Artificial Intelligence

Second Edition

Elaine Rich

Microelectronics and Computer Technology Corporation

Kevin Knight

Carnegie Mellon University

§ 10.1: Conceptual Dependency
10.2: Scripts

© 1991

McGraw-Hill, Inc.
New York St. Louis San Francisco Auckland Bogotá Caracas
Hamburg Lisbon London Madrid Mexico Milan Montreal
New Delhi Paris San Juan São Paulo Singapore Sydney
Tokyo Toronto

$$\begin{tikzcd}
 & & & & \text{man} \\
I \arrow[r, Leftrightarrow, "p"] & \text{ATRANS} & \text{book} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur, "\text{to}"] & \\
 & & & & I \arrow[ul, "\text{from}"]
\end{tikzcd}$$

where the symbols have the following meanings:

- Arrows indicate direction of dependency.
- Double arrow indicates two way link between actor and action.
- p indicates past tense.
- ATRANS is one of the primitive acts used by the theory. It indicates transfer of possession.
- o indicates the object case relation.
- R indicates the recipient case relation.

Figure 10.1: A Simple Conceptual Dependency Representation

As a simple example of the way knowledge is represented in CD, the event represented by the sentence

I gave the man a book.

would be represented as shown in Figure 10.1.

In CD, representations of actions are built from a set of primitive acts. Although there are slight differences in the exact set of primitive actions provided in the various sources on CD, a typical set is the following, taken from Schank and Abelson [1977]:

ATRANS	Transfer of an abstract relationship (e.g., give)
PTRANS	Transfer of the physical location of an object (e.g., go)
PROPEL	Application of physical force to an object (e.g., push)
MOVE	Movement of a body part by its owner (e.g., kick)
GRASP	Grasping of an object by an actor (e.g., clutch)
INGEST	Ingestion of an object by an animal (e.g., eat)
EXPEL	Expulsion of something from the body of an animal (e.g., cry)
MTRANS	Transfer of mental information (e.g., tell)
MBUILD	Building new information out of old (e.g., decide)
SPEAK	Production of sounds (e.g., say)
ATTEND	Focusing of a sense organ toward a stimulus (e.g., listen)

Chapter 10

Strong Slot-and-Filler Structures

The slot-and-filler structures described in the previous chapter are very general. Individual semantic networks and frame systems may have specialized links and inference procedures, but there are no hard and fast rules about what kinds of objects and links are good in general for knowledge representation. Such decisions are left up to the builder of the semantic network or frame system.

The three structures discussed in this chapter, *conceptual dependency*, *scripts*, and *CYC*, on the other hand, embody specific notions of what types of objects and relations are permitted. They stand for powerful theories of how AI programs can represent and use knowledge about common situations.

10.1 Conceptual Dependency

Conceptual dependency (often nicknamed CD) is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences. The goal is to represent the knowledge in a way that

- Facilitates drawing inferences from the sentences.
- Is independent of the language in which the sentences were originally stated.

Because of the two concerns just mentioned, the CD representation of a sentence is built not out of primitives corresponding to the words used in the sentence, but rather out of conceptual primitives that can be combined to form the meanings of words in any particular language. The theory was first described in Schank [1973] and was further developed in Schank [1975]. It has since been implemented in a variety of programs that read and understand natural language text. Unlike semantic nets, which provide only a structure into which nodes representing information at any level can be placed, conceptual dependency provides both a structure and a specific set of primitives, at a particular level of granularity, out of which representations of particular pieces of information can be constructed.

$$\begin{tikzcd}
 & & & & \text{man} \\
\text{I} \arrow[r, leftrightarrow, "p"] & \text{ATRANS} & \text{book} \arrow[l, "o"'] & {} \arrow[l, "R"'] \arrow[ur, "\text{to}"] & \\
 & & & & \text{I} \arrow[ul, "\text{from}"]
\end{tikzcd}$$

where the symbols have the following meanings:

- Arrows indicate direction of dependency.
- Double arrow indicates two way link between actor and action.
- p indicates past tense.
- ATRANS is one of the primitive acts used by the theory. It indicates transfer of possession.
- o indicates the object case relation.
- R indicates the recipient case relation.

Figure 10.1: A Simple Conceptual Dependency Representation

As a simple example of the way knowledge is represented in CD, the event represented by the sentence

I gave the man a book.

would be represented as shown in Figure 10.1.

In CD, representations of actions are built from a set of primitive acts. Although there are slight differences in the exact set of primitive actions provided in the various sources on CD, a typical set is the following, taken from Schank and Abelson [1977]:

ATRANS	Transfer of an abstract relationship (e.g., give)
PTRANS	Transfer of the physical location of an object (e.g., go)
PROPEL	Application of physical force to an object (e.g., push)
MOVE	Movement of a body part by its owner (e.g., kick)
GRASP	Grasping of an object by an actor (e.g., clutch)
INGEST	Ingestion of an object by an animal (e.g., eat)
EXPEL	Expulsion of something from the body of an animal (e.g., cry)
MTRANS	Transfer of mental information (e.g., tell)
MBUILD	Building new information out of old (e.g., decide)
SPEAK	Production of sounds (e.g., say)
ATTEND	Focusing of a sense organ toward a stimulus (e.g., listen)

A second set of CD building blocks is the set of allowable dependencies among the conceptualizations described in a sentence. There are four primitive conceptual categories from which dependency structures can be built. These are

ACTs	Actions
PPs	Objects (picture producers)
AAs	Modifiers of actions (action aiders)
PAs	Modifiers of PPs (picture aiders)

In addition, dependency structures are themselves conceptualizations and can serve as components of larger dependency structures.

The dependencies among conceptualizations correspond to semantic relations among the underlying concepts. Figure 10.2 lists the most important ones allowed by CD.[1] The first column contains the rules; the second contains examples of their use; and the third contains an English version of each example. The rules shown in the figure can be interpreted as follows:

- Rule 1 describes the relationship between an actor and the event he or she causes. This is a two-way dependency since neither actor nor event can be considered primary. The letter p above the dependency link indicates past tense.
- Rule 2 describes the relationship between a PP and a PA that is being asserted to describe it. Many state descriptions, such as height, are represented in CD as numeric scales.
- Rule 3 describes the relationship between two PPs, one of which belongs to the set defined by the other.
- Rule 4 describes the relationship between a PP and an attribute that has already been predicated of it. The direction of the arrow is toward the PP being described.
- Rule 5 describes the relationship between two PPs, one of which provides a particular kind of information about the other. The three most common types of information to be provided in this way are possession (shown as POSS-BY), location (shown as LOC), and physical containment (shown as CONT). The direction of the arrow is again toward the concept being described.
- Rule 6 describes the relationship between an ACT and the PP that is the object of that ACT. The direction of the arrow is toward the ACT since the context of the specific ACT determines the meaning of the object relation.
- Rule 7 describes the relationship between an ACT and the source and the recipient of the ACT.
- Rule 8 describes the relationship between an ACT and the instrument with which it is performed. The instrument must always be a full conceptualization (i.e., it must contain an ACT), not just a single physical object.

[1]The table shown in the figure is adapted from several tables in Schank [1973].

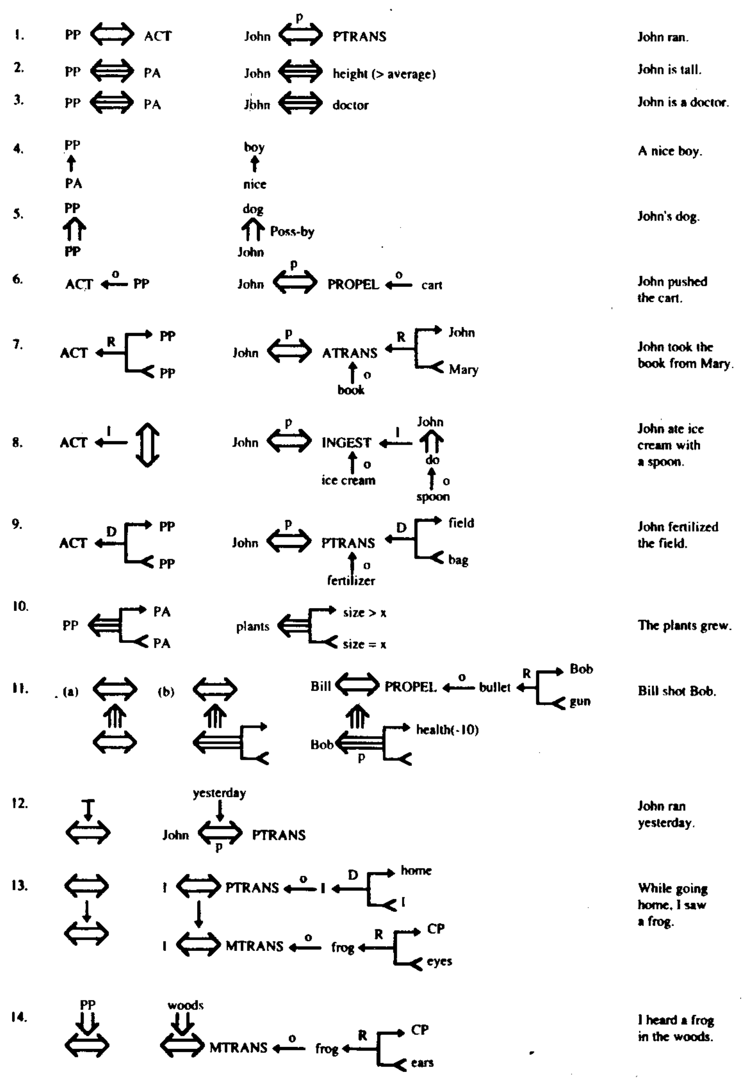


Figure 10.2: The Dependencies of CD

- Rule 9 describes the relationship between an ACT and its physical source and destination.
- Rule 10 represents the relationship between a PP and a state in which it started and another in which it ended.
- Rule 11 describes the relationship between one conceptualization and another that causes it. Notice that the arrows indicate dependency of one conceptualization on another and so point in the opposite direction of the implication arrows. The two forms of the rule describe the cause of an action and the cause of a state change.
- Rule 12 describes the relationship between a conceptualization and the time at which the event it describes occurred.
- Rule 13 describes the relationship between one conceptualization and another that is the time of the first. The example for this rule also shows how CD exploits a model of the human information processing system; *see* is represented as the transfer of information between the eyes and the conscious processor.
- Rule 14 describes the relationship between a conceptualization and the place at which it occurred.

Conceptualizations representing events can be modified in a variety of ways to supply information normally indicated in language by the tense, mood, or aspect of a verb form. The use of the modifier p to indicate past tense has already been shown. The set of conceptual tenses proposed by Schank [1973] includes

p	Past
f	Future
t	Transition
t_s	Start transition
t_f	Finished transition
k	Continuing
?	Interrogative
/	Negative
nil	Present
delta	Timeless
c	Conditional

As an example of the use of these tenses, consider the CD representation shown in Figure 10.3 (taken from Schank [1973]) of the sentence

Since smoking can kill you, I stopped.

The vertical causality link indicates that smoking kills one. Since it is marked c, however, we know only that smoking can kill one, not that it necessarily does. The horizontal causality link indicates that it is that first causality that made me stop smoking. The qualification t_{fp} attached to the dependency between I and INGEST indicates that the smoking (an instance of INGESTING) has stopped and that the stopping happened in the past.

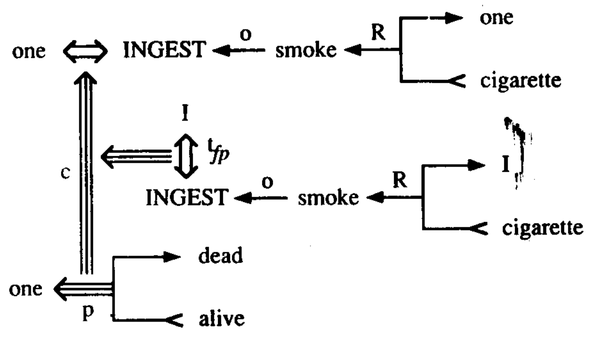


Figure 10.3: Using Conceptual Tenses

There are three important ways in which representing knowledge using the conceptual dependency model facilitates reasoning with the knowledge:

1. Fewer inference rules are needed than would be required if knowledge were not broken down into primitives.

2. Many inferences are already contained in the representation itself.

3. The initial structure that is built to represent the information contained in one sentence will have holes that need to be filled. These holes can serve as an attention focuser for the program that must understand ensuing sentences.

Each of these points merits further discussion.

The first argument in favor of representing knowledge in terms of CD primitives rather than in the higher-level terms in which it is normally described is that using the primitives makes it easier to describe the inference rules by which the knowledge can be manipulated. Rules need only be represented once for each primitive ACT rather than once for every word that describes that ACT. For example, all of the following verbs involve a transfer of ownership of an object:

- Give
- Take
- Steal
- Donate

If any of them occurs, then inferences about who now has the object and who once had the object (and thus who may know something about it) may be important. In a CD representation, those possible inferences can be stated once and associated with the primitive ACT ATRANS.

A second argument in favor of the use of CD representation is that to construct it, we must use not only the information that is stated explicitly in a sentence but also a set

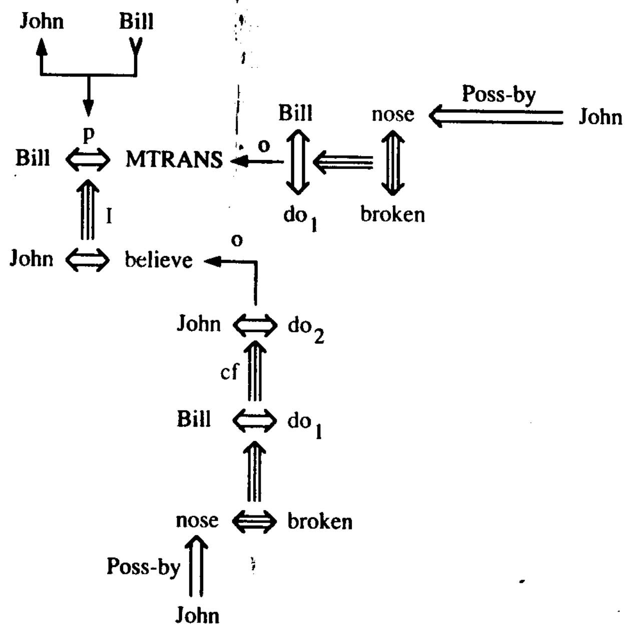


Figure 10.4: The CD Representation of a Threat

of inference rules associated with the specific information. Having applied these rules once, we store these results as part of the representation and they can be used repeatedly without the rules being reapplied. For example, consider the sentence

Bill threatened John with a broken nose.

The CD representation of the information contained in this sentence is shown in Figure 10.4. (For simplicity, *believe* is shown as a single unit. In fact, it must be represented in terms of primitive ACTs and a model of the human information processing system.) It says that Bill informed John that he (Bill) will do something to break John's nose. Bill did this so that John will believe that if he (John) does some other thing (different from what Bill will do to break his nose), then Bill will break John's nose. In this representation, the word "believe" has been used to simplify the example. But the idea behind *believe* can be represented in CD as an MTRANS of a fact into John's memory. The actions do_1 and do_2 are dummy placeholders that refer to some as yet unspecified actions.

A third argument for the use of the CD representation is that unspecified elements of the representation of one piece of information can be used as a focus for the understanding of later events as they are encountered. So, for example, after hearing that

Bill threatened John with a broken nose.

we might expect to find out what action Bill was trying to prevent John from performing. That action could then be substituted for the dummy action represented in Figure 10.4 as do_2. The presence of such dummy objects provides clues as to what other events or objects are important for the understanding of the known event.

Of course, there are also arguments against the use of CD as a representation formalism. For one thing, it requires that all knowledge be decomposed into fairly low-level primitives. In Section 4.3.3 we discussed how this may be inefficient or perhaps even impossible in some situations. As Schank and Owens [1987] put it,

> CD is a theory of representing fairly simple actions. To express, for example, "John bet Sam fifty dollars that the Mets would win the World Series" takes about two pages of CD forms. This does not seem reasonable.

Thus, although there are several arguments in favor of the use of CD as a model for representing events, it is not always completely appropriate to do so, and it may be worthwhile to seek out higher-level primitives.

Another difficulty with the theory of conceptual dependency as a general model for the representation of knowledge is that it is only a theory of the representation of events. But to represent all the information that a complex program may need, it must be able to represent other things besides events. There have been attempts to define a set of primitives, similar to those of CD for actions, that can be used to describe other kinds of knowledge. For example, physical objects, which in CD are simply represented as atomic units, have been analyzed in Lehnert [1978]. A similar analysis of social actions is provided in Schank and Carbonell [1979]. These theories continue the style of representation pioneered by CD, but they have not yet been subjected to the same amount of empirical investigation (i.e., use in real programs) as CD.

We have discussed the theory of conceptual dependency in some detail in order to illustrate the behavior of a knowledge representation system built around a fairly small set of specific primitive elements. But CD is not the only such theory to have been developed and used in AI programs. For another example of a primitive-based system, see Wilks [1972].

10.2 Scripts

CD is a mechanism for representing and reasoning about events. But rarely do events occur in isolation. In this section, we present a mechanism for representing knowledge about common sequences of events.

A *script* is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available. So far, this definition of a script looks very similar to that of a frame given in Section 9.2, and at this level of detail, the two structures are identical. But now, because of the specialized role to be played by a script, we can make some more precise statements about its structure.

Figure 10.5 shows part of a typical script, the restaurant script (taken from Schank and Abelson [1977]). It illustrates the important components of a script:

Entry conditions	Conditions that must, in general, be satisfied before the events described in the script can occur.
Result	Conditions that will, in general, be true after the events described in the script have occurred.
Props	Slots representing objects that are involved in the events described in the script. The presence of these objects can be inferred even if they are not mentioned explicitly.
Roles	Slots representing people who are involved in the events described in the script. The presence of these people, too, can be inferred even if they are not mentioned explicitly. If specific individuals are mentioned, they can be inserted into the appropriate slots.
Track	The specific variation on a more general pattern that is represented by this particular script. Different tracks of the same script will share many but not all components.
Scenes	The actual sequences of events that occur. The events are represented in conceptual dependency formalism.

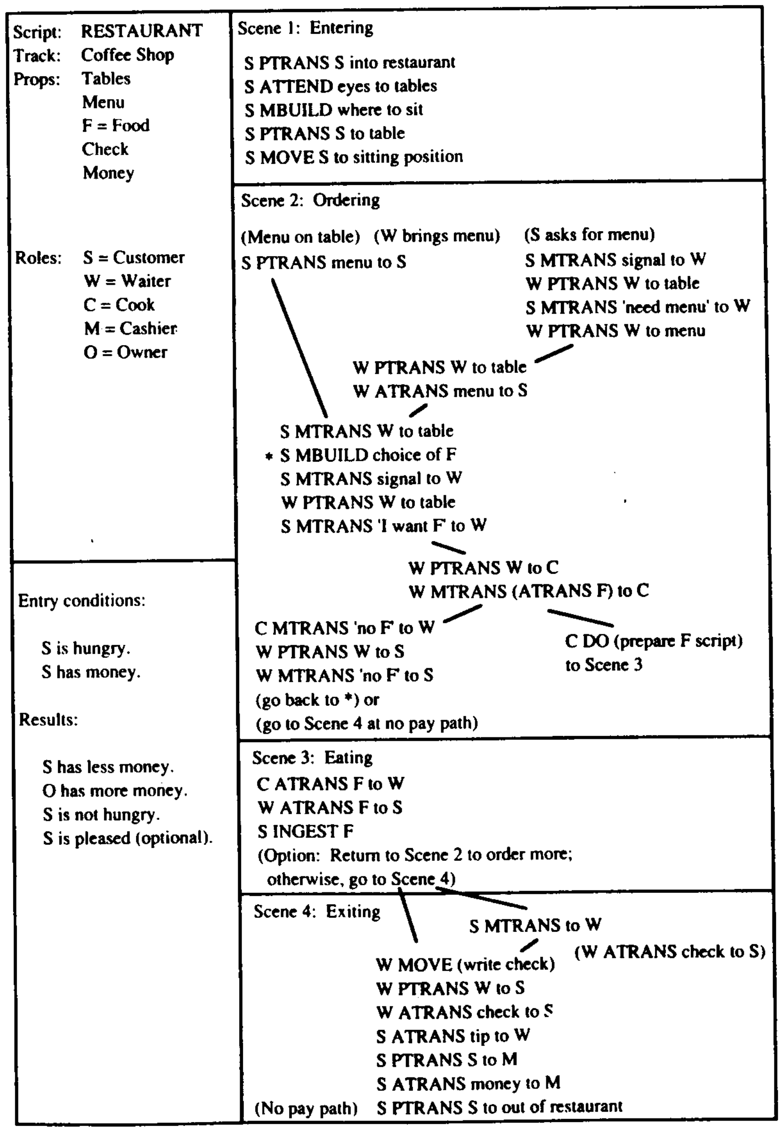
Scripts are useful because, in the real world, there are patterns to the occurrence of events. These patterns arise because of causal relationships between events. Agents will perform one action so that they will then be able to perform another. The events described in a script form a giant *causal chain*. The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events or event sequences (possibly described by other scripts) to occur. Within the chain, events are connected both to earlier events that make them possible and to later events that they enable.

If a particular script is known to be appropriate in a given situation, then it can be very useful in predicting the occurrence of events that were not explicitly mentioned. Scripts can also be useful by indicating how events that were mentioned relate to each other. For example, what is the connection between someone's ordering steak and someone's eating steak? But before a particular script can be applied, it must be activated (i.e., it must be selected as appropriate to the current situation). There are two ways in which it may be useful to activate a script, depending on how important the script is likely to be:

- For fleeting scripts (ones that are mentioned briefly and may be referred to again but are not central to the situation), it may be sufficient merely to store a pointer to the script so that it can be accessed later if necessary. This would be an appropriate strategy to take with respect to the restaurant script when confronted with a story such as

  > Susan passed her favorite restaurant on her way to the museum. She really enjoyed the new Picasso exhibit.

- For nonfleeting scripts it is appropriate to activate the script fully and to attempt to fill in its slots with particular objects and people involved in the current situation.



Script: RESTAURANT
Track: Coffee Shop
Props: Tables
Menu
F = Food
Check
Money

Roles: S = Customer
W = Waiter
C = Cook
M = Cashier
O = Owner

Entry conditions:

S is hungry.
S has money.

Results:

S has less money.
O has more money.
S is not hungry.
S is pleased (optional).

Scene 1: Entering

S PTRANS S into restaurant
S ATTEND eyes to tables
S MBUILD where to sit
S PTRANS S to table
S MOVE S to sitting position

Scene 2: Ordering

(Menu on table) (W brings menu) (S asks for menu)
S PTRANS menu to S
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'need menu' to W
W PTRANS W to menu

W PTRANS W to table
W ATRANS menu to S

S MTRANS W to table
* S MBUILD choice of F
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'I want F' to W

W PTRANS W to C
W MTRANS (ATRANS F) to C

C MTRANS 'no F' to W
W PTRANS W to S
W MTRANS 'no F' to S
(go back to *) or
(go to Scene 4 at no pay path)

C DO (prepare F script)
to Scene 3

Scene 3: Eating

C ATRANS F to W
W ATRANS F to S
S INGEST F
(Option: Return to Scene 2 to order more;
otherwise, go to Scene 4)

Scene 4: Exiting

S MTRANS to W
(W ATRANS check to S)
W MOVE (write check)
W PTRANS W to S
W ATRANS check to S
S ATRANS tip to W
S PTRANS S to M
S ATRANS money to M
(No pay path) S PTRANS S to out of restaurant

Figure 10.5: The Restaurant Script

The headers of a script (its preconditions, its preferred locations, its props, its roles, and its events) can all serve as indicators that the script should be activated. In order to cut down on the number of times a spurious script is activated, it has proved useful to require that a situation contain at least two of a script's headers before the script will be activated.

Once a script has been activated, there are, as we have already suggested, a variety of ways in which it can be useful in interpreting a particular situation. The most important of these is the ability to predict events that have not explicitly been observed. Suppose, for example, that you are told the following story:

> John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money. He hurried home since it had started to rain.

If you were then asked the question

> Did John eat dinner last night?

you would almost certainly respond that he did, even though you were not told so explicitly. By using the restaurant script, a computer question-answerer would also be able to infer that John ate dinner, since the restaurant script could have been activated. Since all of the events in the story correspond to the sequence of events predicted by the script, the program could infer that the entire sequence predicted by the script occurred normally. Thus it could conclude, in particular, that John ate. In their ability to predict unobserved events, scripts are similar to frames and to other knowledge structures that represent stereotyped situations. Once one of these structures is activated in a particular situation, many predictions can be made.

A second important use of scripts is to provide a way of building a single coherent interpretation from a collection of observations. Recall that a script can be viewed as a giant causal chain. Thus it provides information about how events are related to each other. Consider, for example, the following story:

> Susan went out to lunch. She sat down at a table and called the waitress. The waitress brought her a menu and she ordered a hamburger.

Now consider the question

> Why did the waitress bring Susan a menu?

The script provides two possible answers to that question:

- Because Susan asked her to. (This answer is gotten by going backward in the causal chain to find out what caused her to do it.)
- So that Susan could decide what she wanted to eat. (This answer is gotten by going forward in the causal chain to find out what event her action enables.)

A third way in which a script is useful is that it focuses attention on unusual events. Consider the following story:

John went to a restaurant. He was shown to his table. He ordered a large steak. He sat there and waited for a long time. He got mad and left.

The important part of this story is the place in which it departs from the expected sequence of events in a restaurant. John did not get mad because he was shown to his table. He did get mad because he had to wait to be served. Once the typical sequence of events is interrupted, the script can no longer be used to predict other events. So, for example, in this story, we should not infer that John paid his bill. But we can infer that he saw a menu, since reading the menu would have occurred before the interruption. For a discussion of SAM, a program that uses scripts to perform this kind of reasoning, see Cullingford [1981].

From these examples, we can see how information about typical sequences of events, as represented in scripts, can be useful in interpreting a particular, observed sequence of events. The usefulness of a script in some of these examples, such as the one in which unobserved events were predicted, is similar to the usefulness of other knowledge structures, such as frames. In other examples, we have relied on specific properties of the information stored in a script, such as the causal chain represented by the events it contains. Thus although scripts are less general structures than are frames, and so are not suitable for representing all kinds of knowledge, they can be very effective for representing the specific kinds of knowledge for which they were designed.

10.3 CYC

CYC [Lenat and Guha, 1990] is a very large knowledge base project aimed at capturing human commonsense knowledge. Recall that in Section 5.1, our first attempt to prove that Marcus was not loyal to Caesar failed because we were missing the simple fact that all men are people. The goal of CYC is to encode the large body of knowledge that is so obvious that it is easy to forget to state it explicitly. Such a knowledge base could then be combined with specialized knowledge bases to produce systems that are less brittle than most of the ones available today.

Like CD, CYC represents a specific theory of how to describe the world, and like CD, it can be used for AI tasks such as natural language understanding. CYC, however, is more comprehensive; while CD provided a specific theory of representation for events, CYC contains representations of events, objects, attitudes, and so forth. In addition, CYC is particularly concerned with issues of scale, that is, what happens when we build knowledge bases that contain millions of objects.

10.3.1 Motivations

Why should we want to build large knowledge bases at all? There are many reasons, among them:

- Brittleness—Specialized knowledge-based systems are brittle. They cannot cope with novel situations, and their performance degradation is not graceful. Programs built on top of deep, commonsense knowledge about the world should rest on firmer foundations.

- Form and Content—The techniques we have seen so far for representing and using knowledge may or may not be sufficient for the purposes of AI. One good way to find out is to start coding large amounts of commonsense knowledge and see where the difficulties crop up. In other words, one strategy is to focus temporarily on the content of knowledge bases rather than on their form.
- Shared Knowledge—Small knowledge-based systems must make simplifying assumptions about how to represent things like space, time, motion, and structure. If these things can be represented once at a very high level, then domain-specific systems can gain leverage cheaply. Also, systems that share the same primitives can communicate easily with one another.

Building an immense knowledge base is a staggering task, however. We should ask whether there are any methods for acquiring this knowledge automatically. Here are two possibilities:

1. Machine Learning—In Chapter 17, we discuss some techniques for automated learning. However, current techniques permit only modest extensions of a program's knowledge. In order for a system to learn a great deal, it must already know a great deal. In particular, systems with a lot of knowledge will be able to employ powerful analogical reasoning.
2. Natural Language Understanding—Humans extend their own knowledge by reading books and talking with other humans. Since we now have on-line versions of encyclopedias and dictionaries, why not feed these texts into an AI program and have it assimilate all the information automatically? Although there are many techniques for building language understanding systems (see Chapter 15), these methods are themselves very knowledge-intensive. For example, when we hear the sentence

   John went to the bank and withdrew $50.

   we easily decide that "bank" means a financial institution, and not a river bank. To do this, we apply fairly deep knowledge about what a financial institution is, what it means to withdraw money, etc. Unfortunately, for a program to assimilate the knowledge contained in an encyclopedia, that program must already know quite a bit about the world.

The approach taken by CYC is to hand-code (what its designers consider to be) the ten million or so facts that make up commonsense knowledge. It may then be possible to bootstrap into more automatic methods.

10.3.2 CYCL

CYC's knowledge is encoded in a representation language called CYCL. CYCL is a frame-based system that incorporates most of the techniques described in Chapter 9 (multiple inheritance, slots as full-fledged objects, *transfers-through*, *mutually-disjoint-with*, etc). CYCL generalizes the notion of inheritance so that properties can be inherited along any link, not just *isa* and *instance*. Consider the two statements: