Rightarrow$ operator with a logical $\wedge$. The overall framework remains the same, with the λ-variables P and Q standing for the restriction and the nuclear scopes, to be filled in later. With this addition, our existing *NP* rule will create the appropriate representation for *a restaurant*:

$$\lambda Q.\exists x\, Restaurant(x) \wedge Q(x)$$

Next, let's move on to the *Verb* and *VP* rules. Two arguments need to be incorporated into the underlying meaning representation. One argument is available at the level of the transitive *VP* rule, and the second at the *S* rule. Let's assume the following form for the *VP* semantic attachment.

$$VP \rightarrow Verb\, NP \qquad \{Verb.Sem(NP.Sem)\} \tag{18.23}$$

This attachment assumes that the verb's semantic attachment will be applied as a functor to the semantics of its noun phrase argument. And let's assume for now that the representations we developed earlier for quantified noun phrases and proper nouns will remain unchanged. With these assumptions in mind, the following attachment for the verb *opened* will do what we want.

Grammar Rule	**Semantic Attachment**
$S \rightarrow NP\ VP$	$\{NP.sem(VP.sem)\}$
$NP \rightarrow Det\ Nominal$	$\{Det.sem(Nominal.sem)\}$
$NP \rightarrow ProperNoun$	$\{ProperNoun.sem\}$
$Nominal \rightarrow Noun$	$\{Noun.sem\}$
$VP \rightarrow Verb$	$\{Verb.sem\}$
$VP \rightarrow Verb\ NP$	$\{Verb.sem(NP.sem)\}$
$Det \rightarrow every$	$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$
$Det \rightarrow a$	$\{\lambda P.\lambda Q.\exists x P(x) \wedge Q(x)\}$
$Noun \rightarrow restaurant$	$\{\lambda r.Restaurant(r)\}$
$ProperNoun \rightarrow Matthew$	$\{\lambda m.m(Matthew)\}$
$ProperNoun \rightarrow Franco$	$\{\lambda f.f(Franco)\}$
$ProperNoun \rightarrow Frasca$	$\{\lambda f.f(Frasca)\}$
$Verb \rightarrow closed$	$\{\lambda x.\exists e Closed(e) \wedge ClosedThing(e,x)\}$
$Verb \rightarrow opened$	$\{\lambda w.\lambda z.w(\lambda x.\exists e Opened(e) \wedge Opener(e,z)$ $\wedge Opened(e,x))\}$

Figure 18.4 Semantic attachments for a fragment of our English grammar and lexicon.

$$Verb \rightarrow opened \tag{18.24}$$
$$\{\lambda w.\lambda z.w(\lambda x.\exists e\ Opened(e) \wedge Opener(e,z) \wedge OpenedThing(e,x))\}$$

With this attachment in place, the transitive *VP* rule will incorporate the variable standing for *a restaurant* as the second argument to *opened*, incorporate the entire expression representing the *opening* event as the nuclear scope of *a restaurant*, and finally produce a λ-expression suitable for use with our *S* rule. As with the previous example, you should walk through this example step by step to make sure that you arrive at our intended meaning representation.

$$\exists x Restaurant(x) \wedge \exists e Opened(e) \wedge Opener(e, Matthew) \wedge OpenedThing(e,x)$$

The list of semantic attachments we've developed for this small grammar fragment is shown in Fig. 18.4.

In walking through these examples, we have introduced three techniques that instantiate the rule-to-rule approach to semantic analysis introduced at the beginning of this section:

1. Associating complex, function-like λ-expressions with lexical items
2. Copying semantic values from children to parents in non-branching rules
3. Applying the semantics of one of the children of a rule to the semantics of the other children of the rule through λ-reduction.

These techniques serve to illustrate a general division of labor that guides the design of semantic attachments in this compositional framework. In general, the lexical rules introduce quantifiers, predicates, and terms into our meaning representations. The semantic attachments for grammar rules put these elements together in the right ways but generally do not introduce new elements into the representations being created.

18.3 Quantifier Scope Ambiguity and Underspecification

The grammar fragment developed in the last section should handle examples like the following that contain multiple quantified noun phrases.

(18.25) Every restaurant has a menu.

Systematically applying the rules given in Fig. 18.4 to this example produces the following perfectly reasonable meaning representation:

$$\forall x\, Restaurant(x) \Rightarrow \exists y\, (Menu(y) \wedge \exists e\, (Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

This formula more or less corresponds to the common-sense notion that all restaurants have menus.

Unfortunately, this isn't the only possible interpretation for this example. The following is also possible:

$$\exists y\, Menu(y) \wedge \forall x\, (Restaurant(x) \Rightarrow \exists e\, (Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$$

This formula asserts that there is one menu out there in the world and all restaurants share it. From a common-sense point of view, this seems pretty unlikely, but remember that our semantic analyzer only has access to the semantic attachments in the grammar and the lexicon in producing meaning representations. Of course, world knowledge and contextual information can be used to select between these two readings, but only if we are able to produce both.

This example illustrates that expressions containing quantified terms can give rise to ambiguous representations even in the absence of syntactic, lexical, or anaphoric ambiguities. This is known as the problem of **quantifier scoping**. The difference between the two interpretations given above arises from which of the two quantified variables has the outer scope (i.e., which is outside the other in the expression).

Quantifier scoping

The approach outlined in the last section cannot handle this phenomena. The interpretation that is produced is based on the order in which the λ-expressions are reduced as dictated by the grammar and its semantic attachments. To fix this, we need the following capabilities.

- The ability to efficiently create *underspecified representations* that embody all possible readings without explicitly enumerating them
- A means to generate, or extract, all of the possible readings from this representation
- The ability to choose among the possible readings

The following sections outline approaches to the first two problems. The solution to the last, most important, problem requires the use of context and world knowledge and, unfortunately, the problem remains largely unsolved.

18.3.1 Store and Retrieve Approaches

One way to address the quantifier scope problem is to rethink our notion of what the semantic expressions attached to syntactic constituents should consist of. To see this,

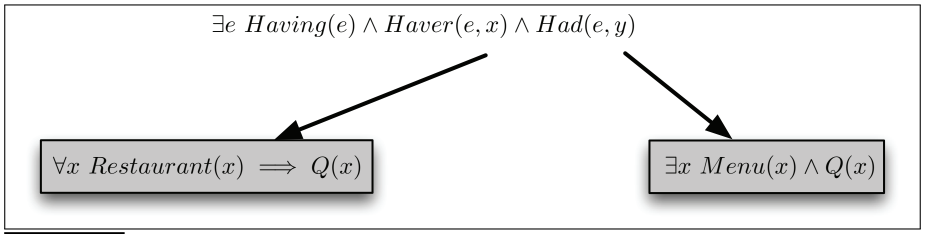

$\exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e,y)$
$\forall x\ Restaurant(x) \implies Q(x)$
$\exists x\ Menu(x) \wedge Q(x)$

Figure 18.5 An abstract depiction of how component meaning representations contribute to the meaning of (18.25).

let's examine the various parts that play a common role in both of the representations given earlier for (18.25). Ignoring the λ-expressions, the representations provided by *has*, *every restaurant*, and *a menu* are as follows.

$$\exists e\, Having(e) \wedge Haver(e,x) \wedge Had(e,y)$$

$$\forall x Restaurant(x) \Rightarrow Q(x)$$

$$\exists x Menu(x) \wedge Q(x)$$

An underspecified representation of this sentence's meaning should specify what we know about how these representations combine and no more. In particular, it would capture that the *restaurant* fills the *Haver* role and that the *menu* fills the *Had* role. However, it should remain agnostic about the placement of the quantifiers in the final representation. Figure 18.5 illustrates these facts graphically.

Cooper storage

To provide this kind of functionality, we introduce the notion of **Cooper storage** (Cooper, 1983) and once again leverage the power of λ-expressions. Recall that in our original approach to semantic analysis, we assigned a single FOL formula to each node in a parse tree. In the new approach, we replace this single semantic attachment with a store. The store includes a core meaning representation for a node along with an indexed list of quantified expressions gathered from the nodes below this node in the tree. These quantified expressions are in the form of λ-expressions that can be combined with the core meaning representation to incorporate the quantified expressions in the right way.

The following store would be associated with the node at the top of the parse tree for (18.25).

$$\begin{aligned}&\exists e\, Having(e) \wedge Haver(e,s_1) \wedge Had(e,s_2)\\ &\quad(\lambda Q.\forall x\, Restaurant(x) \Rightarrow Q(x), 1),\\ &\quad(\lambda Q.\exists x\, Menu(x) \wedge Q(x), 2)\end{aligned}$$

This store contains exactly what we said we needed—the *Haver* and *Had* roles have been correctly assigned just as in our earlier approach but we also have access to the original quantified expressions through the indexes on the variables s_1 and s_2. These indexes pick out the corresponding quantified expressions in the store. Note that contents of this store correspond directly to the representations given in Fig. 18.5.

To retrieve a fully-specified representation from a store, we first choose one of the elements of the store and apply it to the core representation through a λ-reduction. As an example, let's assume that we pull the second element of the store first. This results in the following λ-application

$$\lambda Q.\exists x\,(Menu(x) \wedge Q(x))$$
$$(\lambda s_2.\exists e\, Having(e) \wedge Haver(e,s_1) \wedge Had(e,s_2))$$

and its subsequent reduction.

$$\exists x\,(Menu(x) \wedge \exists e\, Having(e) \wedge Haver(e,s_1) \wedge Had(e,x))$$

Note that we use the index variable s_2 as the λ-variable for core representation's λ-expression. This ensures that the quantified expression retrieved from the store will be assigned to the correct role in the core representation.

The store now contains this new core representation and one remaining quantified λ-expression. Pulling that expression out of the store and applying it to the core results in the following λ-application

$$\lambda Q.\forall x\,(Restaurant(x) \Rightarrow Q(x))$$
$$(\lambda .s_1 \exists y\,(Menu(y) \wedge \exists e\, Having(e) \wedge Haver(e,s_1) \wedge Had(e,x))$$

and subsequent reduction.

$$\forall x\, Restaurant(x) \Rightarrow \exists y\, Menu(y) \wedge \exists e\, Having(e) \wedge Haver(e,x) \wedge Had(e,y))$$

This process yields exactly the same expression that we got with our original approach. To get the alternative interpretation, in which *a menu* has the outer scope, we simply pull the elements out of the store in the opposite order.

In working through this example, we assumed that we already had the contents of the store for this sentence. But how did this store get populated? For the core representations in the store, we simply apply our semantic attachments in exactly the same fashion as we normally would. That is, we either copy them up unchanged from child to parent in the tree, or we apply the core representation of one child to the representation of the others as dictated by our semantic attachments in the grammar. We store the result as the core representation in the parent's store. Any quantified indexed expressions in the store are simply copied up to the store of the parent.

To see how this works, let's consider the *VP* node in the derivation of (18.25). Figure 18.6 illustrates the stores for all of the nodes in this example. The core representation for this *VP* results from the application of the transitive *VP* rule given earlier in Fig. 18.4. In this case, we apply the representation of the *Verb* to the representation of the argument *NP*. The indexed quantified expression in the *NP*'s store is copied up to the *VP*'s store.

Things get more interesting in the computation of the store for *a menu*. The relevant rule in Fig. 18.4 instructs us to apply the meaning representation of the *Det* to the representation of the *Nominal*, thus providing us with the representation shown as the second element of the *NP*'s store in Fig. 18.6. Now, if we were to store this expression

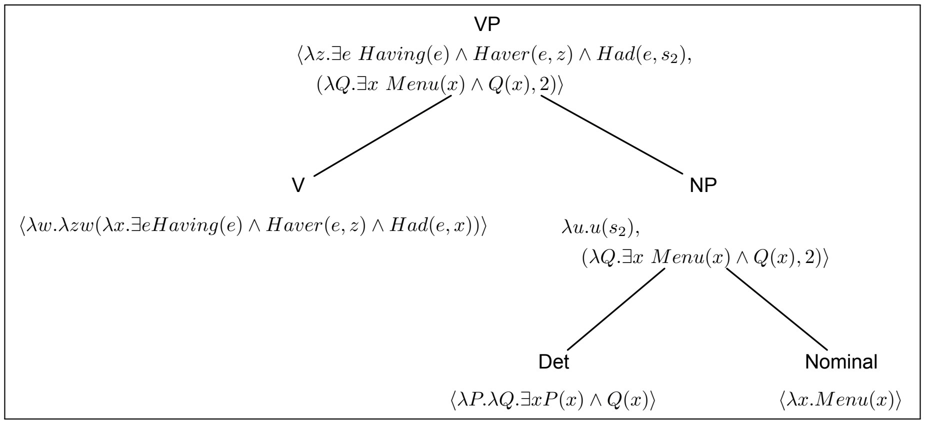

VP
$\langle \lambda z.\exists e\ Having(e) \wedge Haver(e,z) \wedge Had(e,s_2),$
$(\lambda Q.\exists x\ Menu(x) \wedge Q(x), 2)\rangle$
V
$\langle \lambda w.\lambda zw(\lambda x.\exists e Having(e) \wedge Haver(e,z) \wedge Had(e,x))\rangle$
NP
$\lambda u.u(s_2),$
$(\lambda Q.\exists x\ Menu(x) \wedge Q(x), 2)\rangle$
Det
$\langle \lambda P.\lambda Q.\exists x P(x) \wedge Q(x)\rangle$
Nominal
$\langle \lambda x.Menu(x)\rangle$

Figure 18.6 Semantic stores for the *VP* subtree for (18.25).

as the core representation for this *NP* we would be right back to our original formulation. It would subsequently be consumed by the *VP* node and a hard commitment to the placement of this quantified term would be made.

To avoid this, we use the same representational trick we used earlier to turn FOL constants into λ-expressions. When a quantified *NP* is introduced, we create a new index variable, in this case s_2, and wrap a λ-application around that variable. This index on the variable points at the corresponding quantified expression in the store. This new λ-expression serves as the core representation for the new *NP*. And, as is shown in Fig. 18.6, it is this variable that is subsequently bound as the appropriate semantic argument in the event representation.

18.3.2 Constraint-Based Approaches

Unfortunately, this storage-based approach suffers from two problems. First, it only addresses the problem of scope ambiguities introduced by quantified noun phrases. However, a wide array of syntactic constructions and lexical items also introduce similar ambiguities. Consider the following example and its associated interpretations.

(18.26) Every restaurant did not close.

$$\neg(\forall x\, Restaurant(x) \Rightarrow \exists e\, Closing(e) \wedge Closed(e,x))$$

$$\forall x\, Restaurant(x) \Rightarrow \neg(\exists e\, Closing(e) \wedge Closed(e,x))$$

Nothing in the store-and-retrieve method allows us to handle this problem. Of course, we could incorporate additional mechanisms into the method to handle negation but the resulting approach becomes more and more ad hoc as additional problems are encountered.

Even if we could extend the store-and-retrieve approach to handle additional sources of ambiguity, there is a second more critical shortcoming. Although it allows us to enumerate all the possible scopings for a given expression, it doesn't allow us to impose additional constraints on those possibilities. This is an ability that is crucial if we wish to be able to apply specific lexical, syntactic, and pragmatic knowledge to narrow down the range of possibilities for any given expression.

The solution to these problems lies in a change of perspective. Instead of taking what is essentially a procedural focus on how to retrieve fully-specified representations from stores, we ought to focus instead on how to effectively represent underspecified representations, including any constraints that any final representation must satisfy. In this view, any fully-specified FOL expression that is consistent with the constraints is valid.

Hole semantics

There are a number of current approaches that address the underspecification problem from this constraint-based perspective. The **hole semantics** (Bos, 1996) approach we describe here is representative of the field. The Historical Notes section at the end of the chapter surveys the other constraint-based approaches.

Hole

Labels

To get a feel for this approach, let's return to Fig. 18.5 on page 593. The Q predicates in the quantified expressions are place-holders that will eventually be replaced by arbitrary FOL expressions through a series of coordinated λ-reductions. In the hole semantics approach, we replace these λ-variables with **holes**. Instead of using λ-reductions to fill these holes, we first add **labels** to all of the candidate FOL subexpressions. In a fully-specified formula, all holes will be filled with labeled subexpressions.

Dominance constraints

Of course, we can't fill holes with just any labeled expression, so we'll add **dominance constraints** between holes and labels that restrict which labels can fill which holes. More formally, an expression of the form $l \leq h$ asserts that the expression containing hole h dominates the expression with label l. This simply means that the expression containing h must ultimately have l as a subexpression, and not the other way around.

Let's walk through this in the context of (18.25). As we showed in Fig. 18.5 on page 593, there are three quantified expressions that play a role in this example: the *restaurant*, the *menu*, and the *having*. Let's label them l_1, l_2, and l_3, respectively. We'll use the hole h_0 as a place-holder for the final representation. Replacing the λ-variables, we'll use h_1 to stand for the nuclear scope of *restaurant*, and h_1 to stand for the nuclear scope of *menu*.

Since h_0 stands for the entire expression it trivially dominates all of the expressions in the underspecified representation. Moreover, we know that in both of the two possible interpretations, the holes h_1 and h_2 outscope the event labeled by l_3. However, since we want to remain agnostic about how l_1 and l_2 relate to each other no dominance relations are specified concerning them. This leaves us with the following underspecified representation for (18.25)

$$\begin{aligned}
&l_1 : \forall x\, Restaurant(x) \Rightarrow h_1\\
&l_2 : \exists x\, Menu(y) \wedge h_2\\
&l_3 : \exists e\, Having(e) \wedge Haver(e,x) \wedge Had(e,y)\\
&l_1 \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2
\end{aligned}$$

Figure 18.7 captures these facts graphically.

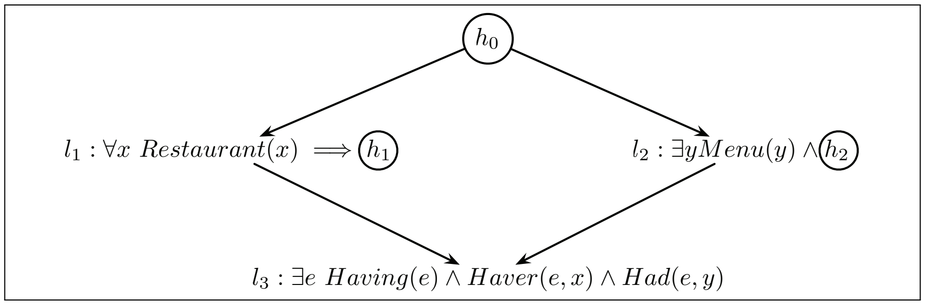

h_0
$l_1 : \forall x\ Restaurant(x) \implies h_1$
$l_2 : \exists y Menu(y) \wedge h_2$
$l_3 : \exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e,y)$

Figure 18.7 Hole semantic representation for *Every restaurant has a menu.*

Plugging

Now that we have an underspecified representation, how can we retrieve a fully-specified FOL representation? Clearly, what we need is a **plugging** that fills all the holes with labeled expressions in a way that respects all of the stated constraints. More formally, a plugging is a one-to-one mapping from holes to labels that satisfies all the given constraints. When we have plugged all the holes consistently, we have a fully-specified representation.

Let's start by filling h_0. The two candidate plugs for this hole are l_1 and l_2. Since h_0 dominates both l_1 and l_2 and neither one dominates the other, the choice is arbitrary. Let's assume that we plug h_0 with l_1, denoted as $P(h_0) = l_1$. Once we've made this choice, the remaining assignments are fully determined by the constraints; that is, $P(h_1) = l_2$ and $P(h_2) = l_3$. This plugging corresponds to the following interpretation where $\forall x\, Restaurant(x)$ has the outer-scope.

$$\forall x Restaurant(x) \Rightarrow \exists y(Menu(y) \wedge \exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y))$$

Figure 18.8 illustrates the steps in this process graphically. Not surprisingly, the alternative interpretation in which *menu* has the outer scope is obtained by starting with the assignment $P(h_0) = l_2$.

To implement this approach, we have to define a language to associate labels and holes with FOL expressions and to express dominance constraints between holes and labels. Blackburn and Bos (2005) describe a meta-language based on FOL and λ-expressions for doing just that. These FOL statements play the role of our semantic attachments to grammar rules.

This constraint-based approach to underspecification addresses many of the problems with the store-and-retrieve approach that we raised at the beginning of this section. First, the approach is not specific to any particular grammatical construction or source of scope ambiguity. This follows since we can label, or designate as holes, essentially arbitrary pieces of FOL formula. Second, and perhaps more importantly, dominance constraints give us the power to express constraints that can rule out unwanted interpretations. The source of these constraints can come from specific lexical and syntactic knowledge and can be expressed directly in the semantic attachments to lexical entries and grammar rules.

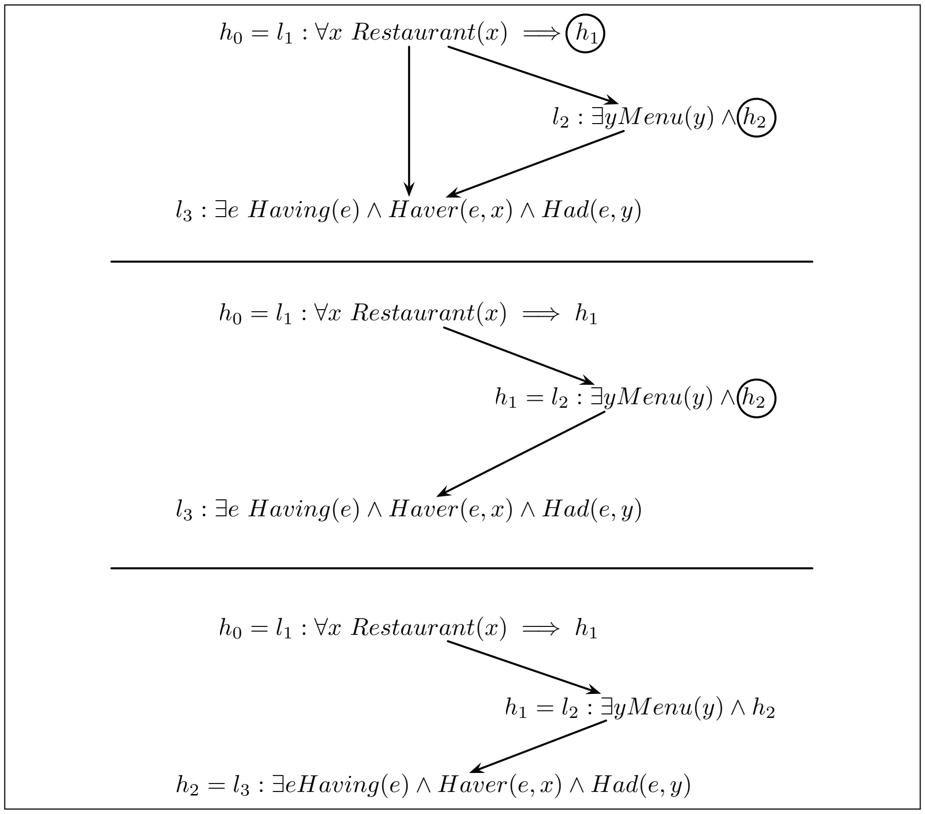

$h_0 = l_1 : \forall x\ Restaurant(x) \implies h_1$
$l_2 : \exists y Menu(y) \wedge h_2$
$l_3 : \exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e,y)$
$h_0 = l_1 : \forall x\ Restaurant(x) \implies h_1$
$h_1 = l_2 : \exists y Menu(y) \wedge h_2$
$l_3 : \exists e\ Having(e) \wedge Haver(e,x) \wedge Had(e,y)$
$h_0 = l_1 : \forall x\ Restaurant(x) \implies h_1$
$h_1 = l_2 : \exists y Menu(y) \wedge h_2$
$h_2 = l_3 : \exists e Having(e) \wedge Haver(e,x) \wedge Had(e,y)$

Figure 18.8 Steps in arriving at a valid plugging for *Every restaurant has a menu.*

18.4 Unification-Based Approaches to Semantic Analysis

As mentioned in Section 18.2, feature structures and the unification operator provide an effective way to implement syntax-driven semantic analysis. Recall that in Chapter 15 we paired complex feature structures with individual context-free grammar rules to encode syntactic constraints such as number agreement and subcategorization, constraints that were awkward or in some cases impossible to convey directly with context-free grammars. For example, the following rule was used to capture agreement constraints on English noun phrases:

NP $\rightarrow$ *Det Nominal*
⟨*Det* AGREEMENT⟩ = ⟨*Nominal* AGREEMENT⟩
⟨*NP* AGREEMENT⟩ = ⟨*Nominal* AGREEMENT⟩

Rules such as this serve two functions at the same time: they ensure that the grammar rejects expressions that violate this constraint, and more importantly for our current topic, they create complex structures that can be associated with parts of grammatical derivations. The following structure, for example, results from the application of the above rule to a singular noun phrase.

$$\begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{NUMBER} & sg\end{bmatrix}\end{bmatrix}$$

We'll use this latter capability to compose meaning representations and associate them with constituents in parse.

In this unification-based approach, our FOL representations and λ-based semantic attachments are replaced by complex feature structures and unification equations. To see how this works, let's walk through a series of examples similar to those discussed earlier in Section 18.2. Let's start with a simple intransitive sentence with a proper noun as its subject.

(18.27) Rhumba closed

With an event-oriented approach, the meaning representation for this sentence should be something like the following.

$$\exists e\, Closing(e) \wedge Closed(e, Rhumba)$$

Our first task is to show that we can encode representations like this within a feature-structure framework. The most straightforward way to approach this task is to simply follow the BNF-style definition of FOL statements given in Chapter 17. The relevant elements of this definition stipulate that FOL formulas come in three varieties: atomic formulas consisting of predicates with the appropriate number of term arguments; formulas conjoined with other formulas by the $\wedge$, $\vee$ and $\Rightarrow$ operators; and quantified formulas that consist of a quantifier, variables, and a formula. Using this definition as a guide, we can capture this FOL expression with the following feature structure.

$$\begin{bmatrix}\text{QUANT} & \exists \\ \text{VAR} & \boxed{1} \\ \text{FORMULA} & \begin{bmatrix}\text{OP} & \text{AND} \\ \text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{CLOSING} \\ \text{ARG0} & \boxed{1}\end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix}\text{PRED} & \text{CLOSED} \\ \text{ARG0} & \boxed{1} \\ \text{ARG1} & \text{RHUMBA}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 18.9 shows this expression in the DAG-style notation introduced in Chapter 15. The figure reveals how variables are handled. Instead of introducing explicit FOL variables, we use the path-based, feature-sharing capability of feature structures to accomplish the same goal. In this example, the event variable e is captured by the three paths leading to the same shared node.

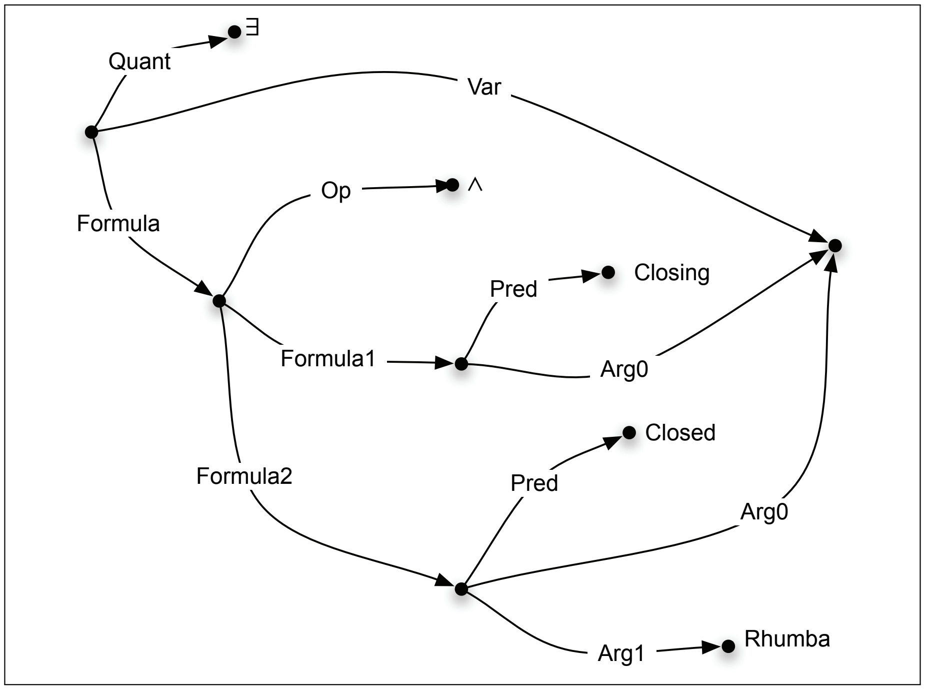


Figure 18.9 A directed graph notation for semantic feature structures.

Our next step is to associate unification equations with the grammar rules involved in this example's derivation. Let's start at the top with the *S* rule.

$$\begin{aligned} & S \rightarrow NP\ VP \\ & \quad \langle S\ \textsc{sem} \rangle = \langle NP\ \textsc{sem} \rangle \\ & \quad \langle VP\ \textsc{arg0} \rangle = \langle NP\ \textsc{indexvar} \rangle \\ & \quad \langle NP\ \textsc{scope} \rangle = \langle VP\ \textsc{sem} \rangle \end{aligned}$$

The first equation equates the meaning representation of the *NP* (encoded under the SEM feature) with our top-level *S*. The second equation assigns the subject *NP* to the appropriate role inside the *VP*'s meaning representation. More concretely, the second equation fills the appropriate role in the *VP*'s semantic representation by unifying the ARG0 feature with a path that leads to a representation of the semantics of the *NP*. Finally, the third equation unifies the SCOPE feature in the *NP*'s meaning representation with a pointer to the VP's meaning representation. As we show, this is a somewhat convoluted way to bring the representation of an event up to where it belongs in the representation. The motivation for this apparatus should become clear in the ensuing discussion when we consider quantified noun phrases.

Carrying on, let's consider the attachments for the *NP* and *ProperNoun* parts of this derivation.

$$
\begin{aligned}
&NP \rightarrow ProperNoun\\
&\quad \langle NP\ \text{SEM}\rangle = \langle ProperNoun\ \text{SEM}\rangle\\
&\quad \langle NP\ \text{SCOPE}\rangle = \langle \text{ProperNoun SCOPE}\rangle\\
&\quad \langle NP\ \text{INDEXVAR}\rangle = \langle ProperNoun\ \text{INDEXVAR}\rangle
\end{aligned}
$$

$$
\begin{aligned}
&ProperNoun \rightarrow Rhumba\\
&\quad \langle ProperNoun\ \text{SEM PRED}\rangle = \text{RHUMBA}\\
&\quad \langle ProperNoun\ \text{INDEXVAR}\rangle = \langle ProperNoun\ \text{SEM PRED}\rangle
\end{aligned}
$$

As we saw earlier, there isn't much to the semantics of proper nouns in this approach. Here we're just introducing a constant and providing an index variable to point at that constant.

Next, let's move on to the semantic attachments for the *VP* and *Verb* rules.

$$
\begin{aligned}
&VP \rightarrow Verb\\
&\quad \langle VP\ \text{SEM}\rangle = \langle \text{Verb SEM}\rangle\\
&\quad \langle VP\ \text{ARG0}\rangle = \langle \text{Verb ARG0}\rangle
\end{aligned}
$$

$$
\begin{aligned}
&Verb \rightarrow closed\\
&\quad \langle Verb\ \text{SEM QUANT}\rangle = \exists\\
&\quad \langle Verb\ \text{SEM FORMULA OP}\rangle = \wedge\\
&\quad \langle Verb\ \text{SEM FORMULA FORMULA1 PRED}\rangle = \text{CLOSING}\\
&\quad \langle Verb\ \text{SEM FORMULA FORMULA1 ARG0}\rangle = \langle Verb\ \text{SEM VAR}\rangle\\
&\quad \langle Verb\ \text{SEM FORMULA FORMULA2 PRED}\rangle = \text{CLOSED}\\
&\quad \langle Verb\ \text{SEM FORMULA FORMULA2 ARG0}\rangle = \langle Verb\ \text{SEM VAR}\rangle\\
&\quad \langle Verb\ \text{SEM FORMULA FORMULA2 ARG1}\rangle = \langle Verb\ \text{ARG0}\rangle
\end{aligned}
$$

The attachments for the *VP* rule parallel our earlier treatment of non-branching grammatical rules. These unification equations are simply making the appropriate semantic fragments of the *Verb* available at the *VP* level. In contrast, the unification equations for the *Verb* introduce the bulk of the event representation that is at the core of this example. Specifically, it introduces the quantifier, event variable, and predications that make up the body of the final expression. What would be an event variable in FOL is captured by the equations unifying the *Verb* SEM VAR path with the appropriate arguments to the predicates in the body of the formula. Finally, it exposes the single missing argument (the entity being closed) through the ⟨*Verb* ARG0⟩ equation.

Taking a step back, we can see that these equations serve the same basic functions as the λ-expressions in Section 18.2; they provide the content of the FOL formula being created, and they serve to expose and name the external arguments that will be filled in later at higher levels in the grammar.

These last few rules also display the division of labor that we've seen several times now; lexical rules introduce the bulk of the semantic content, and higher level grammatical rules assemble the pieces in the right way, rather than introducing content.

Of course, as was the case with the λ-based approach, things get quite a bit more complex when we look at expressions containing quantifiers. To see this, let's work through the following example.

(18.28) Every restaurant closed.

Again, the meaning representation for this expression should be the following:

$$\forall x\, Restaurant(x) \Rightarrow (\exists e\, Closing(e) \wedge Closed(e,x))$$

which is captured by the following feature structure.

$$\begin{bmatrix} \text{QUANT} & \forall \\ \text{VAR} & \boxed{1} \\ \text{FORMULA} & \begin{bmatrix} \text{OP} & \Rightarrow \\ \text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{RESTAURANT}\\ \text{ARG0} & \boxed{1}\end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix} \text{QUANT} & \text{EXISTS}\\ \text{VAR} & \boxed{2} \\ \text{FORMULA} & \begin{bmatrix} \text{OP} & \wedge \\ \text{FORMULA1} & \begin{bmatrix}\text{PRED} & \text{CLOSING}\\ \text{ARG0} & \boxed{2}\end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix}\text{PRED} & \text{CLOSED}\\ \text{ARG0} & \boxed{2}\\ \text{ARG1} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

As we saw earlier with the λ-based approach, the outer structure for expressions like this comes largely from the subject noun phrase. Recall that schematically this semantic structure has the form $\forall x P(x) \Rightarrow Q(x)$, where the P expression is traditionally referred to as the *restrictor* and is provided by the head noun, and Q is referred to as the *nuclear scope* and comes from the verb phrase.

This structure gives rise to two distinct tasks for our semantic attachments: the semantics of the *VP* semantics must be unified with the nuclear scope of the subject noun phrase, and the variable representing that noun phrase must be assigned to the ARG1 role of the CLOSED predicate in the event structure. The following rules involved in the derivation of *Every restaurant* address these two tasks:

$NP \rightarrow Det\ Nominal$

$\langle NP\ \text{SEM}\rangle = \langle Det\ \text{SEM}\rangle$

$\langle NP\ \text{SEM VAR}\rangle = \langle NP\ \text{INDEXVAR}\rangle$

$\langle NP\ \text{SEM FORMULA FORMULA1}\rangle = \langle Nominal\ \text{SEM}\rangle$

$\langle NP\ \text{SEM FORMULA FORMULA2}\rangle = \langle NP\ \text{SCOPE}\rangle$

$$\begin{array}{l} \textit{Nominal} \rightarrow \textit{Noun} \\ \quad \langle\, \textit{Nominal}\ \text{SEM}\,\rangle = \langle\, \textit{Noun}\ \text{SEM}\,\rangle \\ \quad \langle\, \textit{Nominal}\ \text{INDEXVAR}\,\rangle = \langle\, \textit{Noun}\ \text{INDEXVAR}\,\rangle \end{array}$$

$$\begin{array}{l} \textit{Noun} \rightarrow \textit{restaurant} \\ \quad \langle\, \textit{Noun}\ \text{SEM PRED}\,\rangle = \langle\, \text{RESTAURANT}\,\rangle \\ \quad \langle\, \textit{Noun}\ \text{INDEXVAR}\,\rangle = \langle\, \textit{Noun}\ \text{SEM PRED}\,\rangle \end{array}$$

$$\begin{array}{l} \textit{Det} \rightarrow \textit{every} \\ \quad \langle\, \textit{Det}\ \text{SEM QUANT}\,\rangle = \forall \\ \quad \langle\, \textit{Det}\ \text{SEM FORMULA OP}\,\rangle = \Rightarrow \end{array}$$

As one final exercise, let's walk through an example with a transitive verb phrase.

(18.29) Franco opened a restaurant.

This example has the following meaning representation.

$$\exists x\, Restaurant(x) \wedge \exists e\, Opening(e) \wedge Opener(e, Franco) \wedge Opened(e, x)$$

$$\begin{bmatrix} \text{QUANT} & \text{EXISTS} \\ \text{VAR} & \boxed{1} \\ \text{FORMULA} & \begin{bmatrix} \text{OP} & \wedge \\ \text{FORMULA1} & \begin{bmatrix} \text{PRED} & \text{RESTAURANT} \\ \text{ARG1} & \boxed{1} \end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix} \text{QUANT} & \exists \\ \text{VAR} & \boxed{2} \\ \text{FORMULA} & \begin{bmatrix} \text{OP} & \wedge \\ \text{FORMULA1} & \begin{bmatrix} \text{PRED} & \text{OPENING} \\ \text{ARG0} & \boxed{2} \end{bmatrix} \\ \text{FORMULA2} & \begin{bmatrix} \text{PRED} & \text{OPENER} \\ \text{ARG0} & \boxed{2} \\ \text{ARG1} & \text{FRANCO} \end{bmatrix} \\ \text{FORMULA3} & \begin{bmatrix} \text{PRED} & \text{OPENED} \\ \text{ARG0} & \boxed{2} \\ \text{ARG1} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The only really new element that we need to address in this example is the following transitive *VP* rule.

$$\begin{array}{l} \textit{VP} \rightarrow \textit{Verb NP} \\ \quad \langle \textit{VP}\ \text{SEM}\rangle = \langle \textit{Verb}\ \text{SEM}\rangle \\ \quad \langle \textit{NP}\ \text{SCOPE}\rangle = \langle \textit{VP}\ \text{SEM}\rangle \\ \quad \langle \textit{Verb}\ \text{ARG1}\rangle = \langle \textit{NP}\ \text{INDEXVAR}\rangle \end{array}$$

This rule has the two primary tasks that parallel those in our *S* rule: it has to fill the nuclear scope of the object *NP* with the semantics of the *VP*, and it has to insert the variable representing the object into the right role in the *VP*'s meaning representation.

One obvious problem with the approach we just described is that it fails to generate all the possible ambiguous representations arising from quantifier scope ambiguities. Fortunately, the approaches to underspecification described earlier in Section 18.3 can be adapted to the unification-based approach.

18.5 Integration of Semantics into the Earley Parser

In Section 18.1, we suggested a simple pipeline architecture for a semantic analyzer where the results of a complete syntactic parse are passed to a semantic analyzer. The motivation for this notion stems from the fact that the compositional approach requires the syntactic parse before it can proceed. It is, however, also possible to perform semantic analysis in parallel with syntactic processing. This is possible because in our compositional framework, the meaning representation for a constituent can be created as soon as all of its constituent parts are present. This section describes just such an approach to integrating semantic analysis into the Earley parser from Chapter 13.

The integration of semantic analysis into an Earley parser is straightforward and follows precisely the same lines as the integration of unification into the algorithm given in Chapter 15. Three modifications are required to the original algorithm:

1. The rules of the grammar are given a new field to contain their semantic attachments.
2. The states in the chart are given a new field to hold the meaning representation of the constituent.
3. The ENQUEUE function is altered so that when a complete state is entered into the chart, its semantics are computed and stored in the state's semantic field.

Figure 18.10 shows ENQUEUE modified to create meaning representations. When ENQUEUE is passed a complete state that can successfully unify its unification constraints, it calls APPLY-SEMANTICS to compute and store the meaning representation for this state. Note the importance of performing feature-structure unification prior to semantic analysis. This ensures that semantic analysis will be performed only on valid trees and that features needed for semantic analysis will be present.

The primary advantage of this integrated approach over the pipeline approach lies in the fact that APPLY-SEMANTICS can fail in a manner similar to the way that unification can fail. If a semantic ill-formedness is found in the meaning representation being created, the corresponding state can be blocked from entering the chart. In this way, semantic considerations can be brought to bear during syntactic processing. Chapter 19 describes in some detail the various ways by which this notion of ill-formedness can be realized.

Unfortunately, this also illustrates one of the primary disadvantages of integrating semantics directly into the parser—considerable effort may be spent on the semantic analysis of *orphan* constituents that do not in the end contribute to a successful parse.

```
procedure ENQUEUE(state, chart-entry)
 if INCOMPLETE?(state) then
     if state is not already in chart-entry then
         PUSH(state, chart-entry)
 else if UNIFY-STATE(state) succeeds then
     if APPLY-SEMANTICS(state) succeeds then
       if state is not already in chart-entry then
         PUSH(state, chart-entry)

procedure APPLY-SEMANTICS(state)
 meaning-rep ← APPLY(state.semantic-attachment, state)
 if meaning-rep does not equal failure then
   state.meaning-rep ← meaning-rep
```

Figure 18.10 The ENQUEUE function modified to handle semantics. If the state is complete and unification succeeds, then ENQUEUE calls APPLY-SEMANTICS to compute and store the meaning representation of completed states.

The question of whether the gains made by bringing semantics to bear early in the process outweigh the costs involved in performing extraneous semantic processing can only be answered on a case-by-case basis.

18.6 Idioms and Compositionality

Ce corps qui s'appelait et qui s'appelle encore le saint empire romain n'était en aucune manière ni saint, ni romain, ni empire.

This body, which called itself and still calls itself the Holy Roman Empire, was neither Holy, nor Roman, nor an Empire.

Voltaire[2] (1756)

As innocuous as it seems, the principle of compositionality runs into trouble fairly quickly when real language is examined. There are many cases in which the meaning of a constituent is not based on the meaning of its parts, at least not in the straightforward compositional sense. Consider the following WSJ examples:

(18.30) Coupons are just the tip of the iceberg.

(18.31) The SEC's allegations are only the tip of the iceberg.

(18.32) Coronary bypass surgery, hip replacement and intensive-care units are but the tip of the iceberg.

The phrase *the tip of the iceberg* in each of these examples clearly doesn't have much to do with tips or icebergs. Instead, it roughly means something like *the beginning*. The most straightforward way to handle idiomatic constructions like these is to

2 *Essai sur les moeurs et les esprit des nations.* Translation by Y. Sills, as quoted in Sills and Merton (1991).

introduce new grammar rules specifically designed to handle them. These idiomatic rules mix lexical items with grammatical constituents and introduce semantic content that is not derived from any of its parts. Consider the following rule as an example of this approach:

$$\begin{aligned} NP &\rightarrow \textit{the tip of the iceberg} \\ &\{Beginning\} \end{aligned}$$

The lower-case items on the right-hand side of this rule represent precisely the words in the input. Although, the constant *Beginning* should not be taken too seriously as a meaning representation for this idiom, it does illustrate the idea that the meaning of this idiom is not based on the meaning of any of its parts. Note that an Earley-style analyzer with this rule will produce two parses when this phrase is encountered: one representing the idiom and one representing the compositional meaning.

As with the rest of the grammar, it may take a few tries to get these rules right. Consider the following *iceberg* examples from the WSJ corpus:

(18.33) And that's but the tip of Mrs. Ford's iceberg.

(18.34) These comments describe only the tip of a 1,000-page iceberg.

(18.35) The 10 employees represent the merest tip of the iceberg.

The rule given above is clearly not general enough to handle these cases. These examples indicate that there is a vestigial syntactic structure to this idiom that permits some variation in the determiners used and also permits some adjectival modification of both the *iceberg* and the *tip*. A more promising rule would be something like the following:

$$\begin{aligned} NP &\rightarrow \textit{TipNP of IcebergNP} \\ &\{Beginning\} \end{aligned}$$

Here, the categories *TipNP* and *IcebergNP* can be given an internal nominal-like structure that permits some adjectival modification and some variation in the determiners while still restricting the heads of these noun phrases to the lexical items *tip* and *iceberg*. Note that this syntactic solution ignores the thorny issue that the modifiers *mere* and *1000-page* seem to indicate that both the *tip* and *iceberg* may in fact play some compositional role in the meaning of the idiom. We return to this topic in Chapter 19, when we take up the issue of metaphor.

To summarize, handling idioms requires at least the following changes to the general compositional framework:

- Allow the mixing of lexical items with traditional grammatical constituents.
- Allow the creation of additional idiom-specific constituents to handle the correct range of productivity of the idiom.
- Permit semantic attachments that introduce logical terms and predicates that are not related to any of the constituents of the rule.

This discussion is obviously only the tip of an enormous iceberg. Idioms are far more frequent and far more productive than is generally recognized and pose serious difficulties for many applications, including, as we discuss in Chapter 25, machine translation.

18.7 Summary

This chapter explores the notion of syntax-driven semantic analysis. Among the highlights of this chapter are the following topics:

- **Semantic analysis** is the process whereby meaning representations are created and assigned to linguistic inputs.
- **Semantic analyzers** that make use of static knowledge from the lexicon and grammar can create context-independent literal or conventional meanings.
- The **Principle of Compositionality** states that the meaning of a sentence can be composed from the meanings of its parts.
- In **syntax-driven semantic analysis**, the parts are the syntactic constituents of an input.
- Compositional creation of FOL formulas is possible with a few notational extensions, including **λ-expressions** and **complex terms**.
- Compositional creation of FOL formulas is also possible with the mechanisms provided by feature structures and unification.
- **Natural language quantifiers** introduce a kind of ambiguity that is difficult to handle compositionally.
- **Underspecified representations** can be used to effectively handle multiple interpretations arising from by scope ambiguities.
- **Idiomatic language** defies the principle of compositionality but can easily be handled by the adaptation of techniques used to design grammar rules and their semantic attachments.

Bibliographical and Historical Notes

As noted earlier, the principle of compositionality is traditionally attributed to Frege; Janssen (1997) discusses this attribution. Using the categorial grammar framework we describe in Chapter 12, Montague (1973) demonstrated that a compositional approach could be systematically applied to an interesting fragment of natural language. The rule-to-rule hypothesis was first articulated by Bach (1976). On the computational side of things, Woods's LUNAR system (Woods, 1977) was based on a pipelined syntax-first compositional analysis. Schubert and Pelletier (1982) developed an incremental rule-to-rule system based on Gazdar's GPSG approach (Gazdar, 1981, 1982; Gazdar et al., 1985). Main and Benson (1983) extended Montague's approach to the domain of question answering.

In one of the all-too-frequent cases of parallel development, researchers in programming languages developed essentially identical compositional techniques to aid in the design of compilers. Specifically, Knuth (1968) introduced the notion of attribute grammars that associate semantic structures with syntactic structures in a one-to-one correspondence. As a consequence, the style of semantic attachments used in this chapter will be familiar to users of the YACC-style (Johnson and Lesk, 1978) compiler tools.

Of necessity, a large number of important topics were not covered in this chapter. See Alshawi (1992) for the standard gap-threading approach to semantic interpretation in the presence of long-distance dependencies. ter Meulen (1995) presents a modern treatment of tense, aspect, and the representation of temporal information. Many limited-domain systems such as dialogue agents make use of an alternative approach to syntax-semantics integration called **semantic grammar** (Brown and Burton, 1975), related to other early models such as pragmatic grammars (Woods, 1977) and performance grammars (Robinson, 1975). All were centered around the notion of reshaping syntactic grammars to serve the needs of semantic processing. See Chapter 24 for details of this approach.

Semantic grammar

The store-and-retrieve approach to quantifier scope problems is due to Cooper (1983). Keller (1988) extended this approach with the introduction of nested stores to deal with a wider range of quantified noun phrases. The hole semantics approach to underspecified representations is due to Bos (1996, 2001, 2004). Blackburn and Bos (2005) provide a detailed description of hole semantics, including how it can be implemented in Prolog and integrated into a semantic analysis system. Other constraint-based approaches include Underspecified Discourse Representation Theory (Reyle, 1993), Minimal Recursion Semantics (MRS) (Copestake et al., 1995), and the Constraint Language for Lambda Structures (CLLS) (Egg et al., 2001). Player (2004) argues that these approaches are, for all practical purposes, notational variants.

Practical computational approaches to quantifier scoping can be found in Hobbs and Shieber (1987) and Alshawi (1992). VanLehn (1978) presents a set of human preferences for quantifier scoping. Higgins and Sadock (2003) use such preferences as features to train a classifier to predict the correct scope for quantifier ambiguities.

Over the years, a considerable amount of effort has been directed toward the interpretation of compound nominals. Linguistic research on this topic can be found in Lees (1970), Downing (1977), Levi (1978), and Ryder (1994); more computational approaches are described in Gershman (1977), Finin (1980), McDonald (1982), Pierre (1984), Arens et al. (1987), Wu (1992), Vanderwende (1994), and Lauer (1995).

The literature on idioms is long and extensive. Fillmore et al. (1988) describe a general grammatical framework called Construction Grammar that places idioms at the center of its underlying theory. Makkai (1972) presents an extensive linguistic analysis of many English idioms. Hundreds of idiom dictionaries for second-language learners are also available. On the computational side, Becker (1975) was among the first to suggest the use of phrasal rules in parsers. Wilensky and Arens (1980) were among the first to successfully make use of this notion in their PHRAN system. Zernik (1987) demonstrated a system that could learn such phrasal idioms in context. A collection of papers on computational approaches to idioms appeared in Fass et al. (1992).

We have neglected an entire branch of semantic analysis in which expectations arising from deep meaning representations drive the analysis process. Such systems avoid the direct representation and use of syntax, rarely making use of anything resembling a parse tree. Some of the earliest and most successful efforts along these lines were developed by Simmons (1973, 1978, 1983) and Wilks (1975a, 1975c). A series of similar approaches was developed by Roger Schank and his students (Riesbeck, 1975; Birnbaum and Selfridge, 1981; Riesbeck, 1986). In these approaches, the semantic analysis process is guided by detailed procedures associated with individual lexical items.

Finally, recent work has focused on the task of automatically learning the mapping from sentences to logical form, based on a training set of sentences labeled with their semantics (Zettlemoyer and Collins, 2005, 2007; Mooney, 2007; Wong and Mooney, 2007).

Exercises

18.1 Develop a set of grammar rules and semantic attachments to handle predicate adjectives such as the following:

1. Flight 308 from New York is expensive.
2. Murphy's restaurant is cheap.

18.2 Develop a set of grammar rules and semantic attachments to handle so-called *control verbs* as in the following:

1. Franco *decided* to leave.
2. Nicolas *told* Franco to go to Frasca.

The first of these is an example of subject control—*Franco* plays the role of the agent for both *decide* and *leave*. The second is an example of object control—there *Franco* is the person being told and the agent of the going event. The challenge in creating attachments for these rules is to properly incorporate the semantic representation of a single noun phrase into two roles.

18.3 None of the attachments given in this chapter provide temporal information. Augment a small number of the most basic rules to add temporal information along the lines sketched in Chapter 17. Use your rules to create meaning representations for the following examples:

1. Flight 299 departed at 9 o'clock.
2. Flight 208 will arrive at 3 o'clock.
3. Flight 1405 will arrive late.

18.4 As noted in Chapter 17, the present tense in English can be used to refer to either the present or the future. However, it can also be used to express habitual behavior, as in the following:

1. Flight 208 leaves at 3 o'clock.

This could be a simple statement about today's Flight 208, or alternatively it might state that this flight leaves at 3 o'clock every day. Create a FOL meaning representation along with appropriate semantic attachments for this habitual sense.

18.5 Implement an Earley-style semantic analyzer based on the discussion on page 604.

18.6 It has been claimed that it is not necessary to explicitly list the semantic attachment for most grammar rules. Instead, the semantic attachment for a rule should be inferable from the semantic types of the rule's constituents. For example, if a rule has two constituents, where one is a single-argument λ-expression and the other is a constant, then the semantic attachment must apply the λ-expression to the constant. Given the attachments presented in this chapter, does this *type-driven semantics* seem like a reasonable idea? Explain your answer.

18.7 Add a simple type-driven semantics mechanism to the Earley analyzer you implemented for Exercise 18.5.

18.8 Using a phrasal search on your favorite Web search engine, collect a small corpus of *the tip of the iceberg* examples. Be certain that you search for an appropriate range of examples (i.e., don't just search for "the tip of the iceberg"). Analyze these examples and come up with a set of grammar rules that correctly accounts for them.

18.9 Collect a similar corpus of examples for the idiom *miss the boat*. Analyze these examples and come up with a set of grammar rules that correctly accounts for them.