Chapter 15
Features and Unification

FRIAR FRANCIS: *If either of you know any inward impediment why you should not be conjoined, charge you, on your souls, to utter it.*
William Shakespeare, *Much Ado About Nothing*

From a reductionist perspective, the history of the natural sciences over the last few hundred years can be seen as an attempt to explain the behavior of larger structures by the combined action of smaller primitives. In biology, the properties of inheritance have been explained by the action of genes, and then again the properties of genes have been explained by the action of DNA. In physics, matter was reduced to atoms and then again to subatomic particles. The appeal of reductionism has not escaped computational linguistics. In this chapter we introduce the idea that grammatical categories like *VPto*, *Sthat*, *Non3sgAux*, or *3sgNP*, as well as the grammatical rules like $S \rightarrow NP\ VP$ that make use of them, should be thought of as *objects* that can have complex sets of *properties* associated with them. The information in these properties is represented by **constraints**, and so these kinds of models are often called **constraint-based formalisms** .

Constraint-based formalisms

Why do we need a more fine-grained way of representing and placing constraints on grammatical categories? One problem arose in Ch. 12, where we saw that naive models of grammatical phenomena such as agreement and subcategorization can lead to overgeneration problems. For example, in order to avoid ungrammatical noun phrases such as *this flights* and verb phrases like *disappeared a flight*, we were forced to create a huge proliferation of primitive grammatical categories such as *Non3sgVPto*, *NPmass*, *3sgNP* and *Non3sgAux*. These new categories led, in turn, to an explosion in the number of grammar rules and a corresponding loss of generality in the grammar. A constraint-based representation scheme will allow us to represent fine-grained information about number and person, agreement, subcategorization, as well as semantic categories like mass/count.

Constraint-based formalisms have other advantages that we will not cover in this chapter, such as the ability to model more complex phenomena than context-free grammars, and the ability to efficiently and conveniently compute semantics for syntactic representations.

Consider briefly how this approach might work in the case of grammatical number. As we saw in Ch. 12, noun phrases like *this flight* and *those flights* can be distinguished based on whether they are singular or plural. This distinction can be captured if we associate a property called NUMBER that can have the value singular or plural, with appropriate members of the *NP* category. Given this ability, we can say that *this flight* is a member of the *NP* category and, in addition, has the value singular for its NUMBER property. This same property can be used in the same way to distinguish singular and plural members of the *VP* category such as *serves lunch* and *serve lunch*.

Of course, simply associating these properties with various words and phrases does not solve any of our overgeneration problems. To make these properties useful, we need the ability to perform simple operations, such as equality tests, on them. By pairing such tests with our core grammar rules, we can add various constraints to help ensure that only grammatical strings are generated by the grammar. For example, we might want to ask whether or not a given noun phrase and verb phrase have the same values for their respective number properties. Such a test is illustrated by the following kind of rule.

$S \rightarrow NP\ VP$

Only if the number of the NP is equal to the number of the VP.

The remainder of this chapter provides the details of one computational implementation of a constraint-based formalism, based on feature structures and unification. The next section describes feature structures, the representation used to capture the kind of grammatical properties we have in mind. Section 15.2 then introduces the unification operator that is used to implement basic operations over feature structures. Section 15.3 then covers the integration of these structures into a grammatical formalism. Section 15.4 then introduces the unification algorithm and its required data structures. Next, Section 15.5 describes how feature structures and the unification operator can be integrated into a parser. Finally, Section 15.6 discusses the most significant extension to this constraint-based formalism, the use of types and inheritance, as well as other extensions.

15.1 Feature Structures

Feature structures

Attribute-value matrix

One of the simplest ways to encode the kind of properties that we have in mind is through the use of **feature structures**. These are simply sets of feature-value pairs, where features are unanalyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures themselves. Such feature structures are illustrated with the following kind of diagram, called an **attribute-value matrix** or AVM:

$$\begin{bmatrix} \text{FEATURE}_1 & value_1 \\ \text{FEATURE}_2 & value_2 \\ \vdots & \\ \text{FEATURE}_n & value_n \end{bmatrix}$$

To make this concrete, consider the number property discussed above. To capture this property, we will use the symbol NUMBER to designate this grammatical attribute, and the symbols *sg* and *pl* (introduced in Ch. 3) to designate the possible values it can take on in English. A simple feature structure consisting of this single feature would then be illustrated as follows:

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

Adding an additional feature-value pair to capture the grammatical notion of person leads to the following feature structure:

$$\begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3rd \end{bmatrix}$$

Next we can encode the grammatical category of the constituent that this structure corresponds to through the use of the CAT feature. For example, we can indicate that these features are associated with a noun phrase by using the following structure:

$$\begin{bmatrix} \text{CAT} & NP \\ \text{NUMBER} & sg \\ \text{PERSON} & 3rd \end{bmatrix}$$

This structure can be used to represent the *3sgNP* category introduced in Ch. 12 to capture a restricted subcategory of noun phrases. The corresponding plural version of this structure would be captured as follows:

$$\begin{bmatrix} \text{CAT} & NP \\ \text{NUMBER} & pl \\ \text{PERSON} & 3rd \end{bmatrix}$$

Note that the values of the CAT and PERSON features remain the same for these last two structures. This illustrates how the use of feature structures allows us to both preserve the core set of grammatical categories and draw distinctions among members of a single category.

As mentioned earlier, features are not limited to atomic symbols as their values; they can also have other feature structures as their values. This is particularly useful when we wish to bundle a set of feature-value pairs together for similar treatment. As an example of this, consider that the NUMBER and PERSON features are often lumped together since grammatical subjects must agree with their predicates in both their number and person properties. This lumping together can be captured by introducing an AGREEMENT feature that takes a feature structure consisting of the NUMBER and PERSON feature-value pairs as its value. Introducing this feature into our third person singular noun phrase yields the following kind of structure.

$$\begin{bmatrix} \text{CAT} & NP \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & sg \\ \text{PERSON} & 3rd \end{bmatrix} \end{bmatrix}$$

Given this kind of arrangement, we can test for the equality of the values for both the NUMBER and PERSON features of two constituents by testing for the equality of their AGREEMENT features.

Feature path

This ability to use feature structures as values leads fairly directly to the notion of a **feature path**. A feature path is nothing more than a sequence of features through a feature structure leading to a particular value. For example, in the last feature structure, we can say that the ⟨AGREEMENT NUMBER⟩ path leads to the value *sg*, while the

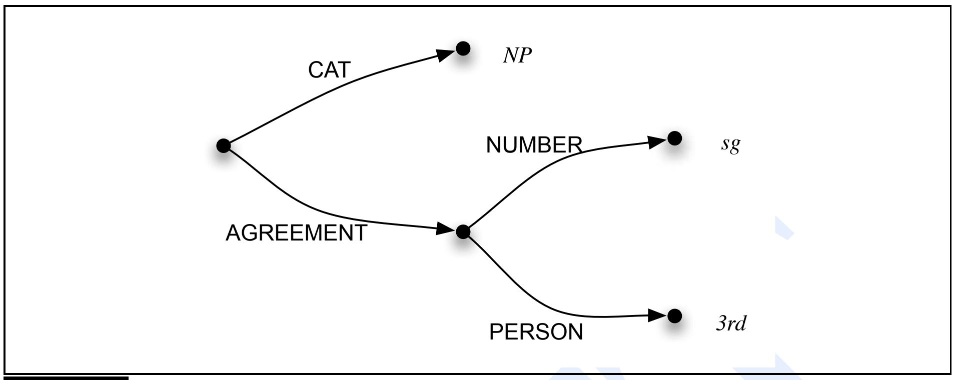

Figure 15.1 A feature structure with shared values. The location (value) found by following the ⟨HEAD SUBJECT AGREEMENT⟩ path is the same as that found via the ⟨HEAD AGREEMENT⟩ path.

⟨AGREEMENT PERSON⟩ path leads to the value *3rd*. This notion of a path leads naturally to an alternative graphical way of illustrating feature structures, shown in Figure 15.1, which as we will see in Section 15.4 is suggestive of how they will be implemented. In these diagrams, feature structures are depicted as directed graphs where features appear as labeled edges and values as nodes.

While this notion of paths will prove useful in a number of settings, we introduce it here to help explain an additional important kind of feature structure: those that contain features that actually share some feature structure as a value. Such feature structures will be referred to as **reentrant structures** structures. What we have in mind here is not the simple idea that two features might have equal values, but rather that they share precisely the same feature structure (or node in the graph). These two cases can be distinguished clearly if we think in terms of paths through a graph. In the case of simple equality, two paths lead to distinct nodes in the graph that anchor identical, but distinct structures. In the case of a reentrant structure, two feature paths actually lead to the same node in the structure.

Reentrant structures

Figure 15.2 illustrates a simple example of reentrancy. In this structure, the ⟨HEAD SUBJECT AGREEMENT⟩ path and the ⟨HEAD AGREEMENT⟩ path lead to the same location. Shared structures like this will be denoted in our AVM diagrams by adding numerical indexes that signal the values to be shared. The AVM version of the feature structure from Figure 15.2 would be denoted as follows, using the notation of the PATR-II system (Shieber, 1986), based on Kay (1979):

$$\begin{bmatrix} \text{CAT} & S \\ \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & sg \\ \text{PERSON} & 3rd\end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

As we will see, these simple structures give us the ability to express linguistic generalizations in surprisingly compact and elegant ways.

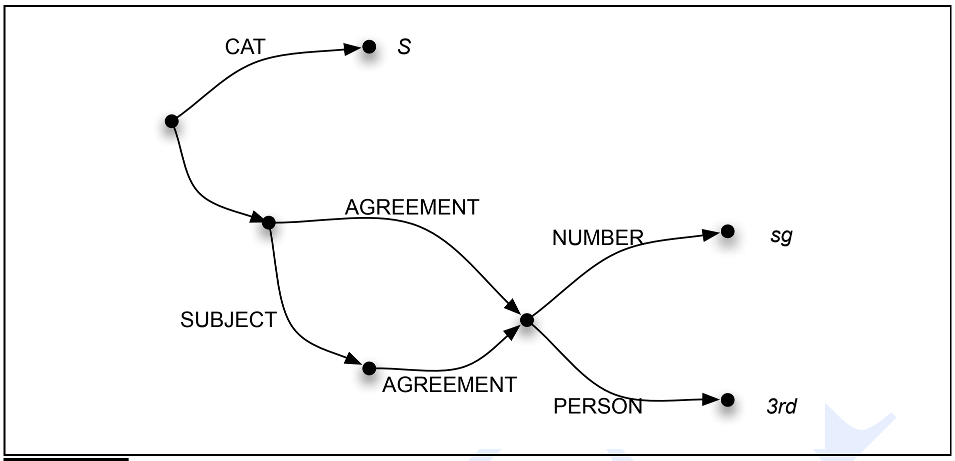

Figure 15.2 A feature structure with shared values. The location (value) found by following the ⟨HEAD SUBJECT AGREEMENT⟩ path is the same as that found via the ⟨HEAD AGREEMENT⟩ path.

15.2 Unification of Feature Structures

As noted earlier, feature structures would be of little use without our being able to perform reasonably efficient and powerful operations on them. As we will show, the two principal operations we need to perform are merging the information content of two structures and rejecting the merger of structures that are incompatible. Fortunately, a single computational technique, called , suffices for both of these purposes. The bulk of this section will illustrate through a series of examples how unification instantiates these notions of merger and compatibility. Discussion of the unification algorithm and its implementation will be deferred to Section 15.4.

We begin with the following simple application of the unification operator.

$$\begin{bmatrix}\text{NUMBER} & sg\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & sg\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & sg\end{bmatrix}$$

As this equation illustrates, unification is a binary operation (represented here as ⊔) that accepts two feature structures as arguments and returns a feature structure when it succeeds. In this example, unification is being used to perform a simple equality check. The unification succeeds because the corresponding NUMBER features in each structure agree as to their values. In this case, since the original structures are identical, the output is the same as the input. The following similar kind of check fails since the NUMBER features in the two structures have incompatible values.

$$\begin{bmatrix}\text{NUMBER} & sg\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & pl\end{bmatrix} \textit{Fails!}$$

This next unification illustrates an important aspect of the notion of compatibility in unification.

$$\begin{bmatrix}\text{NUMBER} & sg\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & []\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & sg\end{bmatrix}$$

In this situation, these features structures are taken to be compatible, and are hence capable of being merged, despite the fact that the given values for the respective NUM-

BER features are different. The [] value in the second structure indicates that the value has been left unspecified. A feature with such a [] value can be successfully matched to any value in a corresponding feature in another structure. Therefore, in this case, the value *sg* from the first structure can match the [] value from the second, and as is indicated by the output shown, the result of this type of unification is a structure with the value provided by the more specific, non-null, value.

The next example illustrates another of the merger aspects of unification.

$$\begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix} \sqcup \begin{bmatrix} \text{PERSON} & \textit{3rd} \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

Here the result of the unification is a merger of the original two structures into one larger structure. This larger structure contains the union of all the information stored in each of the original structures. Although this is a simple example, it is important to understand why these structures are judged to be compatible: they are compatible because they contain no features that are explicitly incompatible. The fact that they each contain a feature-value pair that the other does not is not a reason for the unification to fail.

We will now consider a series of cases involving the unification of somewhat more complex reentrant structures. The following example illustrates an equality check complicated by the presence of a reentrant structure in the first argument.

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \textit{3rd} \\ \text{NUMBER} & \textit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

The important elements in this example are the SUBJECT features in the two input structures. The unification of these features succeeds because the values found in the first argument by following the $\boxed{1}$ numerical index, match those that are directly present in the second argument. Note that, by itself, the value of the AGREEMENT feature in the first argument would have no bearing on the success of unification since the second argument lacks an AGREEMENT feature at the top level. It only becomes relevant because the value of the AGREEMENT feature is shared with the SUBJECT feature.

The following example illustrates the copying capabilities of unification.

(15.1) $$\begin{bmatrix} \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & \mathit{3rd} \\ \text{NUMBER} & \mathit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{PERSON} & \mathit{3rd} \\ \text{NUMBER} & \mathit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Here the value found via the second argument's ⟨SUBJECT AGREEMENT⟩ path is copied over to the corresponding place in the first argument. In addition, the AGREEMENT feature of the first argument receives a value as a side-effect of the index linking it to the value at the end of the ⟨SUBJECT AGREEMENT⟩ path.

The next example demonstrates the important difference between features that actually share values versus those that merely have identical looking values.

(15.2)
$$\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathit{sg} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \mathit{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathit{sg} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathit{sg} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The values at the end of the ⟨SUBJECT AGREEMENT⟩ path and the ⟨AGREEMENT⟩ path are the same, but not shared, in the first argument. The unification of the SUBJECT features of the two arguments adds the PERSON information from the second argument to the result. However, since there is no index linking the AGREEMENT feature to the ⟨SUBJECT AGREEMENT⟩ path, this information is not added to the value of the AGREEMENT feature.

Finally, consider the following example of a failure to unify.

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1} \begin{bmatrix} \text{NUMBER} & \mathit{sg} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\sqcup \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \mathit{sg} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Fails!

Proceeding through the features in order, we first find that the AGREEMENT features in these examples successfully match. However, when we move on to the SUBJECT features, we find that the values found at the respective $\langle$ SUBJECT AGREEMENT NUMBER $\rangle$ paths differ, causing a unification failure.

Feature structures are a way of representing partial information about some linguistic object or placing informational constraints on what the object can be. Unification can be seen as a way of merging the information in each feature structure, or describing objects which satisfy both sets of constraints. Intuitively, unifying two feature structures produces a new feature structure which is more specific (has more information) than, or is identical to, either of the input feature structures. We say that a less specific (more abstract) feature structure **subsumes** an equally or more specific one. Subsumption is represented by the operator $\sqsubseteq$. A feature structure F subsumes a feature structure G ($F \sqsubseteq G$) if and only if:

Subsumes

1. For every feature x in F, $F(x) \sqsubseteq G(x)$ (where $F(x)$ means "the value of the feature x of feature structure F").
2. For all paths p and q in F such that $F(p) = F(q)$, it is also the case that $G(p) = G(q)$.

For example, consider these feature structures:

(15.3) $\begin{bmatrix}\text{NUMBER} & sg\end{bmatrix}$

(15.4) $\begin{bmatrix}\text{PERSON} & 3\end{bmatrix}$

(15.5) $\begin{bmatrix}\text{NUMBER} & sg \\ \text{PERSON} & 3\end{bmatrix}$

(15.6) $\begin{bmatrix}\text{CAT} & \text{VP} \\ \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}$

(15.7) $\begin{bmatrix}\text{CAT} & \text{VP} \\ \text{AGREEMENT} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{PERSON} & 3 \\ \text{NUMBER} & sg\end{bmatrix}\end{bmatrix}\end{bmatrix}$

The following subsumption relations hold among them:

$$15.3 \sqsubseteq 15.5$$
$$15.4 \sqsubseteq 15.5$$
$$15.6 \sqsubseteq 15.7$$

Subsumption is a partial ordering; there are pairs of feature structures that neither subsume nor are subsumed by each other:

$$15.3 \not\sqsubseteq 15.4$$

$$15.4 \not\sqsubseteq 15.3$$

Since every feature structure is subsumed by the empty structure [], the relation among feature structures can be defined as a **semilattice**. Unification can be defined in terms of the subsumption semilattice. Given two feature structures F and G, $F \sqcup G$ is defined as the most general feature structure H such that $F \sqsubseteq H$ and $G \sqsubseteq H$. Since the information ordering defined by subsumption is a semilattice, the unification operation is **monotonic** (Pereira and Shieber, 1984; Rounds and Kasper, 1986; Moshier, 1988). This means that if some description is true of a feature structure, unifying it with another feature structure results in a feature structure that still satisfies the original description. The unification operation is therefore associative; given a finite set of feature structures to unify, we can check them in any order and get the same result.

Semilattice

Monotonic

To summarize, unification is a way of implementing the integration of knowledge from different constraints. Given two compatible feature structures as input, it produces the most general feature structure which nonetheless contains all the information in the inputs. Given two incompatible feature structures, it fails.

15.3 Feature Structures in the Grammar

Our primary purpose in introducing feature structures and unification has been to provide a way to elegantly express syntactic constraints that would be difficult to express using the mechanisms of context-free grammars alone. Our next step, therefore, is to specify a way to integrate feature structures and unification operations into the specification of a grammar. This can be accomplished by *augmenting* the rules of ordinary context-free grammars with attachments that specify feature structures for the constituents of the rules, along with appropriate unification operations that express constraints on those constituents. From a grammatical point of view, these attachments will be used to accomplish the following goals:

- to associate complex feature structures with both lexical items and instances of grammatical categories
- to guide the composition of feature structures for larger grammatical constituents based on the feature structures of their component parts
- to enforce compatibility constraints between specified parts of grammatical constructions

We will use the following notation to denote the grammar augmentations that will allow us to accomplish all of these goals, based on the PATR-II system described in Shieber (1986):

$$\begin{array}{c} \beta_0 \rightarrow \beta_1 \cdots \beta_n \\ \{set\ of\ constraints\} \end{array}$$

The specified constraints have one of the following forms.

$$\begin{aligned} \langle \beta_i\ feature\ path \rangle &= Atomic\ value \\ \langle \beta_i\ feature\ path \rangle &= \langle \beta_j\ feature\ path \rangle \end{aligned}$$

The notation $\langle \beta_i$ *feature path*$\rangle$ denotes a feature path through the feature structure associated with the β_i component of the context-free part of the rule. The first style of constraint specifies that the value found at the end of the given path must unify with the specified atomic value. The second form specifies that the values found at the end of the two given paths must be unifiable.

To illustrate the use of these constraints, let us return to the informal solution to the number agreement problem proposed at the beginning of this chapter.

$S \rightarrow NP\ VP$

Only if the number of the NP is equal to the number of the VP.

Using the new notation, this rule can now be expressed as follows.

$$S \rightarrow NP\ VP$$
$$\langle NP\ \text{NUMBER}\rangle = \langle VP\ \text{NUMBER}\rangle$$

Note that in cases where there are two or more constituents of the same syntactic category in a rule, we will subscript the constituents to keep them straight, as in $VP \rightarrow V\ NP_1\ NP_2$.

Taking a step back from the notation, it is important to note that in this approach the simple generative nature of context-free rules has been fundamentally changed by this augmentation. Ordinary context-free rules are based on the simple notion of concatenation; an *NP* followed by a *VP* is an *S*, or generatively, to produce an *S* all we need to do is concatenate an *NP* to a *VP*. In the new scheme, this concatenation must be accompanied by a successful unification operation. This leads naturally to questions about the computational complexity of the unification operation and its effect on the generative power of this new grammar. These issues will be discussed in Ch. 15.

To review, there are two fundamental components to this approach.

- The elements of context-free grammar rules will have feature-based constraints associated with them. This reflects a shift from atomic grammatical categories to more complex categories with properties.
- The constraints associated with individual rules can make reference to the feature structures associated with the parts of the rule to which they are attached.

The following sections present applications of unification constraints to four interesting linguistic phenomena: agreement, grammatical heads, subcategorization, and long-distance dependencies.

15.3.1 Agreement

As discussed in Ch. 12, agreement phenomena show up in a number of different places in English. This section illustrates how unification can be used to capture the two main types of English agreement phenomena: subject-verb agreement and determiner-nominal agreement. We will use the following ATIS sentences as examples throughout this discussion to illustrate these phenomena.

(15.8) This flight serves breakfast.

(15.9) Does this flight serve breakfast?

(15.10) Do these flights serve breakfast?

Notice that the constraint used to enforce SUBJECT-VERB agreement given above is deficient in that it ignores the PERSON feature. The following constraint which makes use of the AGREEMENT feature takes care of this problem.

$$\begin{array}{l} S \longrightarrow NP\ VP \\ \quad \langle NP\ \text{AGREEMENT} \rangle = \langle VP\ \text{AGREEMENT} \rangle \end{array}$$

Examples 15.9 and 15.10 illustrate a minor variation on SUBJECT-VERB agreement. In these yes-no-questions, the subject *NP* must agree with the auxiliary verb, rather than the main verb of the sentence, which appears in a non-finite form. This agreement constraint can be handled by the following rule.

$$\begin{array}{l} S \longrightarrow Aux\ NP\ VP \\ \quad \langle Aux\ \text{AGREEMENT} \rangle = \langle NP\ \text{AGREEMENT} \rangle \end{array}$$

Agreement between determiners and nominals in noun phrases is handled in a similar fashion. The basic task is to permit the expressions given above, but block the unwanted **this flights* and **those flight* expressions where the determiners and nominals clash in their NUMBER feature. Again, the logical place to enforce this constraint is in the grammar rule that brings the parts together.

$$\begin{array}{l} NP \longrightarrow Det\ Nominal \\ \quad \langle Det\ \text{AGREEMENT} \rangle = \langle Nominal\ \text{AGREEMENT} \rangle \\ \quad \langle NP\ \text{AGREEMENT} \rangle = \langle Nominal\ \text{AGREEMENT} \rangle \end{array}$$

This rule states that the AGREEMENT feature of the *Det* must unify with the AGREEMENT feature of the *Nominal*, and moreover, that the AGREEMENT feature of the *NP* must also unify with the *Nominal*.

Having expressed the constraints needed to enforce subject-verb and determiner-nominal agreement, we must now fill in the rest of the machinery needed to make these constraints work. Specifically, we must consider how the various constituents that take part in these constraints (the *Aux*, *VP*, *NP*, *Det*, and *Nominal*) acquire values for their various agreement features.

We can begin by noting that our constraints involve both lexical and non-lexical constituents. The simpler lexical constituents, *Aux* and *Det*, receive values for their respective agreement features directly from the lexicon as in the following rules.

$$\begin{array}{l} Aux \longrightarrow do \\ \quad \langle Aux\ \text{AGREEMENT NUMBER} \rangle = pl \\ \quad \langle Aux\ \text{AGREEMENT PERSON} \rangle = 3rd \end{array}$$

$$\begin{array}{l} Aux \longrightarrow does \\ \quad \langle Aux\ \text{AGREEMENT NUMBER} \rangle = sg \\ \quad \langle Aux\ \text{AGREEMENT PERSON} \rangle = 3rd \end{array}$$

$$\begin{array}{l} Det \rightarrow this \\ \quad \langle Det \text{ AGREEMENT NUMBER} \rangle = sg \end{array}$$

$$\begin{array}{l} Det \rightarrow these \\ \quad \langle Det \text{ AGREEMENT NUMBER} \rangle = pl \end{array}$$

Returning to our first *S* rule, let us first consider the AGREEMENT feature for the *VP* constituent. The constituent structure for this *VP* is specified by the following rule.

$$VP \rightarrow Verb\ NP$$

It seems clear that the agreement constraint for this constituent must be based on its constituent verb. This verb, as with the previous lexical entries, can acquire its agreement feature values directly from lexicon as in the following rules.

$$\begin{array}{l} Verb \rightarrow serve \\ \quad \langle Verb \text{ AGREEMENT NUMBER} \rangle = pl \end{array}$$

$$\begin{array}{l} Verb \rightarrow serves \\ \quad \langle Verb \text{ AGREEMENT NUMBER} \rangle = sg \\ \quad \langle Verb \text{ AGREEMENT PERSON} \rangle = 3rd \end{array}$$

All that remains is to stipulate that the agreement feature of the parent *VP* is constrained to be the same as its verb constituent.

$$\begin{array}{l} VP \rightarrow Verb\ NP \\ \quad \langle VP \text{ AGREEMENT} \rangle = \langle Verb \text{ AGREEMENT} \rangle \end{array}$$

In other words, non-lexical grammatical constituents can acquire values for at least some of their features from their component constituents.

The same technique works for the remaining *NP* and *Nominal* categories. The values for the agreement features for these categories are derived from the nouns *flight* and *flights*.

$$\begin{array}{l} Noun \rightarrow flight \\ \quad \langle Noun \text{ AGREEMENT NUMBER} \rangle = sg \end{array}$$

$$\begin{array}{l} Noun \rightarrow flights \\ \quad \langle Noun \text{ AGREEMENT NUMBER} \rangle = pl \end{array}$$

Nominal features can be constrained to have the same values as their constituent nouns.

$$\begin{array}{l} Nominal \rightarrow Noun \\ \quad \langle Nominal \text{ AGREEMENT} \rangle = \langle Noun \text{ AGREEMENT} \rangle \end{array}$$

Note that this section has only scratched the surface of the English agreement system, and that the agreement system of other languages can be considerably more complex than English.

15.3.2 Head Features

To account for the way that compositional grammatical constituents such as noun phrases, nominals, and verb phrases come to have agreement features, the preceding section introduced the notion of copying feature structures from phrase structure children to their parents. This turns out to be a specific instance of a much more general phenomenon in constraint-based grammars. Specifically, the features for most grammatical categories are copied from *one* of the children to the parent. The child that provides the features is called the head of the phrase and the features copied are referred to as **head features**.

Head features

This notion of heads, first introduced in Sec. 12.4.4, plays an important role in constraint-based grammars. Consider the following three rules from the last section.

$$VP \rightarrow Verb\ NP$$
$$\langle VP\ \text{AGREEMENT}\rangle = \langle Verb\ \text{AGREEMENT}\rangle$$

$$NP \rightarrow Det\ Nominal$$
$$\langle Det\ \text{AGREEMENT}\rangle = \langle Nominal\ \text{AGREEMENT}\rangle$$
$$\langle NP\ \text{AGREEMENT}\rangle = \langle Nominal\ \text{AGREEMENT}\rangle$$

$$Nominal \rightarrow Noun$$
$$\langle Nominal\ \text{AGREEMENT}\rangle = \langle Noun\ \text{AGREEMENT}\rangle$$

In each of these rules, the constituent providing the agreement feature structure to its parent is the head of the phrase. More specifically, the verb is the head of the verb phrase, the nominal is the head of the noun phrase, and the noun is the head of the nominal. As a result, we can say that the agreement feature structure is a head feature. We can rewrite our rules to reflect these generalizations by placing the agreement feature structure under a HEAD feature and then copying that feature upward as in the following constraints.

(15.11)
$$VP \rightarrow Verb\ NP$$
$$\langle VP\ \text{HEAD}\rangle = \langle Verb\ \text{HEAD}\rangle$$

(15.12)
$$NP \rightarrow Det\ Nominal$$
$$\langle NP\ \text{HEAD}\rangle = \langle Nominal\ \text{HEAD}\rangle$$
$$\langle Det\ \text{HEAD AGREEMENT}\rangle = \langle Nominal\ \text{HEAD AGREEMENT}\rangle$$

(15.13)
$$Nominal \rightarrow Noun$$
$$\langle Nominal\ \text{HEAD}\rangle = \langle Noun\ \text{HEAD}\rangle$$

Similarly, the lexical entries that introduce these features must now reflect this HEAD notion, as in the following.

$$\begin{aligned}&\textit{Noun} \rightarrow \textit{flights}\\&\quad\langle \textit{Noun}\ \text{HEAD AGREEMENT NUMBER}\rangle = \textit{pl}\end{aligned}$$

$$\begin{aligned}&\textit{Verb} \rightarrow \textit{serves}\\&\quad\langle \textit{Verb}\ \text{HEAD AGREEMENT NUMBER}\rangle = \textit{sg}\\&\quad\langle \textit{Verb}\ \text{HEAD AGREEMENT PERSON}\rangle = \textit{3rd}\end{aligned}$$

15.3.3 Subcategorization

Recall that subcategorization is the notion that verbs can be picky about the patterns of arguments they will allow themselves to appear with. In Ch. 12, to prevent the generation of ungrammatical sentences with verbs and verb phrases that do not match, we were forced to split the category of verb into multiple sub-categories. These more specific verb categories were then used in the definition of the specific verb phrases that they were allowed to occur with, as in the following.

$$\begin{aligned}&\textit{Verb-with-S-comp} \rightarrow \textit{think}\\&\textit{VP} \rightarrow \textit{Verb-with-S-comp S}\end{aligned}$$

Clearly, this approach introduces exactly the same undesirable proliferation of categories that we saw with the similar approach to solving the number problem. The proper way to avoid this proliferation is to introduce feature structures to distinguish among the various members of the verb category. This goal can be accomplished by associating an atomic feature called SUBCAT, with an appropriate value, with each of the verbs in the lexicon. For example, the transitive version of *serves* could be assigned the following feature structure in the lexicon.

$$\begin{aligned}&\textit{Verb} \rightarrow \textit{serves}\\&\quad\langle \textit{Verb}\ \text{HEAD AGREEMENT NUMBER}\rangle = \textit{sg}\\&\quad\langle \textit{Verb}\ \text{HEAD SUBCAT}\rangle = \textit{trans}\end{aligned}$$

The SUBCAT feature is a signal to the rest of the grammar that this verb should only appear in verb phrases with a single noun phrase argument. This constraint is enforced by adding corresponding constraints to all the verb phrase rules in the grammar, as in the following.

$$\begin{aligned}&\textit{VP} \rightarrow \textit{Verb}\\&\quad\langle \textit{VP}\ \text{HEAD}\rangle = \langle \textit{Verb}\ \text{HEAD}\rangle\\&\quad\langle \textit{VP}\ \text{HEAD SUBCAT}\rangle = \textit{intrans}\end{aligned}$$

$$\begin{aligned}&\textit{VP} \rightarrow \textit{Verb NP}\\&\quad\langle \textit{VP}\ \text{HEAD}\rangle = \langle \textit{Verb}\ \text{HEAD}\rangle\\&\quad\langle \textit{VP}\ \text{HEAD SUBCAT}\rangle = \textit{trans}\end{aligned}$$

$$\begin{aligned}&VP \rightarrow Verb\ NP\ NP\\&\quad\langle VP\ \textsc{head}\rangle = \langle Verb\ \textsc{head}\rangle\\&\quad\langle VP\ \textsc{head subcat}\rangle = ditrans\end{aligned}$$

The first unification constraint in these rules states that the verb phrase receives its HEAD features from its verb constituent, while the second constraint specifies what the value of that SUBCAT feature must be. Any attempt to use a verb with an inappropriate verb phrase will fail since the value of the SUBCAT feature of the *VP* will fail to unify with the atomic symbol given in the second constraint. Note that this approach requires unique symbols for each of the 50–100 verb phrase frames in English.

This is a somewhat clumsy approach since these unanalyzable SUBCAT symbols do not directly encode either the number or type of the arguments that the verb expects to take. To see this, note that one can not simply examine a verb's entry in the lexicon and know what its subcategorization frame is. Rather, you must use the value of the SUBCAT feature indirectly as a pointer to those verb phrase rules in the grammar that can accept the verb in question.

A more elegant solution, which makes better use of the expressive power of feature structures, allows the verb entries to directly specify the order and category type of the arguments they require. The following entry for *serves* is an example of one such approach, in which the verb's subcategory feature expresses a **list** of its objects and complements.

$$\begin{aligned}&Verb \rightarrow serves\\&\quad\langle Verb\ \textsc{head agreement number}\rangle = sg\\&\quad\langle Verb\ \textsc{head subcat first cat}\rangle = NP\\&\quad\langle Verb\ \textsc{head subcat second}\rangle = end\end{aligned}$$

This entry uses the FIRST feature to state that the first post-verbal argument must be an *NP*; the value of the SECOND feature indicates that this verb expects only one argument. A verb like *leave Boston in the morning*, with two arguments, would have the following kind of entry.

$$\begin{aligned}&Verb \rightarrow leaves\\&\quad\langle Verb\ \textsc{head agreement number}\rangle = sg\\&\quad\langle Verb\ \textsc{head subcat first cat}\rangle = NP\\&\quad\langle Verb\ \textsc{head subcat second cat}\rangle = PP\\&\quad\langle Verb\ \textsc{head subcat third}\rangle = end\end{aligned}$$

This scheme is, of course, a rather baroque way of encoding a list; it is also possible to use the idea of **types** defined in Sec. 15.6 to define a list type more cleanly.

The individual verb phrase rules must now check for the presence of exactly the elements specified by their verb, as in the following transitive rule.

(15.14) $VP \rightarrow Verb\ NP$

$$\langle VP \text{ HEAD}\rangle = \langle Verb \text{ HEAD}\rangle$$
$$\langle VP \text{ HEAD SUBCAT FIRST CAT}\rangle = \langle NP \text{ CAT}\rangle$$
$$\langle VP \text{ HEAD SUBCAT SECOND}\rangle = end$$

The second constraint in this rule's constraints states that the category of the first element of the verb's SUBCAT list must match the category of the constituent immediately following the verb. The third constraint goes on to state that this verb phrase rule expects only a single argument.

Subcategorization frames

Our previous examples have shown rather simple subcategorization structures for verbs. In fact, verbs can subcategorize for quite complex **subcategorization frames**, (e.g., *NP PP*, *NP NP*, or *NP S*) and these frames can be composed of many different phrasal types. In order to come up with a list of possible subcategorization frames for English verbs, we first need to have a list of possible phrase types that can make up these frames. Fig. 15.3.3 shows one short list of possible phrase types for making up subcategorization frames for verbs; this list is modified from one used to create verb subcategorization frames in the FrameNet project (Johnson, 1999; Baker et al., 1998), and includes phrase types for special subjects of verbs like *there* and *it*, as well as for objects and complements.

To use the phrase types in Fig. 15.3.3 in a unification grammar, each phrase type could be described using features. For example the form **VPto**, which is subcategorized for by *want* might be expressed as:

$$Verb \rightarrow want$$
$$\langle Verb \text{ HEAD SUBCAT FIRST CAT}\rangle = VP$$
$$\langle Verb \text{ HEAD SUBCAT FIRST FORM}\rangle = infinitive$$

Each of the 50 to 100 possible verb subcategorization frames in English would be described as a list drawn from these phrase types. For example, here is an example of the two-complement *want*. We can use this example to demonstrate two different notational possibilities. First, lists can be represented via an angle brackets notation ⟨ and ⟩. Second, instead of using a rewrite-rule annotated with path equations, we can represent the lexical entry as a single feature structure:

$$\begin{bmatrix} \text{ORTH} & want \\ \text{CAT} & Verb \\ \text{HEAD} & \begin{bmatrix} \text{SUBCAT} & \left\langle \begin{bmatrix}\text{CAT } NP\end{bmatrix}, \begin{bmatrix} \text{CAT } VP \\ \text{HEAD} \begin{bmatrix}\text{VFORM } infinitival\end{bmatrix}\end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Combining even a limited set of phrase types results in a very large set of possible subcategorization frames. Furthermore, each verb allows many different subcategorization frames. Fig. 15.4 provides a set of subcategorization patterns for the verb *ask*, with examples from the BNC:

A number of comprehensive subcategorization-frame tagsets exist, such as the COMLEX set (Macleod et al., 1998), which includes subcategorization frames for verbs, adjectives, and nouns, and the ACQUILEX tagset of verb subcategorization

Noun Phrase Types		
There	nonreferential there	**There** *is still much to learn*
It	nonreferential it	**It** *was evident that my ideas*
NP	noun phrase	*As he was relating* **his story**
Preposition Phrase Types		
PP	preposition phrase	*couch their message* **in terms**
PPing	gerundive PP	*censured him* **for not having intervened**
PPpart	particle	*turn it* **off**
Verb Phrase Types		
VPbrst	bare stem VP	*she could* **discuss it**
VPto	to-marked infin. VP	*Why do you want* **to know***?*
VPwh	wh-VP	*it is worth considering* **how to write**
VPing	gerundive VP	*I would consider* **using it**
Complement Clause types		
Sfin	finite clause	*maintain* **that the situation was unsatisfactory**
Swh	wh-clause	*it tells us* **where we are**
Sif	whether/if clause	*ask* **whether Aristophanes is depicting a**
Sing	gerundive clause	*see* **some attention being given**
Sto	to-marked clause	*know* **themselves to be relatively unhealthy**
Sforto	for-to clause	*She was waiting* **for him to make some reply**
Sbrst	bare stem clause	*commanded* **that his sermons be published**
Other Types		
AjP	adjective phrase	*thought it* **possible**
Quo	quotes	*asked* **"What was it like?"**

Figure 15.3 A small set of potential phrase types which can be combined to create a set of potential subcategorization frames for verbs. Modified from the FrameNet tagset (Johnson, 1999; Baker et al., 1998). The sample sentence fragments are from the British National Corpus.

Subcat	Example
Quo	asked [$_{Quo}$ "What was it like?"]
NP	asking [$_{NP}$ a question]
Swh	asked [$_{Swh}$ what trades you're interested in]
Sto	ask [$_{Sto}$ him to tell you]
PP	that means asking [$_{PP}$ at home]
Vto	asked [$_{Vto}$ to see a girl called Evelyn]
NP Sif	asked [$_{NP}$ him] [$_{Sif}$ whether he could make]
NP NP	asked [$_{NP}$ myself] [$_{NP}$ a question]
NP Swh	asked [$_{NP}$ him] [$_{Swh}$ why he took time off]

Figure 15.4 A set of sample subcategorization patterns for the verb *ask* with examples from the BNC.

frames (Sanfilippo, 1993). Many subcategorization-frame tagsets add other information about the complements, such as specifying the identity of the implicit subject in a lower verb phrase that has no overt subject; this is called **control** information. For example *Temmy promised Ruth to go* (at least in some dialects) implies that Temmy will do the going, while *Temmy persuaded Ruth to go* implies that Ruth will do the go-

Control

ing. Some of the multiple possible subcategorization frames for a verb can be partially predicted by the semantics of the verb; for example many verbs of transfer (like *give*, *send*, *carry*) predictably take the two subcategorization frames *NP NP* and *NP PP*:

NP NP	sent FAA Administrator James Busey a letter
NP PP	sent a letter to the chairman of the Armed Services Committee

Alternations

These relationships between subcategorization frames across classes of verbs are called argument-structure **alternations**, and will be discussed in Ch. 19 when we discuss the semantics of verbal argument structure. Ch. 14 will introduce probabilities for modeling the fact that verbs generally have preferences even among the different subcategorization frames they allow.

Subcategorization in Other Parts of Speech

Valence

Although the notion of subcategorization, or **valence** as it is often called, was originally conceived for verbs, more recent work has focused on the fact that many other kinds of words exhibit forms of valence-like behavior. Consider the following contrasting uses of the prepositions *while* and *during*.

(15.15) Keep your seatbelt fastened while *we are taking off*.
(15.16) *Keep your seatbelt fastened while *takeoff*.
(15.17) Keep your seatbelt fastened during *takeoff*.
(15.18) *Keep your seatbelt fastened during *we are taking off*.

Despite the apparent similarities between these words, they make quite different demands on their arguments. Representing these differences is left as Exercise 5 for the reader.

Many adjectives and nouns also have subcategorization frames. Here are some examples using the adjectives *apparent*, *aware*, and *unimportant* and the nouns *assumption* and *question*:

It was **apparent** [$_{Sfin}$ that the kitchen was the only room...]
It was **apparent** [$_{PP}$ from the way she rested her hand over his]
aware [$_{Sfin}$ he may have caused offense]
it is **unimportant** [$_{Swheth}$ whether only a little bit is accepted]
the **assumption** [$_{Sfin}$ that wasteful methods have been employed]
the **question** [$_{Swheth}$ whether the authorities might have decided]

See Macleod et al. (1998) and Johnson (1999) for descriptions of subcategorization frames for nouns and adjectives.

Verbs express subcategorization constraints on their subjects as well as their complements. For example, we need to represent the lexical fact that the verb *seem* can take an **Sfin** as its subject (*That she was affected seems obvious*), while the verb *paint* cannot. The SUBJECT feature can be used to express these constraints.

15.3.4 Long-Distance Dependencies

The model of subcategorization we have developed so far has two components. Each head word has a SUBCAT feature which contains a list of the complements it expects.

Then phrasal rules like the *VP* rule in (15.15) match up each expected complement in the SUBCAT list with an actual constituent. This mechanism works fine when the complements of a verb are in fact to be found in the verb phrase.

Long-distance dependencies

Sometimes, however, a constituent subcategorized for by the verb is not locally instantiated, but stands in a **long-distance** relationship with its predicate. Here are some examples of such **long-distance dependencies**:

What cities does Continental service?
What flights do you have from Boston to Baltimore?
What time does that flight leave Atlanta?

In the first example, the constituent *what cities* is subcategorized for by the verb *service*, but because the sentence is an example of a **wh-non-subject-question**, the object is located at the front of the sentence. Recall from Ch. 12 that a (simple) phrase-structure rule for a **wh-non-subject-question** is something like the following:

$$S \rightarrow \textit{Wh-NP Aux NP VP}$$

Now that we have features, we can augment this phrase-structure rule to require the *Aux* and the *NP* to agree (since the *NP* is the subject). But we also need some way to augment the rule to tell it that the *Wh-NP* should fill some subcategorization slot in the *VP*. The representation of such long-distance dependencies is a quite difficult problem, because the verb whose subcategorization requirement is being filled can be quite distant from the filler. In the following (made-up) sentence, for example, the *wh*-phrase *which flight* must fill the subcategorization requirements of the verb *book*, despite the fact that there are two other verbs (*want* and *have*) in between:

Which flight do you want me to have the travel agent book?

Gap list

Filler

Many solutions to representing long-distance dependencies in unification grammars involve keeping a list, often called a **gap list**, implemented as a feature GAP, which is passed up from phrase to phrase in the parse tree. The **filler** (for example *which flight* above) is put on the gap list, and must eventually be unified with the subcategorization frame of some verb. See Sag and Wasow (1999) for an explanation of such a strategy, together with a discussion of the many other complications that must be modeled in long-distance dependencies.

15.4 Implementing Unification

As discussed, the unification operator takes two feature structures as input and returns a single merged feature structure if successful, or a failure signal if the two inputs are not compatible. The input feature structures are represented as directed acyclic graphs (DAGs), where features are depicted as labels on directed edges, and feature values are either atomic symbols or DAGs. As we will see, the implementation of the operator is a relatively straightforward recursive graph matching algorithm, suitably tailored to accommodate the various requirements of unification. Roughly speaking, the algorithm loops through the features in one input and attempts to find a corresponding feature in the other. If all of the respective feature values match, then the unification is successful. If there is a mismatch then the unification fails. Not surprisingly, the recursion is motivated by the need to correctly match those features that have feature structures as their values.

A notable aspect of this algorithm is that rather than constructing a new feature structure with the unified information from the two arguments, it destructively alters the arguments so that in the end they point to exactly the same information. Thus, the result of a successful call to the unification operator consists of suitably altered versions of the arguments. As is discussed in the next section, the destructive nature of this algorithm necessitates certain minor extensions to the simple graph version of feature structures as DAGs we have been assuming.

15.4.1 Unification Data Structures

To facilitate the destructive merger aspect of the algorithm, we add a small complication to the DAGs used to represent the input feature structures; feature structures are represented using DAGs with additional edges, or fields. Specifically, each feature structure consists of two fields: a content field and a pointer field. The content field may be null or contain an ordinary feature structure. Similarly, the pointer field may be null or contain a pointer to another feature structure. If the pointer field of the DAG is null, then the content field of the DAG contains the actual feature structure to be processed. If, on the other hand, the pointer field is non-null, then the destination of the pointer represents the actual feature structure to be processed. The merger aspects of unification will be achieved by altering the pointer field of DAGs during processing.

To see how this works, let's consider the extended DAG representation for the following familiar feature structure.

(15.19) $$\begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

Figure 15.5 shows this extended representation in its graphical form. Note that the extended representation contains content and pointer links both for the top-level layer of features, as well as for each of the embedded feature structures all the way down to the atomic values.

Before presenting the details of the unification algorithm, let's illustrate the use of this extended DAG representation with the following simple example. The original extended representation of the arguments to this unification are shown in Figure 15.6.

(15.20) $$\begin{bmatrix} \text{NUMBER} & \textit{sg} \end{bmatrix} \sqcup \begin{bmatrix} \text{PERSON} & \textit{3rd} \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{3rd} \end{bmatrix}$$

At a basic level, this unification results in the creation of a new structure containing the union of the information from the two original arguments. With the extended notation, we can see how the unification is accomplished by making some additions to the original arguments and changing some of the pointers from one structure to the other so that in the end they contain the same content. In this example, this is accomplished by first adding a PERSON feature to the first argument, and assigning it a value by filling its POINTER field with a pointer to the appropriate location in the second argument, as shown in Figure 15.7.

The process is, however, not yet complete. While it is clear from Figure 15.7 that the first argument now contains all the correct information, the second one does not; it lacks a NUMBER feature. We could, of course, add a NUMBER feature to this argument with a pointer to the appropriate place in the first one. This change would result in the two arguments having all the correct information from this unification. Unfortunately, this solution is inadequate since it does not meet our requirement that the two arguments be truly unified. Since the two arguments are not completely unified at the top level, future unifications involving one of the arguments would not show up in the other. The solution to this problem is to simply set the POINTER field of the second argument to point at the first one. When this is done any future change to either argument will be immediately reflected in both. The result of this final change is shown in Figure 15.8.

15.4.2 The Unification Algorithm

The unification algorithm that we have been leading up to is shown in Figure 15.9. This algorithm accepts two feature structures represented using the extended DAG representation and returns as its value a modified version of one of the arguments, or a failure signal in the event that the feature structures are incompatible.

The first step in this algorithm is to acquire the true contents of both of the arguments. Recall that if the pointer field of an extended feature structure is non-null, then the real content of that structure is found by following the pointer found in pointer field. The variables *f1* and *f2* are the result of this pointer following process, often referred to as **dereferencing**.

Dereferencing

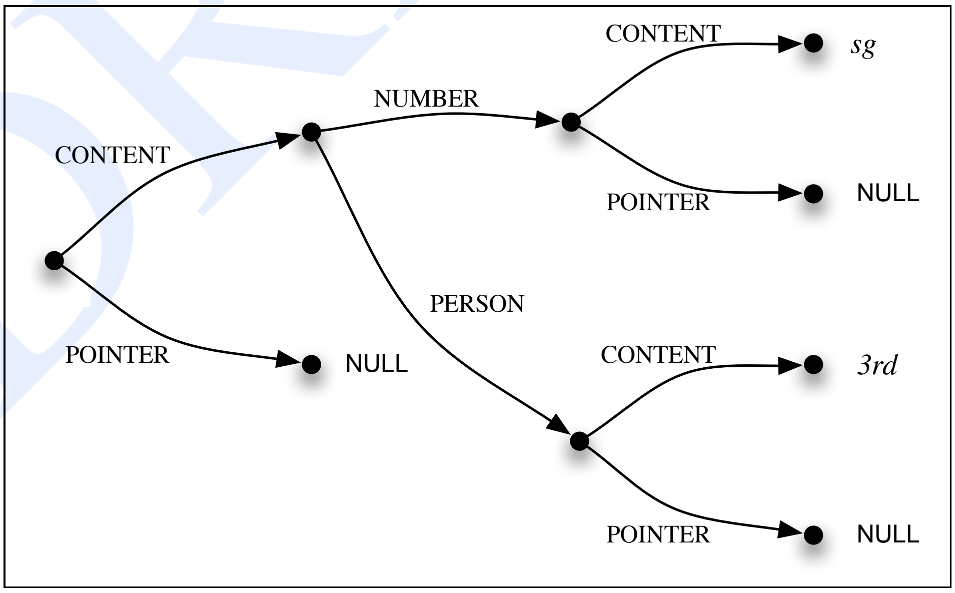

Figure 15.5 An extended DAG notation for Example 15.19.

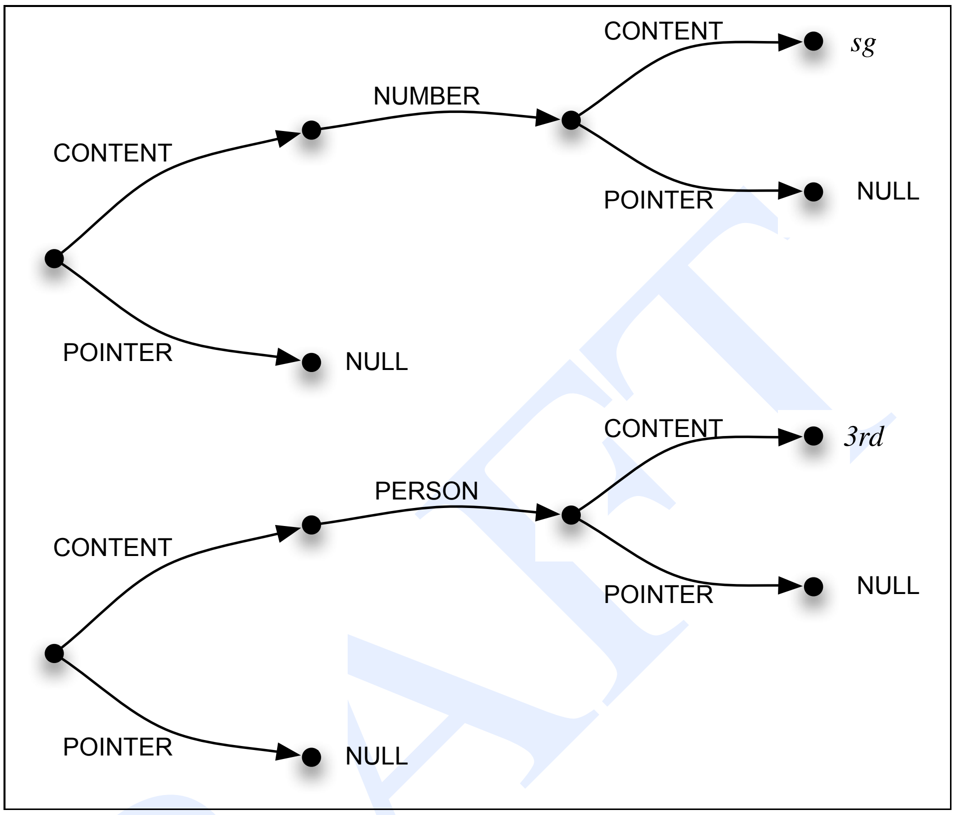

Figure 15.6 The original arguments to Example 15.20.

As with all recursive algorithms, the next step is to test for the various base cases of the recursion before proceeding on to a recursive call involving some part of the original arguments. In this case, there are three possible base cases:

- The arguments are identical
- One or both of the arguments has a null value
- The arguments are non-null and non-identical

If the structures are identical, then the pointer of the first is set to the second and the second is returned. It is important to understand why this pointer change is done in this case. After all, since the arguments are identical, returning either one would appear to suffice. This might be true for a single unification but recall that we want the two arguments to the unification operator to be truly unified. The pointer change is necessary since we want the arguments to be truly identical, so that any subsequent unification that adds information to one will add it to both.

In the case where either of the arguments is null, the pointer field for the null argument is changed to point to the other argument, which is then returned. The result is that both structures now point at the same value.

If neither of the preceding tests is true then there are two possibilities: they are non-identical atomic values, or they are non-identical complex structures. The former

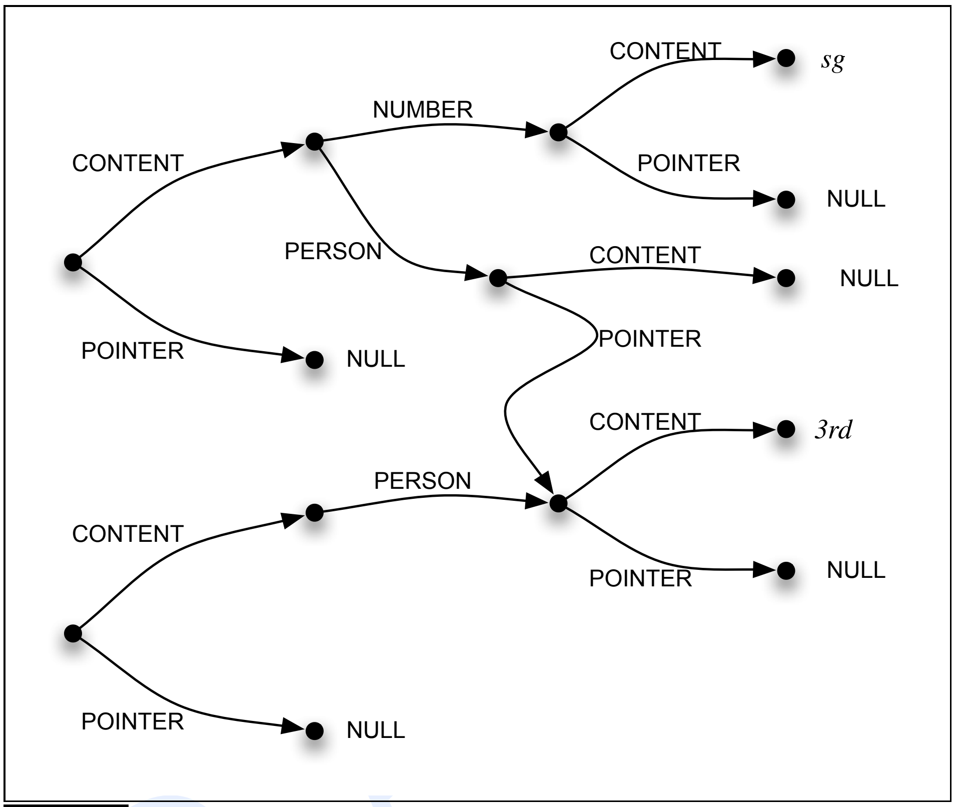

Figure 15.7 The arguments after assigning the first argument's new PERSON feature to the appropriate value in the second argument.

case signals an incompatibility in the arguments that leads the algorithm to return a failure signal. In the latter case, a recursive call is needed to ensure that the component parts of these complex structures are compatible. In this implementation, the key to the recursion is a loop over all the features of the *second* argument, *f2*. This loop attempts to unify the value of each feature in *f2* with the corresponding feature in *f1*. In this loop, if a feature is encountered in *f2* that is missing from *f1*, a feature is added to *f1* and given the value NULL. Processing then continues as if the feature had been there to begin with. If *every* one of these unifications succeeds, then the pointer field of *f2* is set to *f1* completing the unification of the structures and *f1* is returned as the value of the unification.

An Example

To illustrate this algorithm, let's walk through the following example.

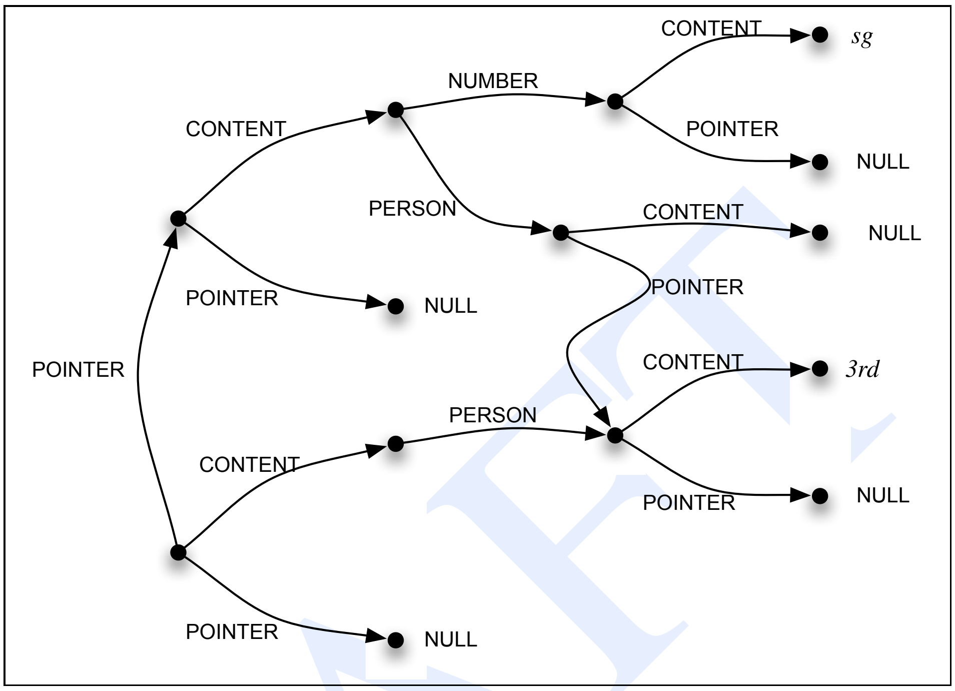

Figure 15.8 The final result of unifying F1 and F2.

(15.21) $$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix}\text{NUMBER} & \textit{sg}\end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix}\end{bmatrix}$$
$$\sqcup \begin{bmatrix}\text{SUBJECT} & \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 15.10 shows the extended representations for the arguments to this unification. These original arguments are neither identical, nor null, nor atomic, so the main loop is entered. Looping over the features of *f2*, the algorithm is led to a recursive attempt to unify the values of the corresponding SUBJECT features of *f1* and *f2*.

$$\begin{bmatrix}\text{AGREEMENT} & \boxed{1}\end{bmatrix} \sqcup \begin{bmatrix}\text{AGREEMENT} & \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}\end{bmatrix}$$

These arguments are also non-identical, non-null, and non-atomic so the loop is entered again leading to a recursive check of the values of the AGREEMENT features.

$$\begin{bmatrix}\text{NUMBER} & \textit{sg}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}$$

In looping over the features of the second argument, the fact that the first argument lacks a PERSON feature is discovered. A PERSON feature initialized with a NULL value is, therefore, added to the first argument. This, in effect, changes the previous unification to the following.

$$\begin{bmatrix}\text{NUMBER} & \textit{sg} \\ \text{PERSON} & \textit{null}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & \textit{3rd}\end{bmatrix}$$

```
function UNIFY(f1-orig,f2-orig) returns f-structure or failure

 f1 ← Dereferenced contents of f1-orig
 f2 ← Dereferenced contents of f2-orig

 if f1 and f2 are identical then
    f1.pointer ← f2
    return f2
 else if f1 is null then
    f1.pointer ← f2
    return f2
 else if f2 is null then
    f2.pointer ← f1
    return f1
 else if both f1 and f2 are complex feature structures then
    f2.pointer ← f1
    for each f2-feature in f2 do
       f1-feature ← Find or create a corresponding feature in f1
       if UNIFY(f1-feature.value,f2-feature.value) returns failure then
          return failure
    return f1
 else return failure
```

Figure 15.9 The unification algorithm.

After creating this new PERSON feature, the next recursive call leads to the unification of the NULL value of the new feature in the first argument with the *3rd* value of the second argument. Since there are no further features to check in the *f2* argument at any level of recursion, each of the recursive calls to UNIFY returns. The result is shown in Figure 15.11.

15.5 Parsing with Unification Constraints

We now have all the pieces necessary to integrate feature structures and unification into a parser. Fortunately, the order-independent nature of unification allows us to largely ignore the actual search strategy used in the parser. Once we have associated unification constraints with the context-free rules of the grammar, and feature structures with the states of the search, any of the standard search algorithms described in Ch. 13 can be used.

Of course, this leaves a fairly large range of possible implementation strategies. We could, for example, simply parse as we did before using the context-free components of the rules, and then build the feature structures for the resulting trees after the fact, filtering out those parses that contain unification failures. Although such an approach would result in only well-formed structures in the end, it fails to use the power of unification to reduce the size of the parser's search space during parsing.

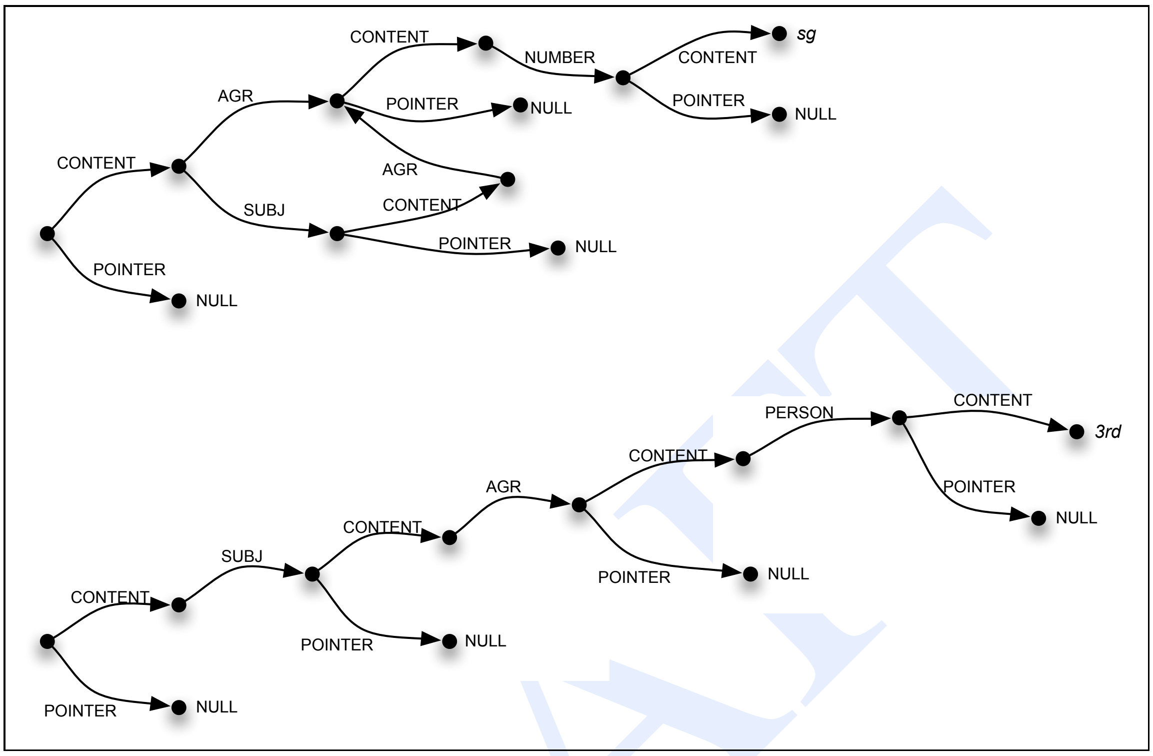

Figure 15.10 The initial arguments *f1* and *f2* to Example 15.21.

The next section describes an approach that makes better use of the power of unification by integrating unification constraints directly into the Earley parsing process, allowing ill-formed structures to be eliminated as soon as they are proposed. As we will see, this approach requires only minimal changes to the basic Earley algorithm. We then move on to briefly consider an approach to unification-based parsing that moves even further away from standard context-free methods.

15.5.1 Integrating Unification into an Earley Parser

We have two goals in integrating feature structures and unification into the Earley algorithm: to use feature structures to provide a richer representation for the constituents of the parse, and to block entry into the chart of ill-formed constituents that violate unification constraints. As we will see, these goals can be accomplished by fairly minimal changes to the original Earley scheme given on page 448.

The first change involves the various representations used in the original code. Recall that the Earley algorithm operates by using a set of unadorned context-free grammar rules to fill in a data-structure called a chart with a set of states. At the end of the parse, the states that make up this chart represent all possible parses of the input. Therefore, we begin our changes by altering the representations of both the context-free grammar rules, and the states in the chart.

The rules are altered so that in addition to their current components, they also in-

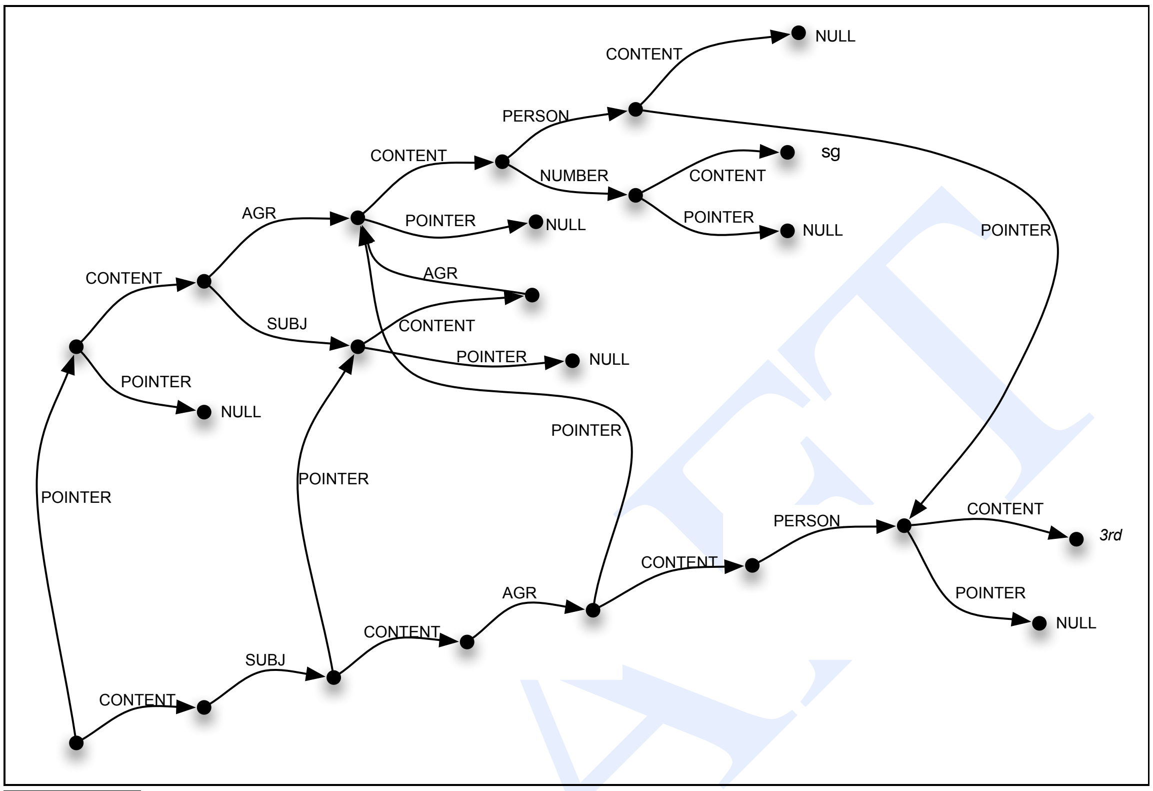

Figure 15.11 The final structures of *f1* and *f2* at the end.

clude a feature structure derived from their unification constraints. More specifically, we will use the constraints listed with a rule to build a feature structure, represented as a DAG, for use with that rule during parsing.

Consider the following context-free rule with unification constraints.

$$S \rightarrow NP\ VP$$
$$\langle NP\ \text{HEAD AGREEMENT}\rangle = \langle VP\ \text{HEAD AGREEMENT}\rangle$$
$$\langle S\ \text{HEAD}\rangle = \langle VP\ \text{HEAD}\rangle$$

Converting these constraints into a feature structure results in the following structure:

$$\begin{bmatrix} \text{S} & \begin{bmatrix}\text{HEAD} & \boxed{1}\end{bmatrix} \\ \text{NP} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{AGREEMENT} & \boxed{2}\end{bmatrix}\end{bmatrix} \\ \text{VP} & \begin{bmatrix}\text{HEAD} & \boxed{1}\begin{bmatrix}\text{AGREEMENT} & \boxed{2}\end{bmatrix}\end{bmatrix} \end{bmatrix}$$

In this derivation, we combined the various constraints into a single structure by first creating top-level features for each of the parts of the context-free rule, S, NP, and VP in this case. We then add further components to this structure by following the path equations in the constraints. Note that this is a purely notational conversion; the DAGs and the constraint equations contain the same information. However, tying the constraints together in a single feature structure puts it in a form that can be passed directly to our unification algorithm.

The second change involves the states used to represent partial parses in the Earley chart. The original states contain fields for the context-free rule being used, the position of the dot representing how much of the rule has been completed, the positions of the beginning and end of the state, and a list of other states that represent the completed sub-parts of the state. To this set of fields, we simply add an additional field to contain the DAG representing the feature structure corresponding to the state. Note that when a rule is first used by PREDICTOR to create a state, the DAG associated with the state will simply consist of the DAG retrieved from the rule. For example, when PREDICTOR uses the above *S* rule to enter a state into the chart, the DAG given above will be its initial DAG. We'll denote states like this as follows, where *Dag* denotes the feature structure given above.

$$S \rightarrow \bullet\, NP\ VP,\ [0,0],[],Dag$$

Given these representational additions, we can move on to altering the algorithm itself. The most important change concerns the actions that take place when a new state is created by the extension of an existing state, which takes place in the COMPLETER routine. Recall that COMPLETER is called when a completed constituent has been added to the chart. Its task is to attempt to find, and extend, existing states in the chart that are looking for constituents that are compatible with the newly completed constituent. COMPLETER is, therefore, a function that creates new states by *combining* the information from two other states, and as such is a likely place to apply the unification operation.

To be more specific, COMPLETER adds a new state into the chart by finding an existing state whose • can be advanced by the newly completed state. A • can be advanced when the category of the constituent immediately following it matches the category of the newly completed constituent. To accommodate the use of feature structures, we can alter this scheme by unifying the feature structure associated with the newly completed state with the appropriate part of the feature structure being advanced. If this unification succeeds, then the DAG of the new state receives the unified structure and is entered into the chart. If it fails, then no new state is entered into the chart. The appropriate alterations to COMPLETER are shown in Figure 15.12.

Consider this process in the context of parsing the phrase *That flight*, where the *That* has already been seen, as is captured by the following state.

$$NP \rightarrow Det\bullet Nominal[0,1],[S_{Det}],Dag_1$$

$$Dag_1 \begin{bmatrix} \text{NP} & \begin{bmatrix}\text{HEAD} & \boxed{1}\end{bmatrix} \\ \text{DET} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{AGREEMENT} \ \boxed{2} \begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{NOMINAL} & \begin{bmatrix}\text{HEAD} & \boxed{1}\begin{bmatrix}\text{AGREEMENT} & \boxed{2}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Now consider the later situation where the parser has processed *flight* and has subsequently produced the following state.

$$Nominal \rightarrow Noun\bullet, [1,2], [S_{Noun}], Dag_2$$

$$Dag_2 \begin{bmatrix} \text{NOMINAL} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{NOUN} & \begin{bmatrix} \text{HEAD} & \boxed{1} \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

To advance the *NP* rule, the parser unifies the feature structure found under the NOMINAL feature of Dag_2, with the feature structure found under the NOMINAL feature of the *NP*'s Dag_1. As in the original algorithm, a new state is created to represent the fact that an existing state has been advanced. This new state's DAG is given the DAG that resulted from this unification.

The final change to the original algorithm concerns the check for states already contained in the chart. In the original algorithm, the ENQUEUE function refused to enter into the chart any state that was *identical* to one already present in the chart. "Identical" meant the same rule, with the same start and finish positions, and the same position of the $\bullet$. It is this check that allows the algorithm to, among other things, avoid the infinite recursion problems associated with left-recursive rules.

The problem, of course, is that our states are now more complex since they have complex feature structures associated with them. States that appeared identical under the original criteria might in fact now be different since their associated DAGs may differ. One solution to this problem is to extend the identity check to include the DAGs associated with the states, but it turns out that we can improve on this solution.

The motivation for the improvement lies in the motivation for the identity check. Its purpose is to prevent the wasteful addition of a state into the chart whose effect on the parse would be accomplished by an already existing state. Put another way, we want to prevent the entry into the chart of any state that would duplicate the work that will eventually be done by other states. Of course, this will clearly be the case with identical states, but it turns out it is also the case for states in the chart that are *more general* than new states being considered.

Consider the situation where the chart contains the following state, where the *Dag* places no constraints on the *Det*.

$$NP \rightarrow \bullet Det\ NP, [i,i], [], Dag$$

Such a state simply says that it is expecting a *Det* at position i, and that any *Det* will do.

Now consider the situation where the parser wants to insert a new state into the chart that is identical to this one, with the exception that its DAG restricts the *Det* to be singular. In this case, although the states in question are not identical, the addition of the new state to the chart would accomplish nothing and should therefore be prevented.

To see this let's consider all the cases. If the new state is added, then a subsequent singular *Det* will match both rules and advance both. Due to the unification of features, both will have DAGs indicating that their *Det*s are singular, with the net result being

```
function EARLEY-PARSE(words, grammar) returns chart

  ADDTOCHART(($\gamma \rightarrow \bullet S$, [0,0], $dag_\gamma$),chart[0])
  for i ← from 0 to LENGTH(words) do
   for each state in chart[i] do
     if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
     elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
        SCANNER(state)
     else
        COMPLETER(state)
   end
  end
  return(chart)

 procedure PREDICTOR(($A \rightarrow \alpha \bullet B\,\beta$, [i, j], $dag_A$))
    for each ($B \rightarrow \gamma$) in GRAMMAR-RULES-FOR(B,grammar) do
       ADDTOCHART(($B \rightarrow \bullet\gamma$, [j, j], $dag_B$),chart[j])
    end

 procedure SCANNER(($A \rightarrow \alpha \bullet B\,\beta$, [i, j], $dag_A$))
    if B ∈ PARTS-OF-SPEECH(word[j]) then
       ADDTOCHART(($B \rightarrow word[j]\bullet$, [j, j+1], $dag_B$),chart[j+1])

 procedure COMPLETER(($B \rightarrow \gamma \bullet$, [j,k], $dag_B$))
    for each ($A \rightarrow \alpha \bullet B\,\beta$, [i, j], $dag_A$) in chart[j] do
      if new-dag ← UNIFY-STATES($dag_B$,$dag_A$,B) ≠ Fails!
       ADDTOCHART(($A \rightarrow \alpha\, B \bullet \beta$, [i,k],new-dag),chart[k])
    end

 procedure UNIFY-STATES(dag1,dag2,cat)
   dag1-cp ← COPYDAG(dag1)
   dag2-cp ← COPYDAG(dag2)
   UNIFY(FOLLOW-PATH(cat,dag1-cp),FOLLOW-PATH(cat,dag2-cp))

 procedure ADDTOCHART(state, chart-entry)
    if state is not subsumed by a state in chart-entry then
       PUSH-ON-END(state, chart-entry)
    end
```

Figure 15.12 Modifications to the Earley algorithm to include unification.

duplicate states in the chart. If on the other hand, a plural *Det* is encountered, the new state will reject it and not advance, while the old rule will advance, entering a single new state into the chart. On the other hand, if the new state is not placed in the chart, a subsequent plural or singular *Det* will match the more general state and advance it, leading to the addition of one new state into the chart. Note that this leaves us in exactly the same situation as if the new state had been entered into the chart, with the exception that the duplication is avoided. In sum, nothing worthwhile is accomplished by entering into the chart a state that is more specific than a state already in the chart.

Fortunately, the notion of **subsumption** introduced earlier gives us a formal way to talk about the generalization and specialization relations among feature structures. This suggests that the proper way to alter ENQUEUE is to check whether a newly created state is *subsumed* by any existing states in the chart. If it is, then it will not be allowed into the chart. More specifically, a new state that is identical in terms of its rule, start and finish positions, subparts, and • position, to an existing state, will be not be entered into the chart if its DAG is subsumed by the DAG of an existing state (ie. if $Dag_{old} \sqsubseteq Dag_{new}$). The necessary change to the original Earley ENQUEUE procedure is shown in Figure 15.12.

The Need for Copying

The calls to COPYDAG within the UNIFY-STATE procedure require some elaboration. Recall that one of the strengths of the Earley algorithm (and of the dynamic programming approach in general) is that once states have been entered into the chart they may be used again and again as part of different derivations, including ones that in the end do not lead to successful parses. This ability is the motivation for the fact that states already in the chart are not updated to reflect the progress of their •, but instead are copied and then updated, leaving the original states intact so that they can be used again in further derivations.

The call to COPYDAG in UNIFY-STATE is required to preserve this behavior because of the destructive nature of our unification algorithm. If we simply unified the DAGs associated with the existing states, those states would be altered by the unification, and hence would not be available in the same form for subsequent uses by the COMPLETER function. Note that this has negative consequences regardless of whether the unification succeeds or fails, since in either case the original states are altered.

Let's consider what would happen if the call to COPYDAG was absent in the following example where an early unification attempt fails.

(15.22) Show me morning flights.

Let's assume that our parser has the following entry for the ditransitive version of the verb *show*, as well as the following transitive and ditransitive verb phrase rules.

$$\begin{aligned}
&\mathit{Verb} \rightarrow \mathit{show}\\
&\quad \langle \mathit{Verb}\ \text{HEAD SUBCAT FIRST CAT}\rangle = \mathit{NP}\\
&\quad \langle \mathit{Verb}\ \text{HEAD SUBCAT SECOND CAT}\rangle = \mathit{NP}\\
&\quad \langle \mathit{Verb}\ \text{HEAD SUBCAT THIRD}\rangle = \mathit{END}
\end{aligned}$$

$$\begin{aligned}
&\mathit{VP} \rightarrow \mathit{Verb}\ \mathit{NP}\\
&\quad \langle \mathit{VP}\ \text{HEAD}\rangle = \langle \mathit{Verb}\ \text{HEAD}\rangle\\
&\quad \langle \mathit{VP}\ \text{HEAD SUBCAT FIRST CAT}\rangle = \langle \mathit{NP}\ \text{CAT}\rangle\\
&\quad \langle \mathit{VP}\ \text{HEAD SUBCAT SECOND}\rangle = \mathit{END}
\end{aligned}$$

$$\mathit{VP} \rightarrow \mathit{Verb}\ \mathit{NP}\ \mathit{NP}$$

$\langle VP$ HEAD$\rangle = \langle Verb$ HEAD$\rangle$
$\langle VP$ HEAD SUBCAT FIRST CAT$\rangle = \langle NP_1$ CAT $\rangle$
$\langle VP$ HEAD SUBCAT SECOND CAT$\rangle = \langle NP_2$ CAT $\rangle$
$\langle VP$ HEAD SUBCAT THIRD$\rangle = END$

When the word *me* is read, the state representing the transitive verb phrase will be completed since its dot has moved to the end. COMPLETER will, therefore, call UNIFY-STATES before attempting to enter this complete state into the chart. This will fail since the SUBCAT structures of these two rules can not be unified. This is, of course, exactly what we want since this version of *show* is ditransitive. Unfortunately, because of the destructive nature of our unification algorithm we have already altered the DAG attached to the state representing *show*, as well as the one attached to the *VP* thereby ruining them for use with the correct verb phrase rule later on. Thus, to make sure that states can be used again and again with multiple derivations, copies are made of the dags associated with states before attempting any unifications involving them.

All of this copying can be quite expensive. As a result, a number of alternative techniques have been developed that attempt to minimize this cost (Pereira, 1985; Karttunen and Kay, 1985; Tomabechi, 1991; Kogure, 1990). Kiefer et al. (1999b) and Penn and Munteanu (2003) describe a set of related techniques used to speed up a large unification-based parsing system.

15.5.2 Unification-Based Parsing

A more radical approach to using unification in parsing can be motivated by looking at an alternative way of denoting our augmented grammar rules. Consider the following *S* rule that we have been using throughout this chapter.

$S \rightarrow NP\ VP$
$\langle NP$ HEAD AGREEMENT$\rangle = \langle VP$ HEAD AGREEMENT$\rangle$
$\langle S$ HEAD$\rangle = \langle VP$ HEAD$\rangle$

An interesting way to alter the context-free part of this rule is to change the way its grammatical categories are specified. In particular, we can place the categorical information about the parts of the rule inside the feature structure, rather than inside the context-free part of the rule. A typical instantiation of this approach would give us the following rule (Shieber, 1986).

$X_0 \rightarrow X_1\ X_2$
$\langle X_0$ CAT$\rangle = S$
$\langle X_1$ CAT$\rangle = NP$
$\langle X_2$ CAT$\rangle = VP$
$\langle X_1$ HEAD AGREEMENT$\rangle = \langle X_2$ HEAD AGREEMENT$\rangle$
$\langle X_0$ HEAD$\rangle = \langle X_2$ HEAD$\rangle$

Focusing solely on the context-free component of the rule, this rule now simply states that the X_0 constituent consists of two components, and that the X_1 constituent is immediately to the left of the X_2 constituent. The information about the actual categories of these components is placed inside the rule's feature structure; in this case, indicating that X_0 is an S, X_1 is an NP, and X_2 is a VP. Altering the Earley algorithm to deal with this notational change is trivial. Instead of seeking the categories of constituents in the context-free components of the rule, it simply needs to look at the CAT feature in the DAG associated with a rule.

Of course, since it is the case that these two rules contain precisely the same information, it isn't clear that there is any benefit to this change. To see the potential benefit of this change, consider the following rules.

$$\begin{aligned} X_0 &\rightarrow X_1\ X_2 \\ &\langle X_0\ \text{CAT}\rangle = \langle X_1\ \text{CAT}\rangle \\ &\langle X_2\ \text{CAT}\rangle = PP \end{aligned}$$

$$\begin{aligned} X_0 &\rightarrow X_1\ \textit{and}\ X_2 \\ &\langle X_1\ \text{CAT}\rangle = \langle X_2\ \text{CAT}\rangle \\ &\langle X_0\ \text{CAT}\rangle = \langle X_1\ \text{CAT}\rangle \end{aligned}$$

The first rule is an attempt to generalize over various rules that we have already seen, such as $NP \rightarrow NP\ PP$ and $VP \rightarrow VP\ PP$. It simply states that any category can be followed by a prepositional phrase, and that the resulting constituent has the same category as the original. Similarly, the second rule is an attempt to generalize over rules such as $S \rightarrow S\ and\ S$, $NP \rightarrow NP\ and\ NP$, and so on.[1] It states that any constituent can be conjoined with a constituent of the same category to yield a new category of the same kind. What these rules have in common is their use of phrase structure rules that contain constituents with constrained, but unspecified, categories, something that can not be accomplished with our old rule format.

Of course, since these rules rely on the use the CAT feature, their effect could be approximated in the old format by simply enumerating all the various instantiations of the rule. A more compelling case for the new approach is motivated by the existence of grammatical rules, or constructions, that contain constituents that are not easily characterized using any existing syntactic category.

Consider the following examples of the English HOW-MANY construction from the WSJ (Jurafsky, 1992).

(15.23) **How early** does it open?
(15.24) **How deep** is her Greenness?
(15.25) **How papery** are your profits?
(15.26) **How quickly** we forget.
(15.27) **How many of you** can name three famous sporting Blanchards?

As is illustrated in these examples, the HOW-MANY construction has two components: the lexical item *how*, and a lexical item or phrase that is rather hard to characterize syntactically. It is this second element that is of interest to us here. As these examples

[1] These rules should not be mistaken for correct, or complete, accounts of the phenomena in question.

show, it can be an adjective, adverb, or some kind of quantified phrase (although not all members of these categories yield grammatical results). Clearly, a better way to describe this second element is as a *scalar* concept, a constraint can be captured using feature structures, as in the following rule.

$$\begin{aligned} X_0 &\rightarrow X_1\ X_2 \\ &\langle X_1\ \text{ORTH}\rangle = \langle \textit{how}\rangle \\ &\langle X_2\ \text{SEM}\rangle = \langle\ \text{SCALAR}\rangle \end{aligned}$$

A complete account of rules like this involves semantics and will therefore have to wait for Ch. 17. The key point here is that by using feature structures a grammatical rule can place constraints on its constituents in a manner that does not make any use of the notion of a syntactic category.

Of course, dealing this kind of rule requires some changes to our parsing scheme. All of the parsing approaches we have considered thus far are driven by the syntactic category of the various constituents in the input. More specifically, they are based on simple atomic matches between the categories that have been predicted, and categories that have been found. Consider, for example, the operation of the COMPLETER function shown in Figure 15.12. This function searches the chart for states that can be advanced by a newly completed state. It accomplishes this by matching the category of the newly completed state against the category of the constituent following the • in the existing state. Clearly this approach will run into trouble when there are no such categories to consult.

The remedy for this problem with COMPLETER is to search the chart for states whose DAGs *unify* with the DAG of the newly completed state. This eliminates any requirement that states or rules have a category. The PREDICTOR can be changed in a similar fashion by having it add states to the chart states whose X_0 DAG component can unify with the constituent following the • of the predicting state. Exercise 6 asks you to make the necessary changes to the pseudo-code in Figure 15.12 to effect this style of parsing. Exercise 7 asks you to consider some of the implications of these alterations, particularly with respect to prediction.

15.6 Types and Inheritance

I am surprised that ancient and modern writers have not attributed greater importance to the laws of inheritance…

Alexis de Tocqueville, *Democracy in America*, 1840

The basic feature structures we have presented so far have two problems that have led to extensions to the formalism. The first problem is that there is no way to place a constraint on what can be the value of a feature. For example, we have implicitly assumed that the NUMBER attribute can take only *sg* and *pl* as values. But in our current system, there is nothing, for example, to stop NUMBER from have the value *3rd* or *feminine* as values:

$\begin{bmatrix}\text{NUMBER} & \text{FEMININE}\end{bmatrix}$

This problem has caused many unification-based grammatical theories to add various mechanisms to try to constrain the possible values of a feature. Formalisms like Functional Unification Grammar (FUG) (Kay, 1979, 1984, 1985) and Lexical Functional Grammar (LFG) (Bresnan, 1982), for example, focused on ways to keep intransitive verb like *sneeze* from unifying with a direct object (*Marin sneezed Toby*). This was addressed in FUG by adding a special atom **none** which is not allowed to unify with anything, and in LFG by adding **coherence** conditions which specified when a feature should not be filled. The Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985, 1988) added a class of **feature co-occurrence restrictions**, to prevent, for example, nouns from having some verbal properties.

None

The second problem with simple feature structures is that there is no way to capture generalizations across them. For example, the many types of English verb phrases described in the Subcategorization section on page 506 share many features, as do the many kinds of subcategorization frames for verbs. Syntacticians were looking for ways to express these generalities.

Types

A general solution to both of these problems is the use of **types**. Type systems for unification grammars have the following characteristics:

1. Each feature structure is labeled by a type.
2. Conversely, each type has **appropriateness conditions** expressing which features are appropriate for it and what types of values then can take.
3. The types are organized into a **type hierarchy**, in which more specific types inherit properties of more abstract ones.
4. The unification operation is modified to unify the types of feature structures in addition to unifying the attributes and values.

Appropriateness

Type hierarchy

Typed feature structures

Simple types

Complex types

In such **typed feature structure** systems, types are a new class of objects, just like attributes and values were for standard feature structures. Types come in two kinds: **simple types** (also called **atomic types**), and **complex types**. Let's begin with simple types. A simple type is an atomic symbol like **sg** or **pl** (we will use **boldface** for all types), and replaces the simple atomic values used in standard feature structures. All types are organized into a multiple-inheritance **type hierarchy** (a kind of **partial order** called a **lattice**). Fig. 15.13 shows the type hierarchy for the new type **agreement**, which will be the type of the kind of atomic object that can be the value of an AGREE feature.

Subtype

In the hierarchy in Fig. 15.13, **3rd** is a **subtype** of **agr**, and **3-sg** is a **subtype** of both **3rd** and **sg**. Types can be unified in the type hierarchy; the unification of any two types is the most-general type that is more specific than the two input types. Thus:

$$\mathbf{3rd} \sqcup \mathbf{sg} = \mathbf{3sg}$$
$$\mathbf{1st} \sqcup \mathbf{pl} = \mathbf{1pl}$$
$$\mathbf{1st} \sqcup \mathbf{agr} = \mathbf{1st}$$
$$\mathbf{3rd} \sqcup \mathbf{1st} = \textit{undefined}$$

The unification of two types which do not have a defined unifier is undefined, although it is also possible to explicitly represent this **fail type** using the symbol ⊥ (Aït-

Fail type

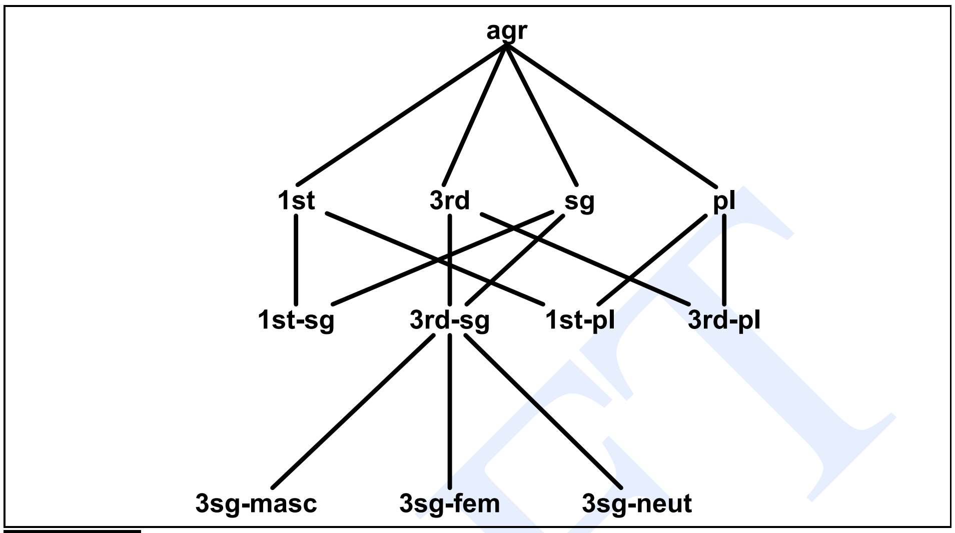

Figure 15.13 A simple type hierarchy for the subtypes of type **agr** which can be the value of the AGREE attribute. After Carpenter (1992).

Kaci, 1984).

The second kind of types are complex types, which specify:

- a set of features that are appropriate for that type
- restrictions on the values of those features (expressed in terms of types)
- equality constraints between the values

Consider a simplified representation of the complex type **verb**, which just represents agreement and verbal morphology information. A definition of **verb** would define the two appropriate features, AGREE and VFORM, and would also define the type of the values of the two features. Let's suppose that the AGREE feature takes values of type **agr** defined in Fig. 15.13 above, and the VFORM feature takes values of type **vform** where **vform** subsumes the seven subtypes **finite**, **infinitive**, **gerund**, **base**, **present-participle**, **past-participle**, and **passive-participle**. Thus **verb** would be defined as follows (where the convention is to indicate the type either at the top of the AVM or just to the lower left of the left bracket):

$$\begin{bmatrix} \textbf{verb} & \\ \text{AGREE} & \textbf{agr} \\ \text{VFORM} & \textbf{vform} \end{bmatrix}$$

By contrast, the type **noun** might be defined with the AGREE feature, but without the VFORM feature:

$$\begin{bmatrix} \textbf{noun} & \\ \text{AGREE} & \textbf{agr} \end{bmatrix}$$

The unification operation is augmented for typed feature structures just by requiring that the types of the two structures unify in addition to the values of the component features unifying.

$$\begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{1st} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix} \sqcup \begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{sg} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix} = \begin{bmatrix}\textbf{verb} & \\ \text{AGREE} & \textbf{1-sg} \\ \text{VFORM} & \textbf{gerund}\end{bmatrix}$$

Complex types are also part of the type hierarchy. Subtypes of complex types inherit all the features of their parents, together with the constraints on their values. Sanfilippo (1993), for example, uses a type hierarchy to encode the hierarchical structure of the lexicon. Fig. 15.14 shows a small part of this hierarchy, the part that models the various subcategories of verbs which take sentential complements; these are divided into the transitive ones (which take direct objects: (*ask yourself whether you have become better informed*) and the intransitive ones (*Monsieur asked whether I wanted to ride*). The type **trans-comp-cat** would introduce the required direct object, constraining it to be of type **noun-phrase**, while types like **sbase-comp-cat** would introduce the baseform (bare stem) complement and constrain its vform to be the baseform.

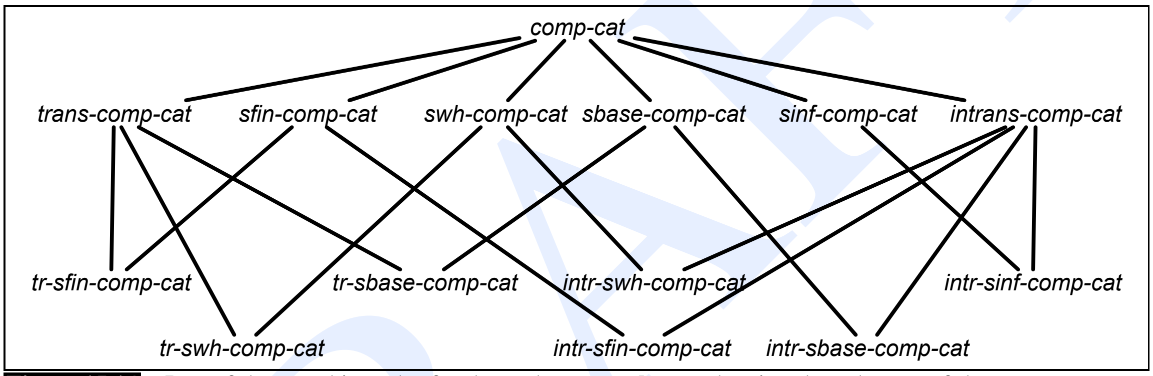

Figure 15.14 Part of the type hierarchy for the verb type **verb-cat**, showing the subtypes of the **comp-cat** type. These are all subcategories of verbs which take sentential complements. After Sanfilippo (1993).

15.6.1 Advanced: Extensions to Typing

Defaults

Priority union

Typed feature structures can be extended by allowing for inheritance with **defaults**. Default systems have mainly been used in lexical type hierarchies of the sort described in the previous section, in order to encode generalizations and subregular exceptions to them. In early versions of default unification the operation was order-dependent, based on the **priority union** operation (Kaplan, 1987). More recent architectures are order-independent (Lascarides and Copestake, 1997; Young and Rounds, 1993), related to Reiter's default logic (Reiter, 1980).

Lexical rule

Many unification-based theories of grammar, including HPSG (Pollard and Sag, 1987, 1994) and LFG (Bresnan, 1982) use an additional mechanism besides inheritance for capturing lexical generalizations: the **lexical rule**. Lexical rules (Jackendoff, 1975) express lexical generalizations by allowing a reduced, hence more redundancy-free lexicon to be automatically expanded by the rules. See Pollard and Sag (1994) for examples, Carpenter (1991) on complexity issues, and Meurers and Minnen (1997) on efficient implementation. Conversely, see Krieger and Nerbonne (1993) on using the type hierarchy to replace lexical rules.

Types can also be used to represent constituency. Rules like (15.12) on page 505 used a normal phrase structure rule template and added the features via path equations. Instead, it's possible to represent the whole phrase structure rule as a type. In order to do this, we need a way to represent constituents as features. One way to do this, following Sag and Wasow (1999), is to take a type **phrase** which has a feature called DTRS ("daughters"), whose value is a list of **phrases**. For example the phrase *I love New York* could have the following representation, (showing only the DTRS feature):

$$\begin{bmatrix} \textbf{phrase} \\ \text{DTRS} \left\langle \begin{bmatrix}\text{CAT } \textit{PRO}\\ \text{ORTH } \textit{I}\end{bmatrix}, \begin{bmatrix}\text{CAT } \textit{VP}\\ \text{DTRS} \left\langle \begin{bmatrix}\text{CAT } \textit{Verb}\\ \text{ORTH } \textit{love}\end{bmatrix}, \begin{bmatrix}\text{CAT } \textit{NP}\\ \text{ORTH } \textit{New York}\end{bmatrix}\right\rangle\end{bmatrix}\right\rangle\end{bmatrix}$$

15.6.2 Other Extensions to Unification

Path inequalities

Negation

Set-valued features

Disjunction

There are many other extensions to unification besides typing, including **path inequalities** (Moshier, 1988; Carpenter, 1992; Carpenter and Penn, 1994), **negation** (Johnson, 1988, 1990), **set-valued features** (Pollard and Moshier, 1990), and **disjunction** (Kay, 1979; Kasper and Rounds, 1986). In some unification systems these operations are incorporated into feature structures. Kasper and Rounds (1986) and others, by contrast, implement them in a separate metalanguage which is used to *describe* feature structures. This idea derives from the work of Pereira and Shieber (1984), and even earlier work by Kaplan and Bresnan (1982), all of whom distinguished between a metalanguage for describing feature structures and the actual feature structures themselves. The descriptions may thus use negation and disjunction to describe a set of feature structures (i.e., a certain feature must not contain a certain value, or may contain any of a set of values) but an actual instance of a feature structure that meets the description would not have negated or disjoint values.

The unification grammars as described so far have no mechanism for disambiguation. Much recent work in unification grammars has focused on this disambiguation problem, particular via the use of probabilistic augmentations. See the History section for important references.

15.7 Summary

This chapter introduced feature structures and the unification operation which is used to combine them.

- A feature structure is a set of features-value pairs, where features are unanalyzable atomic symbols drawn from some finite set, and values are either atomic symbols or feature structures. They are represented either as **attribute-value matrices** (**AVMs**) or as directed acyclic graphs (**DAGs**), where features are directed labeled edges and feature values are nodes in the graph.

- **Unification** is the operation for both combining information (merging the information content of two feature structures) and comparing information (rejecting the merger of incompatible features).
- A phrase-structure rule can be augmented with feature structures, and with feature constraints expressing relations among the feature structures of the constituents of the rule. **Subcategorization** constraints can be represented as feature structures on head verbs (or other predicates). The elements which are subcategorized for by a verb may appear in the verb phrase or may be realized apart from the verb, as a **long-distance dependency**.
- Feature structures can be **typed**. The resulting **typed feature structures** place constraints on which type of values a given feature can take, and can also be organized into a **type hierarchy** to capture generalizations across types.

Bibliographical and Historical Notes

The use of features in linguistic theory comes originally from phonology. Anderson (1985) credits Jakobson (1939) with being the first to use features (called **distinctive features**) as an ontological type in a theory, drawing on previous uses of features by Trubetskoi (1939) and others. The semantic use of features followed soon after; see Ch. 19 for the history of componential analysis in semantics. Features in syntax were well established by the 1950s and were popularized by Chomsky (1965).

The unification operation in linguistics was developed independently by Kay (1979) (feature structure unification) and Colmerauer (1970, 1975) (term unification) (see page 13). Both were working in machine translation and looking for a formalism for combining linguistic information which would be reversible. Colmerauer's original Q-system was a bottom-up parser based on a series of rewrite rules which contained logical variables, designed for a English to French machine translation system. The rewrite rules were reversible to allow them to work for both parsing and generation. Colmerauer, Fernand Didier, Robert Pasero, Philippe Roussel, and Jean Trudel designed the Prolog language based on extended Q-systems to full unification based on the resolution principle of Robinson (1965), and implemented a French analyzer based on it (Colmerauer and Roussel, 1996). The modern use of Prolog and term unification for natural language via **Definite Clause Grammars** was based on Colmerauer's (1975) metamorphosis grammars, and was developed and named by Pereira and Warren (1980). Meanwhile Martin Kay and Ron Kaplan had been working with Augmented Transition Network (**ATN**) grammars. An ATN is a Recursive Transition Network (RTN) in which the nodes are augmented with feature registers. In an ATN analysis of a passive, the first NP would be assigned to the subject register, then when the passive verb was encountered, the value would be moved into the object register. In order to make this process reversible, they restricted assignments to registers so that certain registers could only be filled once, that is, couldn't be overwritten once written. They thus moved toward the concepts of logical variables without realizing it. Kay's original unification algorithm was designed for feature structures rather than terms (Kay,

Definite Clause Grammars

ATN

1979). The integration of unification into an Earley-style approach given in Section 15.5 is based on Shieber (1985b).

See Shieber (1986) for a clear introduction to unification, and Knight (1989) for a multidisciplinary survey of unification.

Inheritance and appropriateness conditions were first proposed for linguistic knowledge by Bobrow and Webber (1980) in the context of an extension of the KL-ONE knowledge representation system (Brachman and Schmolze, 1985). Simple inheritance without appropriateness conditions was taken up by number of researchers; early users include Jacobs (1985, 1987). Aït-Kaci (1984) borrowed the notion of inheritance in unification from the logic programming community. Typing of feature structures, including both inheritance and appropriateness conditions, was independently proposed by Calder (1987), Pollard and Sag (1987), and Elhadad (1990). Typed feature structures were formalized by King (1989) and Carpenter (1992). There is an extensive literature on the use of type hierarchies in linguistics, particularly for capturing lexical generalizations; besides the papers previously discussed, the interested reader should consult Evans and Gazdar (1996) for a description of the DATR language, designed for defining inheritance networks for linguistic knowledge representation, Fraser and Hudson (1992) for the use of inheritance in a dependency grammar, and Daelemans et al. (1992) for a general overview. Formalisms and systems for the implementation of constraint-based grammars via typed feature structures include the PAGE system using the TDL language (Krieger and Schäfer, 1994), ALE (Carpenter and Penn, 1994), ConTroll (Götz et al., 1997) and LKB (Copestake, 2002).

Efficiency issues in unification parsing are discussed by Kiefer et al. (1999a), Malouf et al. (2000), and Munteanu and Penn (2004).

Grammatical theories based on unification include Lexical Functional Grammar (LFG) (Bresnan, 1982), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994), Construction Grammar (Kay and Fillmore, 1999), and Unification Categorial Grammar (Uszkoreit, 1986).

Much recent computational work on unification grammars has focused on probabilistic augmentations for disambiguation. Key relevant papers include Abney (1997), Goodman (1997), Johnson et al. (1999), Riezler et al. (2000), Geman and Johnson (2002), Riezler et al. (2002, 2003), Kaplan et al. (2004), Miyao and Tsujii (2005), Toutanova et al. (2005), Ninomiya et al. (2006) and Blunsom and Baldwin (2006).

Exercises

15.1 Draw the DAGs corresponding to the AVMs given in Examples 15.1–15.2.

15.2 Consider the following BERP examples, focusing on their use of pronouns.

I want to spend lots of money.
Tell me about Chez-Panisse.

I'd like to take her to dinner.
She doesn't like Italian.

Assuming that these pronouns all belong to the category *Pro*, write lexical and grammatical entries with unification constraints that block the following examples.

*Me want to spend lots of money.
*Tell I about Chez-Panisse.
*I would like to take she to dinner.
*Her doesn't like Italian.

15.3 Draw a picture of the subsumption semilattice corresponding to the feature structures in Examples 15.3 to 15.7. Be sure to include the most general feature structure [].

15.4 Consider the following examples.

The sheep are baaaaing.
The sheep is baaaaing.

Create appropriate lexical entries for the words *the*, *sheep*, and *baaaaing*. Show that your entries permit the correct assignment of a value to the NUMBER feature for the subjects of these examples, as well as their various parts.

15.5 Create feature structures expressing the different SUBCAT frames for *while* and *during* shown on page 510.

15.6 Alter the pseudocode shown in Figure 15.12 so that it performs the more radical kind of unification-based parsing described on page 524.

15.7 Consider the following problematic grammar suggested by Shieber (1985b).

$$S \rightarrow T$$
$$\langle T \text{ F} \rangle = \text{a}$$

$$T_1 \rightarrow T_2\ A$$
$$\langle T_1 \text{ F} \rangle = \langle T_2 \text{ F F} \rangle$$

$$S \rightarrow A$$
$$A \rightarrow a$$

Show the first *S* state entered into the chart using your modified PREDICTOR from the previous exercise, then describe any problematic behavior displayed by PREDICTOR on subsequent iterations. Discuss the cause of the problem and how in might be remedied.

15.8 Using the list approach to representing a verb's subcategorization frame, show how a grammar could handle any number of verb subcategorization frames with only the following two *VP* rules. More specifically, show the constraints that would have to be added to these rules to make this work.

$$VP \rightarrow Verb$$
$$VP \rightarrow VP\ X$$

The solution to this problem involves thinking about a recursive walk down a verb's subcategorization frame. This is a hard problem; you might consult Shieber (1986) if you get stuck.

15.9 Page 530 showed how to use typed feature structures to represent constituency. Use that notation to represent rules 15.12, 15.13, and 15.14 shown on page 505.