29 Responding to Questions and Commands

In this chapter, you learn how it is possible to capture the knowledge required to *translate English questions or commands* into *relational-database commands.* Thus, you see that, in certain special circumstances, you can arrange for a computer to deal with natural language.

The key idea is to exploit the relatively regular word order and small, domain-specific vocabularies found in the questions and commands typically directed at relational databases. Word-order regularity is captured in *semantic transition-tree grammars*, which are evolutionary descendants of *augmented transition-net grammars*, usually referred to as ATN grammars.[†]

By way of illustration, you see how semantic transition-tree grammars enable programs to translate English questions and commands involving tools into database commands.

You should understand, however, that the ideas involved are just barely strong enough to support database retrieval via natural language. The problem of understanding unconstrained text seems unthinkably difficult, and worse yet, the more you understand about natural language, the more difficult the problems seem.

Once you have finished this chapter, you will be able to make simple extensions to the sample grammar, targeting it, if you like, to another database.

[†]ATN grammars have never been popular with linguists because they are partly specified by unrestricted computer programs, which linguists legitimately argue are a poor representation for linguistic knowledge.

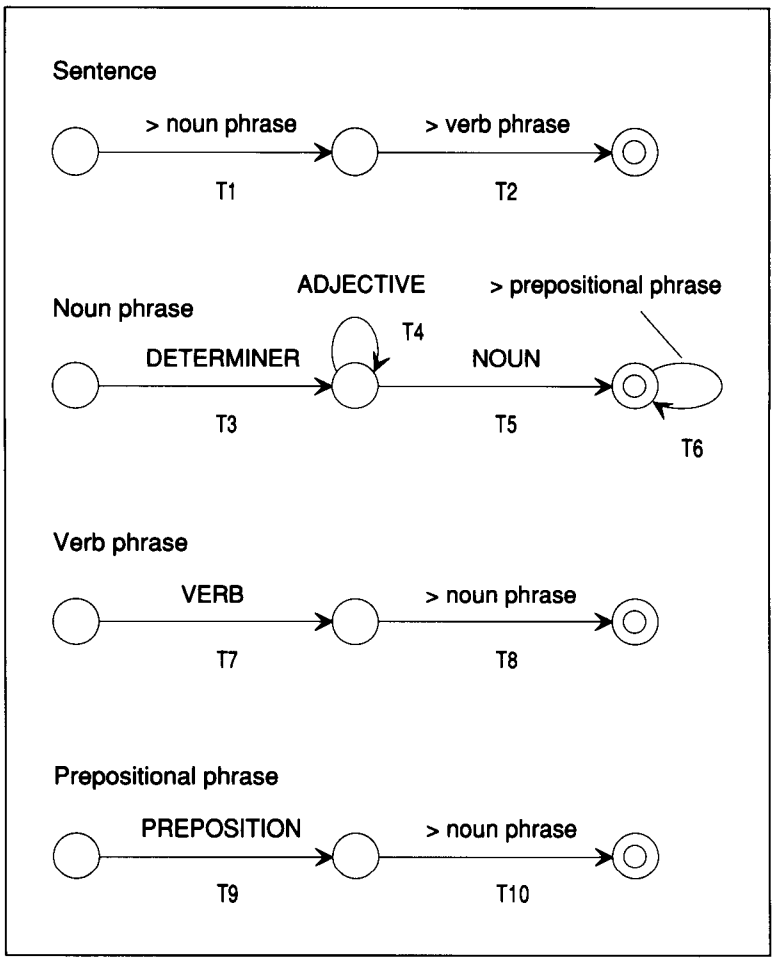

Figure 29.1 A simple transition-net grammar.

SYNTACTIC TRANSITION NETS

In this section, you learn how syntactic transition-net grammars capture linguistic knowledge about word order in English.

Syntactic Transition Nets Are Like Roadmaps

Generally, a **grammar**, viewed as a representation, is a set of conventions for capturing linguistic knowledge of the sort you learned about in elementary school.

A syntactic transition-net grammar consists of a sentence net and a collection of supporting nets, like those shown in figure 29.1.

Think of those nets as though they were an atlas full of roadmaps, and think of words as directions for driving through those roadmaps. To test a string of words to see whether it constitutes a valid sentence, you try to move along a series of links from the initial node in the sentence net

to a terminal node—one with a circle in the center—using the words as directions. If a particular string of words enables you to reach a terminal node, then that string is said to be *accepted* or *recognized* by the transition-net grammar, from which you conclude that the string of words is a *valid sentence* with respect to the grammar.

Consider the sentence net shown in figure 29.1, for example. To move through it, you must first traverse the link labeled > *noun phrase*, which means that you must successfully traverse the noun-phrase net.

To move through the noun-phrase net, you must have a sequence of words starting with a determiner, a word such as *a* or *the*; followed by zero or more adjectives, such as *powerful*, *long*, or *big*; followed by a noun, such as *computer*, *screwdrivers*, or *table*. Each time a word enables you to cross a link, that word is removed from the list of words to be analyzed.

The noun may be followed by a prepositional phrase, such as *on the big table*. The words in prepositional phrases are traversed using the prepositional-phrase net.

Once you have traversed the noun-phrase net, you must move through the verb-phrase net, as indicated by the link labeled > *verb phrase* in the sentence net. Providing that all the words in the sentence are accounted for in traversing the verb-phrase net, your analysis is complete, because you end up at a terminal node, one marked as a double circle.

A Powerful Computer Counted the Long Screwdrivers on the Big Table

To see how such an analysis works on an actual sentence, consider the sentence, "A powerful computer counted the long screwdrivers on the big table." To verify that it is, in fact, a valid sentence, you must determine whether the words specify a path through the grammar. Accordingly, you start to move through the sentence net. Just following the entry node, you encounter a noun-phrase link, labeled T1 in figure 29.1. This sends you off to the noun-phrase net, which in turn requires you to attempt to traverse the determiner link, T3, in the noun-phrase net.

Now it is time to look at the first word in the sentence. It is *a*, one of the words in the determiner class. Having consumed *a* on the determiner link, you are in a position to take either the adjective link, T4, or the noun link, T5.

Assume, by convention, that you always are to try the uppermost of the untried links first. Accordingly, you try the adjective link, T4, consuming the word *powerful*.

Moving on to the next word, *computer*, your attempt to try the adjective link fails, so you try the noun link instead. This time, you succeed, inasmuch as the word *computer* is a noun.

This success brings you to the noun-phrase net's terminal node, where there is the possibility of traversing the prepositional-phrase net.

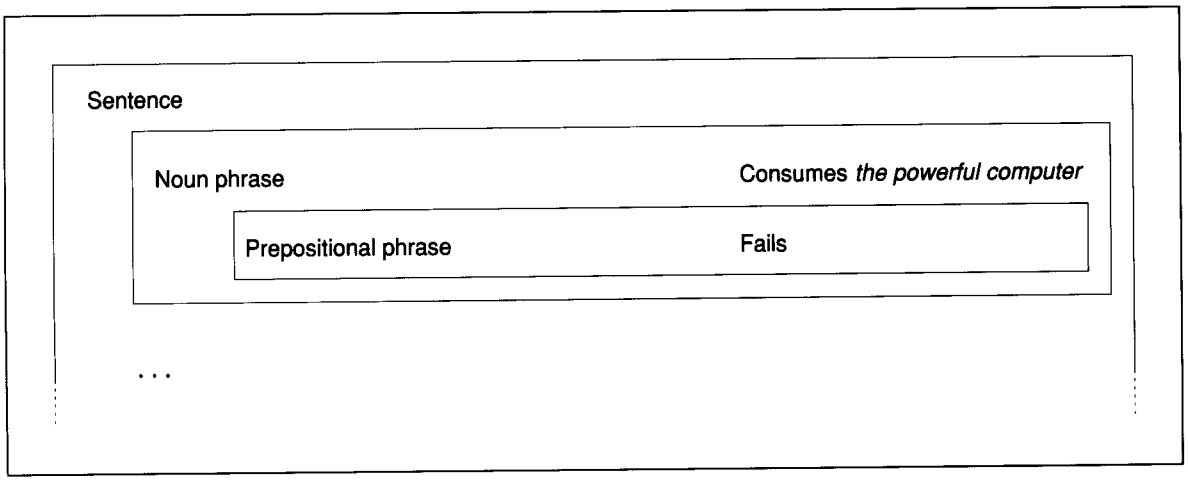

Figure 29.2 The analysis of a sentence beginning "The powerful computer counted . . . ." In this nested-box diagram, graphical *inside* corresponds to a temporal *during* relation among the analyses of the indicated nets.

Assume, by convention, that you are always to try to traverse the net specified by a link, if a link is available, even if you are at a terminal node. Accordingly, you try the prepositional-phrase link, T6, but you see instantly that the next word, *counted*, is not a preposition. You conclude that the prepositional-phrase net cannot be traversed, so you return to the sentence net.

Figure 29.2 shows what has happened so far, from the point of view of net traversals, using a *nested-box diagram*. By convention, a **nested-box diagram** is a diagram in which graphical *inside* corresponds to temporal *during*. Hence, the noun-phrase net must be traversed *during* the analysis of the sentence net because the box describing the noun-phrase traversal is *inside* the box describing the sentence-net traversal. Similarly, the failing attempt to traverse the prepositional-phrase net occurs during the successful analysis of the noun-phrase net.

Next, you have to see whether the remaining words in the sentence can get you through the verb-phrase net, as required by link T2. After you traverse the link labeled T7 with the verb *counted*, link T8 tells you that you have to look for another noun phrase. This search takes you back to the noun-phrase net again.

You quickly proceed through T3, T4, and T5, with the words *the long screwdrivers*. Then, you try to traverse the prepositional-phrase link, link T6. To get through the corresponding prepositional-phrase net, you need a preposition, as dictated by link T9, and a noun phrase, as dictated by link T10. The word *on* is a preposition, and *the big table* is a noun phrase, so you get through the prepositional-phrase net, and return successfully to the noun-phrase net. As there are no more prepositional phrases, a subsequent attempt to go through the propositional-phrase net fails. You are in a terminal noun-phrase node, however, so the noun-phrase net is traversed successfully.

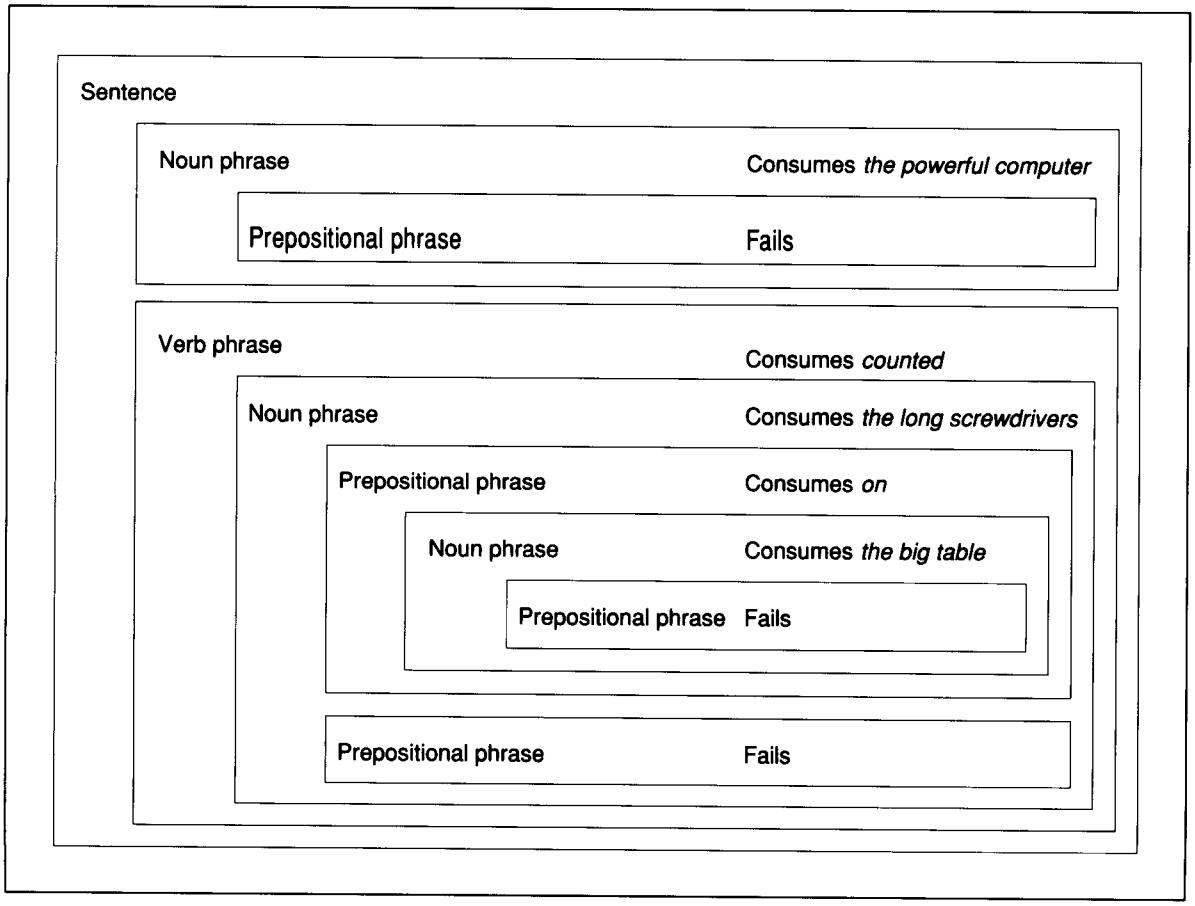

Figure 29.3 Analysis of the sentence, "The powerful computer counted the long screwdrivers on the big table," using a semantic transition-tree grammar. The comments show the words that are consumed as nets are traversed or else indicate failure.

Now you are in the verb-phrase net's terminal node, indicating a successful traversal of a verb-phrase net as well. Having finished with a verb-phrase net, you return to the sentence net, where you find you are, again, in a terminal node. Because there are no more words, you conclude that the sentence is valid with respect to the grammar.

Everything you did during this analysis is summarized, in nested-box form, in figure 29.3.

Such a simple grammar is easy to derail, of course. For example, because the grammar makes no provision for proper names, such as *Hal*, or adverbs, such as *quickly*, it cannot deal with sentences such as "Hal quickly counted the long screwdrivers on the big table."

SEMANTIC TRANSITION TREES

In this section, you learn how it is possible to translate sentences, such as the following, into relational-database commands:

Count the long screwdrivers.
What is the location of the long red screwdriver?

A Relational Database Makes a Good Target

Relational databases, described briefly in Chapter 7 and in more detail in the appendix, consist of one or more *relations*, each of which is a table consisting of labeled columns, called *fields*, and data-containing rows, called *records*. The following example, the tools relation, consists of eight records with entries for the class, color, size, weight, and location fields.

Class	Color	Size	Weight	Location
Saw	black	medium	heavy	pegboard
Hammer	blue	large	heavy	workbench
Wrench	gray	small	light	pegboard
Wrench	gray	large	heavy	pegboard
Screwdriver	blue	long	light	workbench
Screwdriver	black	long	light	toolchest
Screwdriver	red	long	heavy	toolchest
Screwdriver	red	short	light	toolchest

From the perspective of this chapter, the most important characteristic of relational databases is that English descriptions often correspond to simple combinations of the relational-database commands, such as SELECT and PROJECT. For example, you can retrieve the long screwdrivers by first selecting the tools whose class field contains *screwdriver*, and then selecting those tools whose size field contains *long*.

Pattern Instantiation Is the Key to Relational-Database Retrieval in English

Now you are ready to understand the overall semantic transition-tree approach to translating an English question or command into database commands:

- Use the question or command to select database-oriented patterns.
- Use the question or command to instantiate and combine the selected patterns.
- Use the completed pattern to retrieve the database records specified in the question or command.
- Use the retrieved database items to respond to the question or command.

For the first example, the command "Count the long screwdrivers," there is one key pattern:

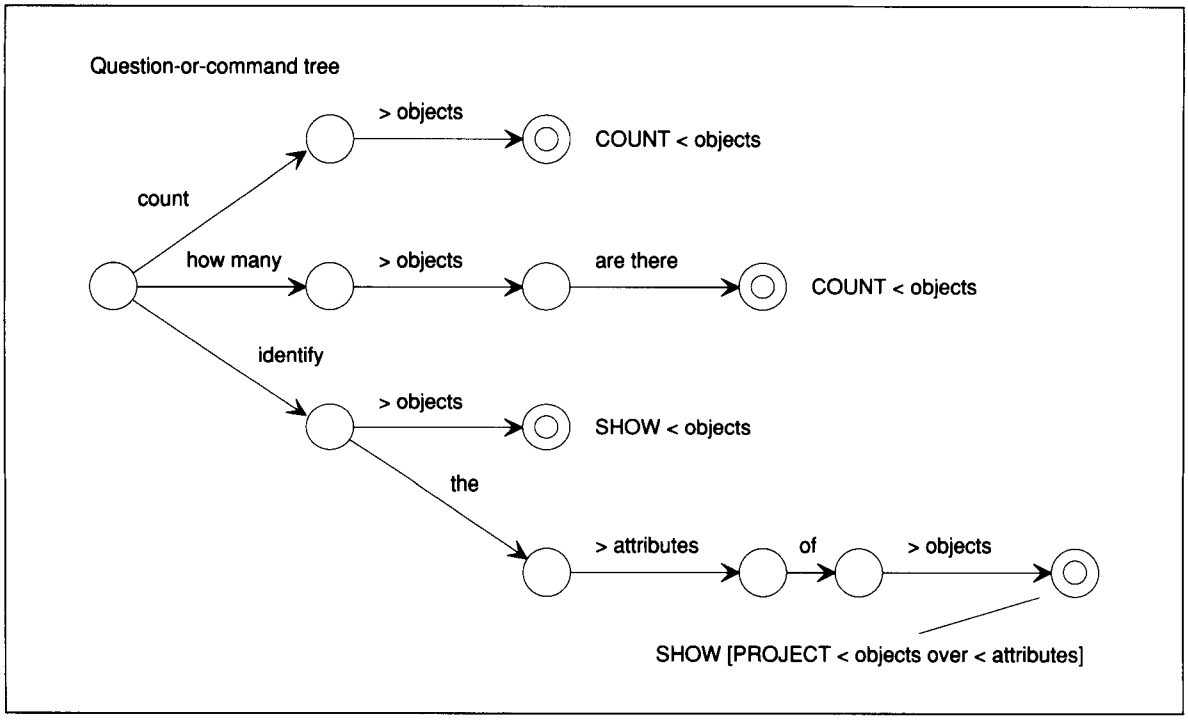

Figure 29.4 The top-level transition tree of a simple semantic grammar.

SELECT < object with < values

Once instantiated, the key pattern looks like this:

SELECT [SELECT Tools with class = screwdrivers]
with size = long

When used on the sample database, the instantiated pattern locates long screwdrivers, as required by the original English command.

Moving from Syntactic Nets to Semantic Trees Simplifies Grammar Construction

A few simple changes to the idea of the syntactic transition net are all that you need to construct systems that use English sentences to drive database retrieval. Evidently, you avoid the overwhelming complexity of natural language, as long as you are able to restrict yourself to database-oriented questions and commands, proscribing declaratives.

The resulting changes lead to the idea of the *semantic transition-tree grammar*. An example of such a grammar is shown in figures 29.4 through 29.7.

From those figures, you see that semantic transition-tree grammars differ from the syntactic transition-net grammars in several ways:

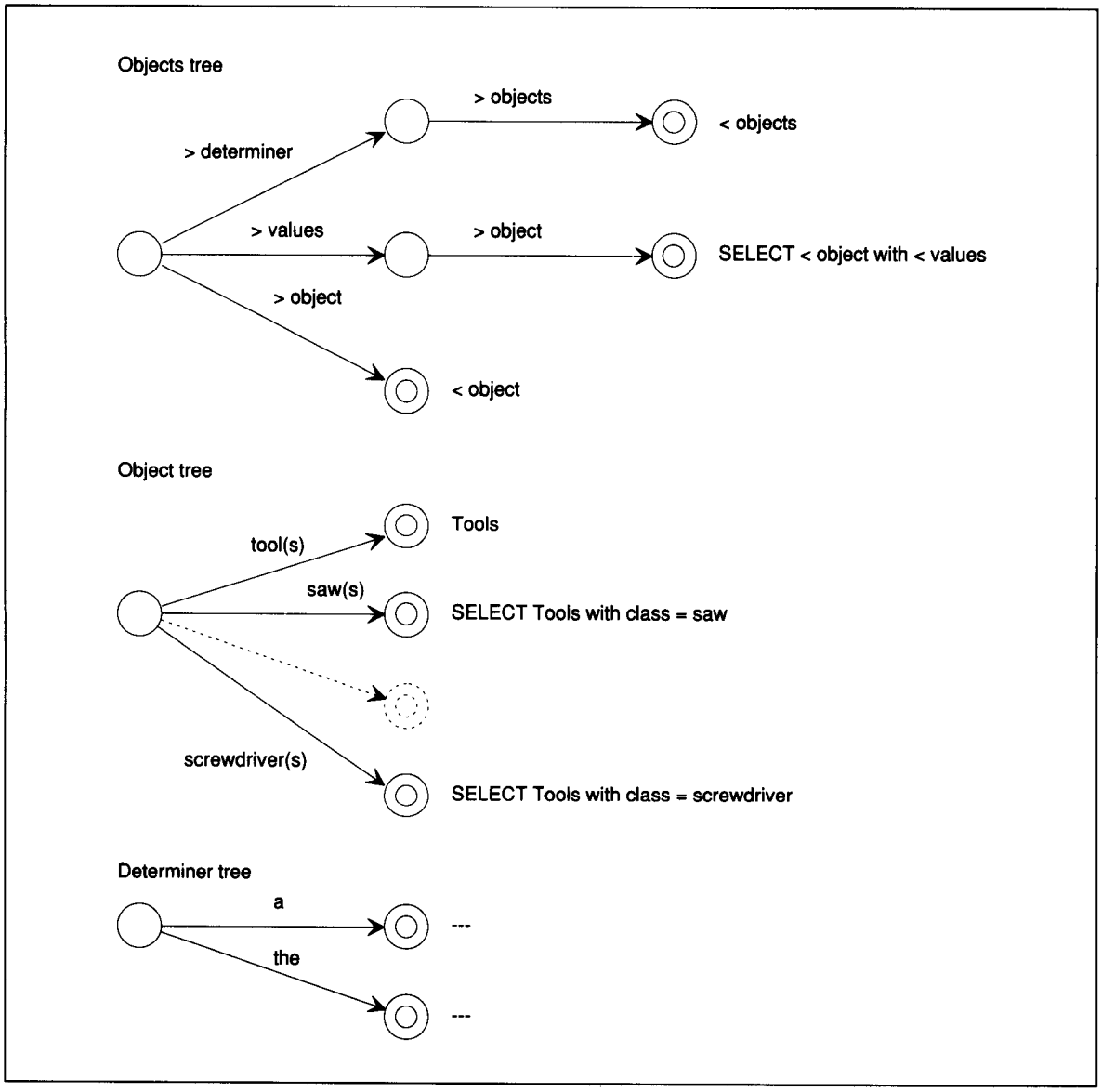

Figure 29.5 The semantic transition trees that enable the analysis of object descriptions.

- Some link transitions require specific words.

For example, a link can be labeled with a word, such as *count*, as in the top-level tree shown in figure 29.4. This link indicates that the first word in the sentence must be *count*. A link also can be labeled with a word followed by *(s)*, such as *tool(s)*, as in the object tree shown in figure 29.5. This link indicates that the next word must be either the word *tool* or its plural, *tools*.

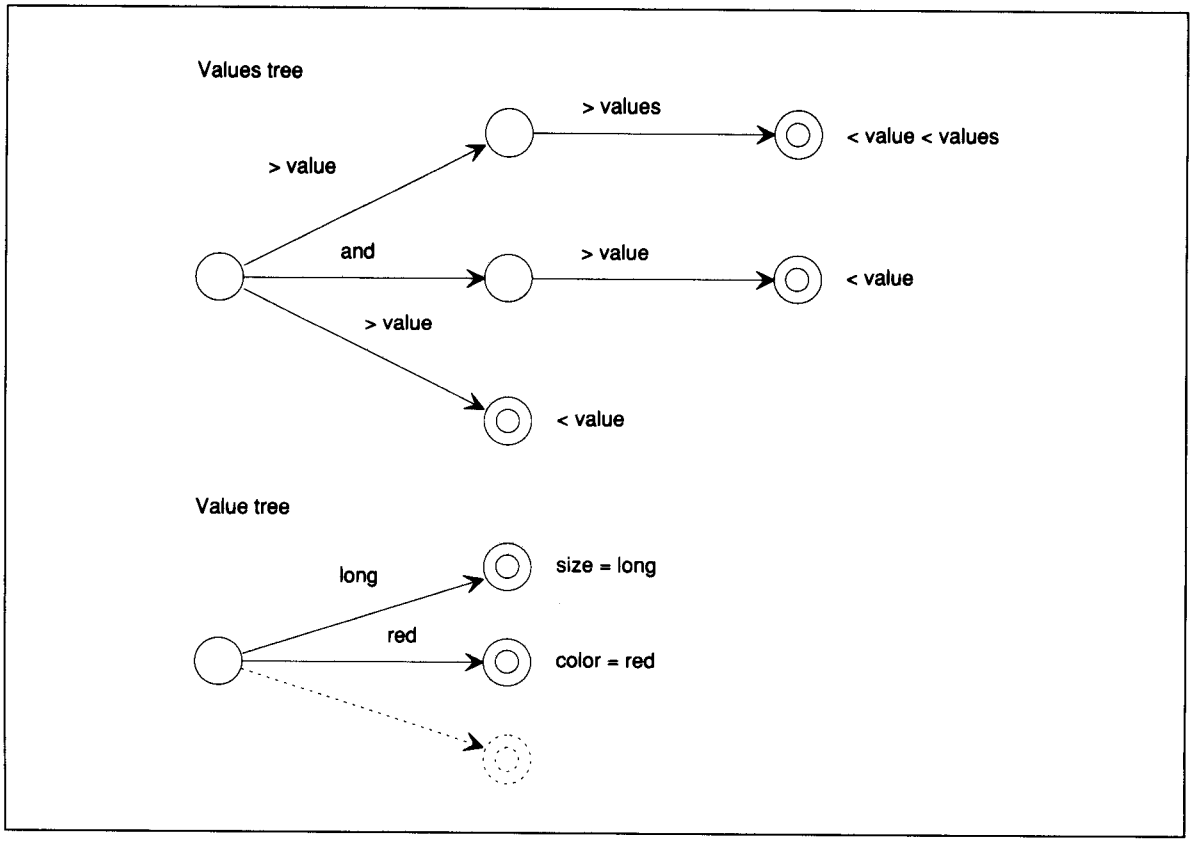

Figure 29.6 The semantic transition trees that enable the analysis of value descriptions.

- Some link transitions specify phrases semantically, rather than syntactically.

Consider, for example, the objects link and the attributes link in figure 29.4. These links—the ones labeled with > *objects* and > *attributes*—focus on database entries, such as those for screwdrivers and sizes, rather than on linguistic entities, such as nouns and adjectives. Accordingly, both the objects link and the attributes link are said to be semantically specified transitions, reflecting the change in name from *syntactic* grammar to *semantic* grammar.

- There are no nodes with two inputs.

This characteristic is why we are talking about transition-*tree* semantic grammars rather than transition-*net* syntactic grammars.

Because of the change from nets to trees, there is a one-to-one correspondence between paths and terminal nodes. Accordingly, once you arrive at a terminal node, there is never any question about how you got there, which makes it easy for you to decide which pattern, if any, to use.

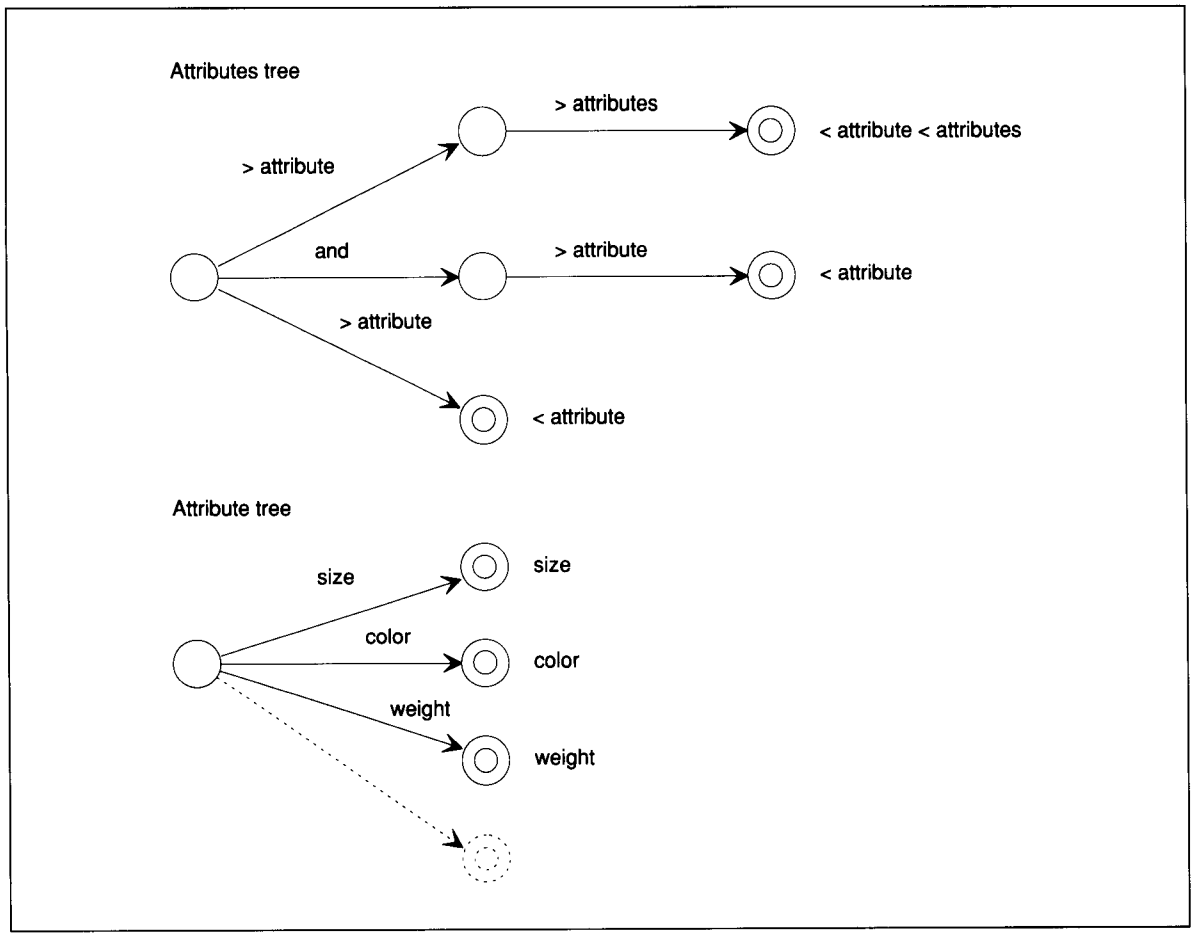

Figure 29.7 The semantic transition trees that enable the analysis of attribute descriptions.

By way of contrast, compare the attribute tree shown in figure 29.7 with the attribute net shown in figure 29.8. In the attribute net, all links converge on one terminal node, so just knowing that you are at that terminal node tells you nothing about how you got there or what you should do. Of course, you could, in principle, keep track of how you got there via some sort of bookkeeping mechanism off to the side, but that alternative would complicate your procedure and diminish your grammar's transparency.

- Whenever a tree is traversed successfully, the tree's name is considered to be a **tree variable**, and is bound to a pattern located at a terminal node. Accordingly, you can refer to the pattern as the tree variable's binding.

Sometimes, a pattern contains one or more tree variables, each of which is marked by a left bracket, <. Such patterns act like templates.

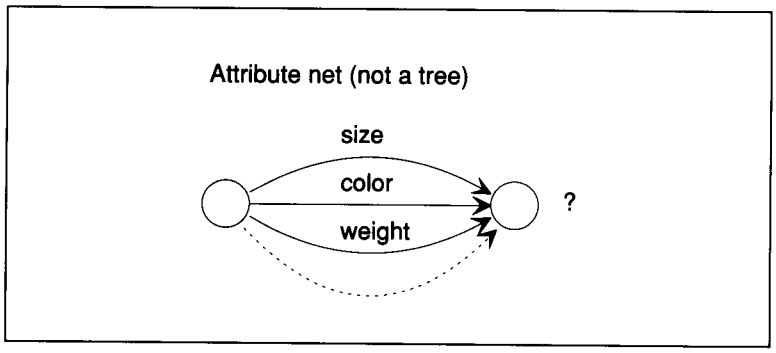

Figure 29.8 An alternate rendering of the attribute tree. Because the attribute tree is shown as a net, rather than as a tree, this rendering cannot be part of a semantic transition-tree grammar.

All the paths traversing the objects tree in figure 29.5 lead to template-like patterns. All the paths traversing the object tree and the determiner tree lead to variable-free patterns.

- Tree variables, marked by left bracket symbols, <, are replaced by their previously established bindings.

For example, the pattern attached to the terminal node of the upper path in the values tree, shown in figure 29.6, contains two marked tree variables, value and values. Whenever this pattern is instantiated, the marked tree variables are replaced by the bindings that were established inside the value tree and the values tree. Thus, the right bracket, >, is a mnemonic preface that is meant to suggest *take processing down* into a subtree, whereas the left bracket, <, is a mnemonic preface meant to suggest *bring a binding up* from a subtree.

Count the Long Screwdrivers

To see how semantic transition trees work together to instantiate patterns, consider the sentence "Count the long screwdrivers." Its analysis, as you will see, is captured by the nested-box diagram shown in figure 29.9.

The top-level tree, the question-or-command tree, is the first used, of course. One link leading out of the entry node is labeled with the word *count*, which matches the first word in the example sentence, taking you to the objects link, whereupon you move your attention to the objects tree.

The objects tree has three links leading out of the entry node, the first of which moves your attention to the determiner tree. Because the next of the remaining words in the sentence is *the*, the determiner tree is traversed successfully. The next link to be traversed in the objects tree is labeled > *objects*, indicating that you should turn your attention to the objects tree again, now on a second level.

At this point, the remaining words are *long screwdrivers*. The word *long* is not a determiner, so the effort to traverse the determiner link fails. Thus, your attention in the objects tree is now focused on the values link.

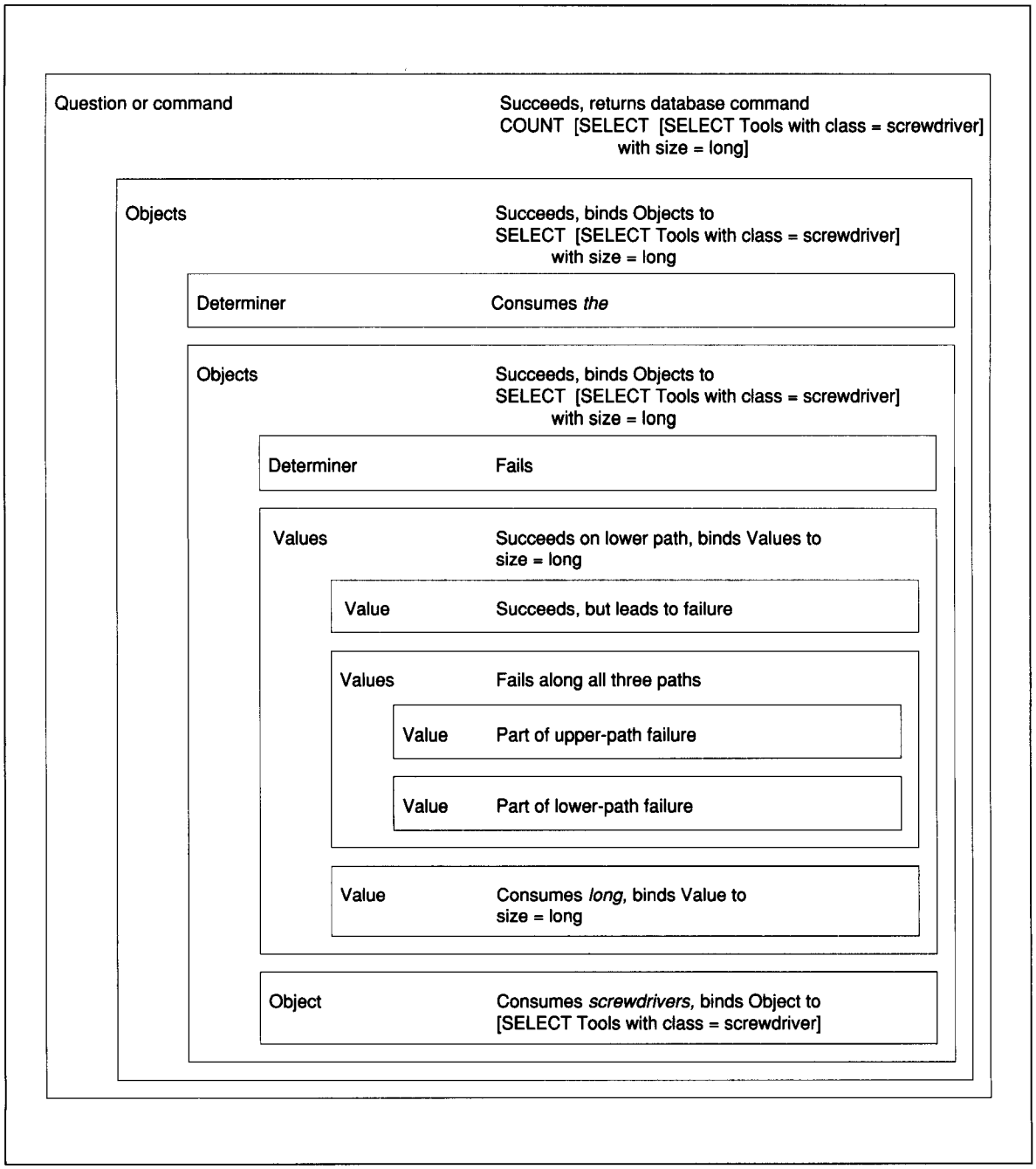

Figure 29.9 The nested-box diagram corresponding to the analysis of "Count the long screwdrivers."

Eventually, you successfully traverse the values tree, consuming the word *long* as you do. To see how you do that, note that you first try to traverse the upper path in the values tree. This attempt requires you to traverse the value tree, which you do easily, consuming the word *long*. Next, however, you must try to traverse the values tree itself on a second level. You fail, however: On that second level, along the upper path, you would need to traverse the value link; along the middle path you would need the word *and*; and along the bottom path you would also need to traverse the value link. Accordingly, you cannot traverse the values tree on a second level, which means, in turn, that you cannot traverse the original values tree along the upper path. You must restore the previously consumed word, *long*, to the remaining words, and try another path from the node where the first alternative happened to be wrong.

You then see that the second path does not work either, because the second path requires the word *and* whereas the first of the remaining words is *long*.

Fortunately, however, the third path requires only the traversal of the value tree, which you have done before and can do again with the word *long*.

Now it is time to think about the consequences of successful traversals. Recall that the successful traversal of a tree causes the tree's name to be bound to the instantiated pattern found at the terminal node. Thus, the successful traversal of the value tree binds value, the tree's name, to the pattern *size = long*.

Similarly, successful traversal of the values tree along the lower path binds the word values to the instantiated form of the pattern *< value*. Because instantiation just replaces tree variables with their bindings, the instantiated form of *< value* is *size = long*. Thus, the tree variables values and value are both bound to *size = long*.

Now recall that all the attention to the values and value trees was launched while you were working on the middle path of the second-level objects tree. Having traversed the values link successfully, you next turn your attention to the object tree, with only the word *screwdrivers* left to be consumed.

Fortunately, there is a link in the object tree labeled with the word *screwdrivers*. Hence, you traversed the object tree successfully, binding the tree variable object to the instantiated pattern, SELECT Tools with class = screwdriver.

At this point, you can see that the meaning of the word *screwdriver*, in this limited context, is captured by a database command that retrieves all the screwdrivers from the tools relation when the database command is executed. After you have traversed the second-level objects tree through the middle path, the tree variable objects is bound to the instantiated

form of the pattern, SELECT < object with < values, which is, of course, as follows:

```
SELECT   [SELECT Tools with class = screwdriver]
         with size = long
```

Now you see that the meaning of the word *long* is captured by way of the incorporation of the *size = long* pattern—the one found in the value tree—into a database command.

Having traversed the second-level objects tree, you revert your attention to the original objects tree—the one that consumed the word *the* along its upper path. Because the original objects tree just binds the objects tree variable to whatever it is bound to on the next level down, you return your attention to the question-or-command tree with the same binding produced by the second-level objects tree.

Now you have just finished your traversal of the upper path in the question-or-command tree. Instantiating the upper path's pattern produces the following, which delivers the required result when the database command is executed:

```
COUNT   [SELECT   [SELECT Tools with class = screwdriver]
                  with size = long]
```

Recursion Replaces Loops

Note that semantic transition trees have no loops of the sort used in syntactic transition nets for adjectives and prepositional phrases. Luckily, no loops are needed, because you can replace loops with trees that use themselves; or said more technically, you can replace loops with recursive tree analysis. The example grammar exhibits this replacement in the objects–object trees, the values–value trees, and the attributes–attribute trees.

To see how such tree pairs handle multiple words, suppose that you are just about to try the values tree and that the remaining words are *long red screwdrivers*. Clearly, you cross the value link on the upper path with the word *long*, leading to another, recursive attempt to cross the values tree, this time with the words *red screwdriver*. This time, after failing on the upper two paths, you succeed on the lower path, successfully traversing the lower level values tree as well as the upper. Figure 29.10 shows how the values tree is traversed in detail.

Thus, you see that the values–value combination handles two word-value sequences successfully. Longer sequences require more recursion—nothing else.

In summary, here are the procedures for traversing semantic transition trees and transition-tree links.

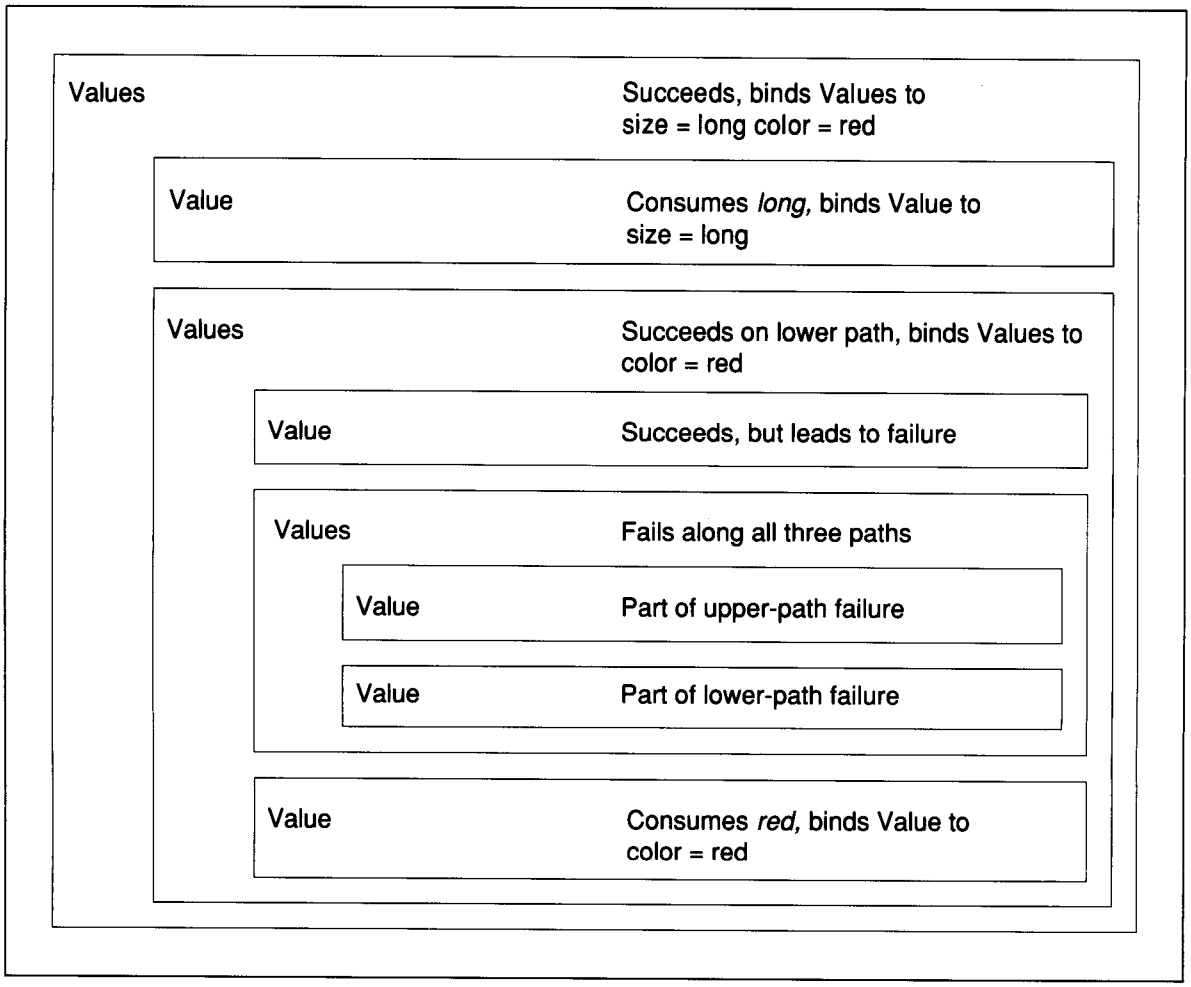

Figure 29.10
Recursion replaces looping in semantic transition trees. Here, the words *long* and *red* are consumed by the values tree and the value tree working together.

To traverse a transition tree,

- ▷ Determine whether it is possible to reach a success node, denoted by a double circle, via word and subtree links.
  - ▷ If it is not possible, announce failure.
  - ▷ Otherwise,
    - ▷ Instantiate the pattern associated with the double circle. Replace pattern variables, marked by < prefixes, with bindings established as subtree links are traversed.
    - ▷ Bind the tree's name to the instantiated pattern.
    - ▷ Announce success.

To traverse a link,

- ▷ If the link is a subtree link, marked by a right bracket, >, and the name of a subtree, try to traverse the subtree. If the traversal is successful, bind the subtree name to the instantiated pattern found at the terminal node of the subtree.
- ▷ If the link is a word link, the next word in the sentence must be that word. The word is consumed as the link is traversed.

SUMMARY

- A full understanding of natural language lies beyond the present state of scientific knowledge. Nevertheless, it is possible to achieve engineering goals in limited contexts.
- One way to interpret questions and commands, in the limited context of database access, is to use the words in the questions and commands to guide you through a collection of nets or trees. You collect information in the course of the movement that makes it possible for you to instantiate database retrieval patterns.
- Moving from syntactic nets to syntactic trees eliminates loops and simplifies pattern instantiation. Moving from syntactic trees to database-specific semantic trees exposes constraints and simplifies tree construction.
- Some links in semantic transition trees are traversed by individual words. Others require the traversal of a subtree.
- When a subtree is traversed successfully, a pattern is instantiated, and that instantiated pattern becomes the value bound to the subtree's name. When the top-level tree is traversed successfully, a pattern is instantiated, and that instantiated pattern is used to access a database.

BACKGROUND

The notion of an augmented transition net was introduced in a paper by J. Thorne, P. Bratley, and H. Dewar [1968]. Work by Daniel G. Bobrow and Bruce Fraser [1969] and by William A. Woods [1970] developed and popularized the idea soon thereafter. Woods, especially, became a major contributor.

The work of Woods on his LUNAR system, and the work of Terry Winograd on his SHRDLU system [1971], were the precursors to today's commercial language interfaces, for they showed that sentence-analysis procedures

Q&A Translates Questions into Database-Retrieval Commands

The popular Q&A system is one example of a database system with a practical natural-language interface based on the semantic-grammar approach. Q&A's semantic grammar is much more complete than is the one you learned about in this chapter, but it basically provides the same capabilities.

Suppose, for example, that you are an investor. You can use Q&A to keep track of information about the companies in which you are an investor, enabling a dialog:

> Tell me the name of the company that makes Q&A?

Q&A translates the question into a database command aimed at a database containing company and product fields. The response to the database command enables Q&A to print a one-record table:

Company	Product
Semantec	Q&A

> Which product category is Q&A?

Product	Product category
Q&A	Database software

> Count the number of software companies.

Q&A responds with a number. The command is easy to obey for any system based on a semantic grammar, because it is, after all, not much different from the command, "Count the long screwdrivers."

> Which companies in Massachusetts are software companies?

Responding, Q&A prints another table:

Company	Business	City	State
Ascent Technology	Software	Cambridge	MA
Bachman Associates	Software	Burlington	MA
⋮			

> What are the product categories of Ascent Technology's products?

Responding, Q&A prints still another table:

Product	Product category
Aris	Resource allocation
Ferret	Database mining
⋮	

can instantiate search-procedure slots. Woods's work involved questions about moon rocks. Winograd's work involved questions about a simulated world of blocks.

For a good exposition of semantic transition-tree grammars, see work by Gary Hendrix and his associates [Hendrix et al. 1978]. The well-known LIFER system is based on this work, and it led eventually to Q&A, which provides natural-language access to databases.