2 Semantic Nets and Description Matching

In this chapter, you learn about the role of *representation* in artificial intelligence, and you learn about *semantic nets*, one of the most ubiquitous representations used in artificial intelligence. You also learn about *describe and match*, an important problem-solving method.

By way of illustration, you see how one describe-and-match program, working on semantic-net descriptions, can solve geometric analogy problems of the sort found on intelligence tests. You also see how another describe-and-match program, again working on semantic-net descriptions, can recognize instances of abstractions, such as "mixed blessing" and "retaliation," in semantic nets that capture story plots. The pièce de résistance involves the analysis of O. Henry's intricate short story, "The Gift of the Magi." Both the analogy program and the abstraction program show that simple descriptions, conforming to appropriate representations, can lead to easy problem solving.

Also, you see that the describe-and-match method is effective with other representations, not just with semantic nets. In particular, you see how the describe-and-match method lies underneath the *feature-based approach* to object identification.

Once you have finished this chapter, you will know how to evaluate representations and you will know what representation-oriented questions you should always ask when you are learning how to deal with an unfamiliar class of problems. You will have started your own personal collection of representations and problem-solving methods by learning about semantic nets, feature spaces, and the describe-and-match method. Finally, you will have started your own personal collection of case studies that will serve as useful precedents when you are confronted with new problems.

SEMANTIC NETS

In this section, you learn about what a representation is, a sense in which most representations are equivalent, and criteria by which representations can be judged. You also learn about semantic nets, a representation that sees both direct and indirect service throughout artificial intelligence.

Good Representations Are the Key to Good Problem Solving

In general, a **representation** is a set of conventions about how to describe a class of things. A **description** makes use of the conventions of a representation to describe some particular thing.

Finding the appropriate representation is a major part of problem solving. Consider, for example, the following children's puzzle:

The Farmer, Fox, Goose, and Grain ———————

A farmer wants to move himself, a silver fox, a fat goose, and some tasty grain across a river. Unfortunately, his boat is so tiny he can take only one of his possessions across on any trip. Worse yet, an unattended fox will eat a goose, and an unattended goose will eat grain, so the farmer must not leave the fox alone with the goose or the goose alone with the grain. What is he to do?

Described in English, the problem takes a few minutes to solve because you have to separate important constraints from irrelevant details. English is not a good representation.

Described more appropriately, however, the problem takes no time at all, for everyone can draw a line from the start to the finish in figure 2.1 instantly. Yet drawing that line solves the problem because each boxed picture denotes a safe arrangement of the farmer and his possessions on the banks of the river, and each connection between pictures denotes a legal crossing. The drawing is a good description because the allowed situations and legal crossings are clearly defined and there are no irrelevant details.

To make such a diagram, you first construct a **node** for each way the farmer and his three possessions can populate the two banks of the river. Because the farmer and his three possessions each can be on either of the two river banks, there are $2^{1+3} = 16$ arrangements, 10 of which are safe in the sense that nothing is eaten. The six unsafe arrangements place an animal and something the animal likes to eat on one side, with the farmer on the other.

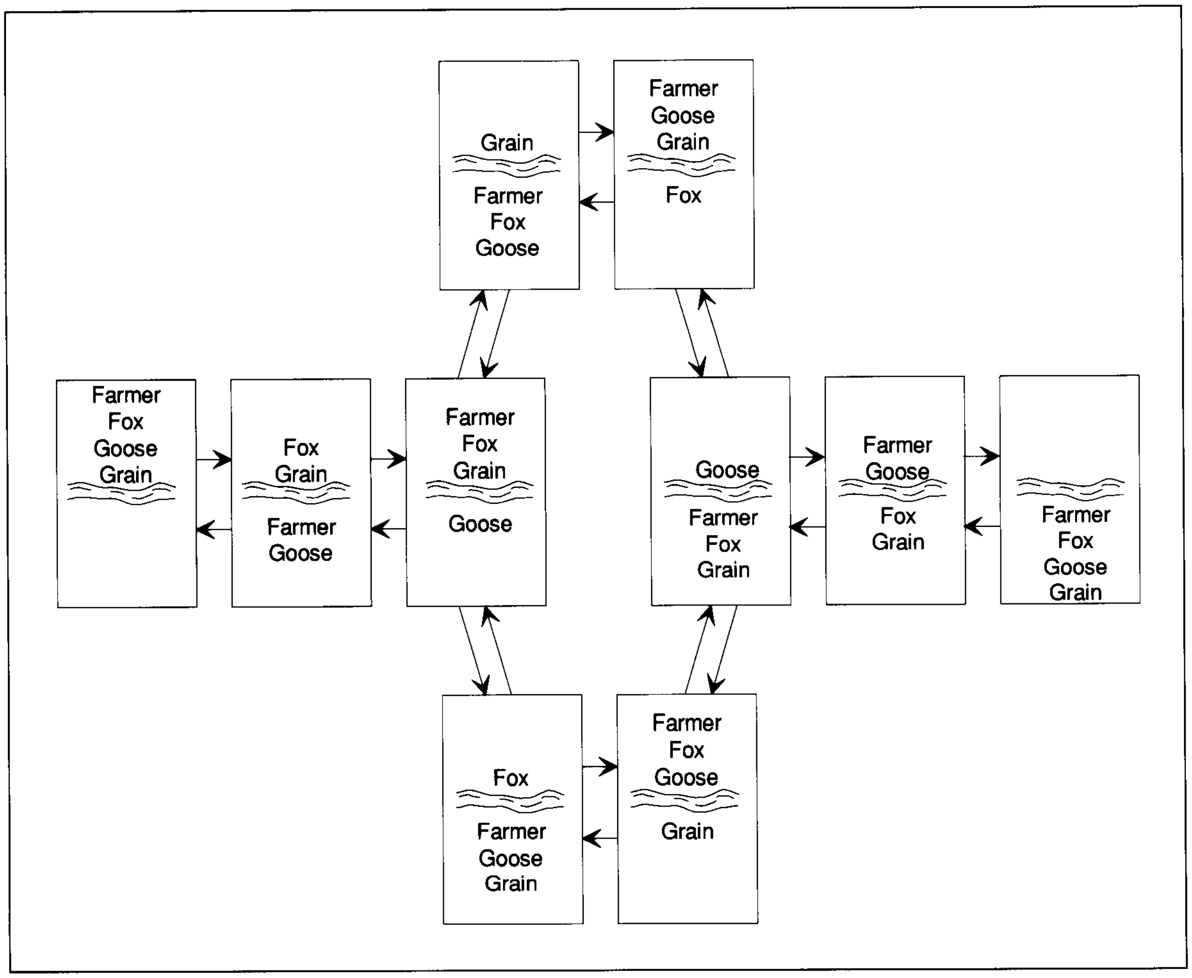


Figure 2.1 The problem of the farmer, fox, goose, and grain. The farmer must get his fox, goose, and grain across the river, from the arrangement on the left to the arrangement on the right. His boat will hold only him and one of his three possessions.

The second and final step is to draw a **link** for each allowable boat trip. For each ordered pair of arrangements, there is a connecting link if and only if the two arrangements meet two conditions: first, the farmer changes sides; and second, at most one of the farmer's possessions changes sides. Because there are 10 safe arrangements, there are $10 \times 9 = 90$ ordered pairs, but only 20 of these pairs satisfy the conditions required for links.

Evidently, the node-and-link description is a good description with respect to the problem posed, for it is easy to make, and, once you have it, the problem is simple to solve.

The important idea illustrated by the farmer, fox, goose, and grain problem is that a good description, developed within the conventions of a good representation, is an open door to problem solving; a bad description, using a bad representation, is a brick wall preventing problem solving.

In this book, the most important ideas—such as the idea that a good representation is important—are called **powerful ideas**; they are highlighted thus:

The **representation principle**:

▷ Once a problem is described using an appropriate representation, the problem is almost solved.

In the rest of this book, you learn about one or two powerful ideas per chapter.

Good Representations Support Explicit, Constraint-Exposing Description

One reason that the node-and-link representation works well with the farmer, fox, goose, and grain problem is that *it makes the important objects and relations explicit.* There is no bothering with the color of the fox or the size of the goose or the quality of the grain; instead, there is an explicit statement about safe arrangements and possible transitions between arrangements.

The representation also is good because *it exposes the natural constraints inherent in the problem.* Some transitions are possible; others are impossible. The representation makes it easy to decide which is true for any particular case: a transition is possible if there is a link; otherwise, it is impossible.

You should always look for such desiderata when you evaluate representations. Here is a list with which you can start, beginning with the two ideas just introduced:

- Good representations make the important objects and relations explicit: You can see what is going on at a glance.
- They expose natural constraints: You can express the way one object or relation influences another.
- They bring objects and relations together: You can see all you need to see at one time, as if through a straw.
- They suppress irrelevant detail: You can keep rarely used details out of sight, but still get to them when necessary.
- They are transparent: You can understand what is being said.
- They are complete: You can say all that needs to be said.
- They are concise: You can say what you need to say efficiently.
- They are fast: You can store and retrieve information rapidly.
- They are computable: You can create them with an existing procedure.

A Representation Has Four Fundamental Parts

With the farmer, fox, goose, and grain problem as a point of reference, you can now appreciate a more specific definition of what a representation is. A **representation** consists of the following four fundamental parts:

- A **lexical** part that determines which symbols are allowed in the representation's **vocabulary**
- A **structural** part that describes **constraints** on how the symbols can be arranged
- A **procedural** part that specifies **access procedures** that enable you to create descriptions, to modify them, and to answer questions using them
- A **semantic** part that establishes a way of associating **meaning** with the descriptions

In the representation used to solve the farmer, fox, goose, and grain problem, the **lexical** part of the representation determines that nodes and links are involved. The **structural** part specifies that links connect node pairs. The **semantic** part establishes that nodes correspond to arrangements of the farmer and his possessions and links correspond to river traversals. And, as long as you are to solve the problem using a drawing, the **procedural** part is left vague because the access procedures are somewhere in your brain, which provides constructors that guide your pencil and readers that interpret what you see.

Semantic Nets Convey Meaning

The representation involved in the farmer problem is an example of a **semantic net**.

From the lexical perspective, semantic nets consist of **nodes**, denoting objects, **links**, denoting relations between objects, and **link labels** that denote particular relations.

From the structural perspective, nodes are connected to each other by labeled links. In diagrams, nodes often appear as circles, ellipses, or rectangles, and links appear as arrows pointing from one node, the **tail node**, to another node, the **head node**.

From the semantic perspective, the meaning of nodes and links depends on the application.

From the procedural perspective, access procedures are, in general, any one of **constructor procedures**, **reader procedures**, **writer procedures**, or possibly **erasure procedures**. Semantic nets use constructors to make nodes and links, readers to answer questions about nodes and links, writers to alter nodes and links, and, occasionally, erasers to delete nodes and links.

By way of summary, the following specifies what it means to be a semantic net in lexical, structural, semantic, and procedural terms, using an informal specification format that appears throughout the rest of this book:

A **semantic net** is a representation

In which

▷ Lexically, there are nodes, links, and application-specific link labels.

▷ Structurally, each link connects a tail node to a head node.

▷ Semantically, the nodes and links denote application-specific entities.

With constructors that

▷ Construct a node

▷ Construct a link, given a link label and two nodes to be connected

With readers that

▷ Produce a list of all links departing from a given node

▷ Produce a list of all links arriving at a given node

▷ Produce a tail node, given a link

▷ Produce a head node, given a link

▷ Produce a link label, given a link

Such specifications are meant to be a little more precise and consistent than ordinary English phrases, but not stuffily so. In particular, they are not so precise as to constitute a specification of the sort you would find in an official standard for, say, a programming language.

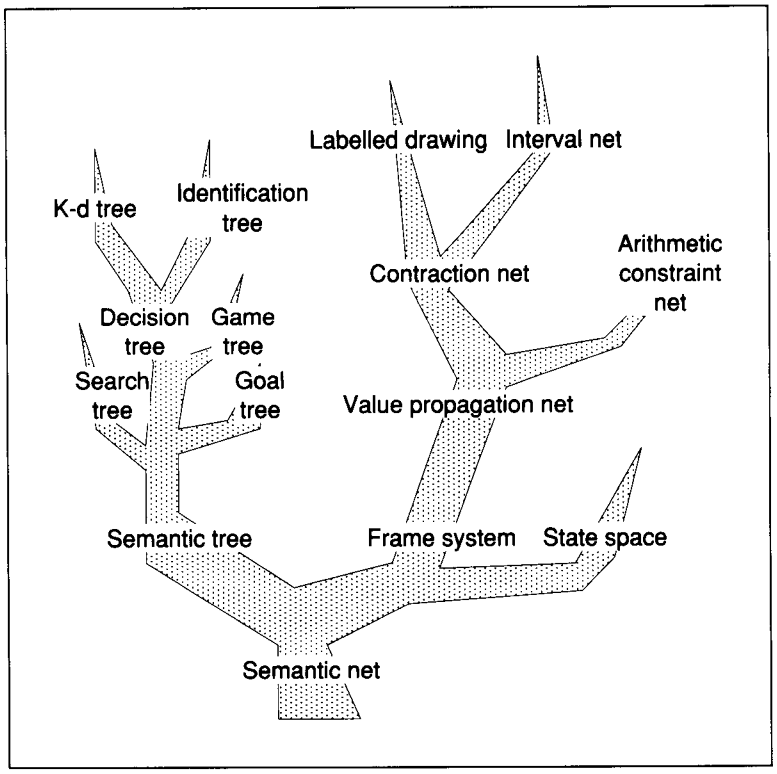
Nevertheless, the specifications are sufficiently precise to show that many of the key representations in artificial intelligence form family groups. Figure 2.2, for example, shows part of the family of representations for which the semantic-net representation is the ultimate ancestor. Although this semantic-net family is large and is used ubiquitously, you should note that it is but one of many that have been borrowed, invented, or reinvented in the service of artificial intelligence.

There Are Many Schools of Thought About the Meaning of Semantics

Arguments about what it means to have a semantics have employed philosophers for millennia. The following are among the alternatives advanced by one school or another:

- **Equivalence semantics.** Let there be some way of relating descriptions in the representation to descriptions in some other representation that already has an accepted semantics.

Figure 2.2 Part of the semantic-net family of representations. Although many programs explained in this book use one of the family members shown, others use important representations that lie outside of the family.



- **Procedural semantics.** Let there be a set of programs that operate on descriptions in the representation. Say that meaning is defined by what the programs do.
- **Descriptive semantics.** Let there be explanations of what descriptions mean in terms we understand intuitively.

From the perspective of descriptive semantics, the net on the left side of figure 2.3 is not a semantic net, because there is neither a prima facie description in terms you understand nor an explanation of what the link labels mean in terms you understand. The net on the right side of figure 2.3, however, is a semantic net, because you naturally tend to ascribe meaning to the links. Asked what the net means, most people would say immediately that it means that an object, known as the lintel, is supported by two other objects, known as posts.

Of course, the objects and relations involved in semantic nets need not be so concrete. The representation used in the farmer illustration is a semantic net because particular arrangements of the farmer and his possessions can be viewed as abstract objects, thereby meriting node status, and allowed river crossings can be viewed as abstract relations, thereby meriting link status.

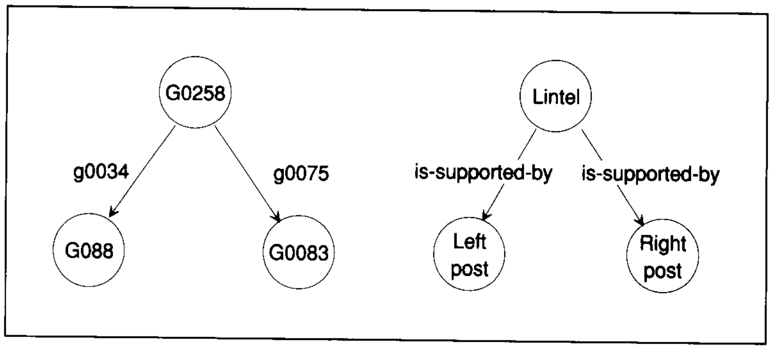


Figure 2.3 An ordinary net (left) and a semantic net (right). Natural-language labels associate intuitive meanings with nodes and links, thereby producing an informal semantics.

Ultimately, both equivalence semantics and procedural semantics lead back to descriptive semantics. In the case of equivalence semantics, descriptions have meaning because they are equivalent to something that means something to you. In the case of procedural semantics, descriptions have meaning because they cause a program to exhibit a behavior that means something to you. Thus, the alternatives all seem rooted in perceptions to which you ascribe meaning intuitively.

Theoretical Equivalence Is Different from Practical Equivalence

In some uninteresting theoretical sense, any computer-based representation can do anything that any other can do, because computer-based representations are based, ultimately, on arrangements of bits in memory. Consequently, any representation that can be used to represent arrangements of bits can be used as a substratum for the construction of any other representation.

In a practical sense, however, some representations help you to focus on the objects and relations you need to solve a class of problems. One representation, therefore, is more powerful than another because it offers you more convenience even though, theoretically, both can do the same work. *Convenience*, however, is perhaps too weak a word. In general, the good qualities of powerful representations make practicable what would be impracticable with weak representations.

THE DESCRIBE-AND-MATCH METHOD

In this section, you learn about the describe-and-match method; by way of illustration, you learn how the describe-and-match method can be used to identify two-dimensional objects.

As illustrated in figure 2.4 the basic idea behind the **describe-and-match method** is that you can identify an object by first describing it and then searching for a matching description in a description library. The objects involved may be simple physical entities such as the blocks with

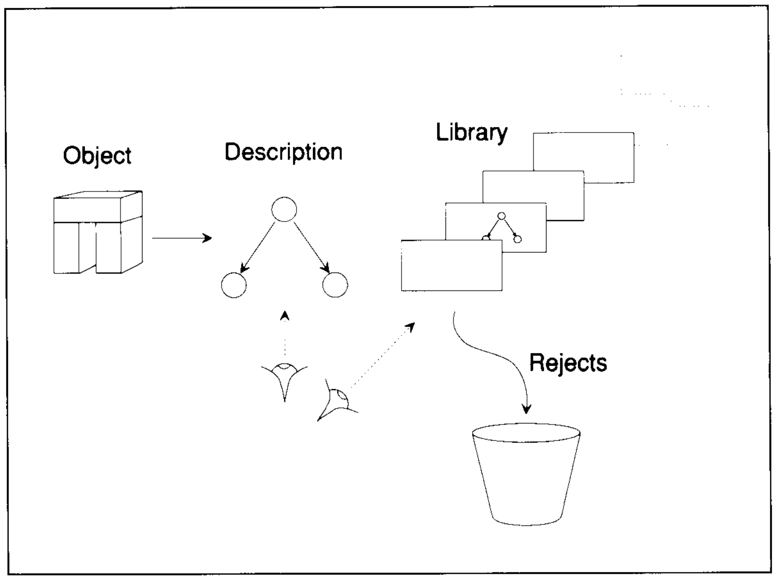


Figure 2.4 The describe-and-match paradigm. To identify an object, you describe it, and then you look for a matching description in a description library.

which children play, or complicated abstractions, such as those that emerge in the forthcoming examples.

As you move through this book, you will see many methods, such as the describe-and-match method, reduced to a specification cast in a form that is more precise than ordinary English, yet more transparent than a programming language—particularly a programming language that you do not happen to know. In this book, this informal, half-English, half-program form is called **procedural English**. Here is the describe-and-match method expressed in procedural English:

```
To identify an object using describe and match,
▷ Describe the object using a suitable representation.
▷ Match the object description against library descriptions
  until there is a satisfactory match or there are no more
  library descriptions.
▷ If you find a satisfactory match, announce it; otherwise,
  announce failure.
```

In general, procedural English allows all these programming constructs:

- Steps and substeps, denoted by indentation, much after the fashion of an outline
- Iterations, denoted by words such as *until* and *for each*
- Conditional actions, denoted by words such as *if* and *otherwise*
- Various sorts of tests, denoted variously

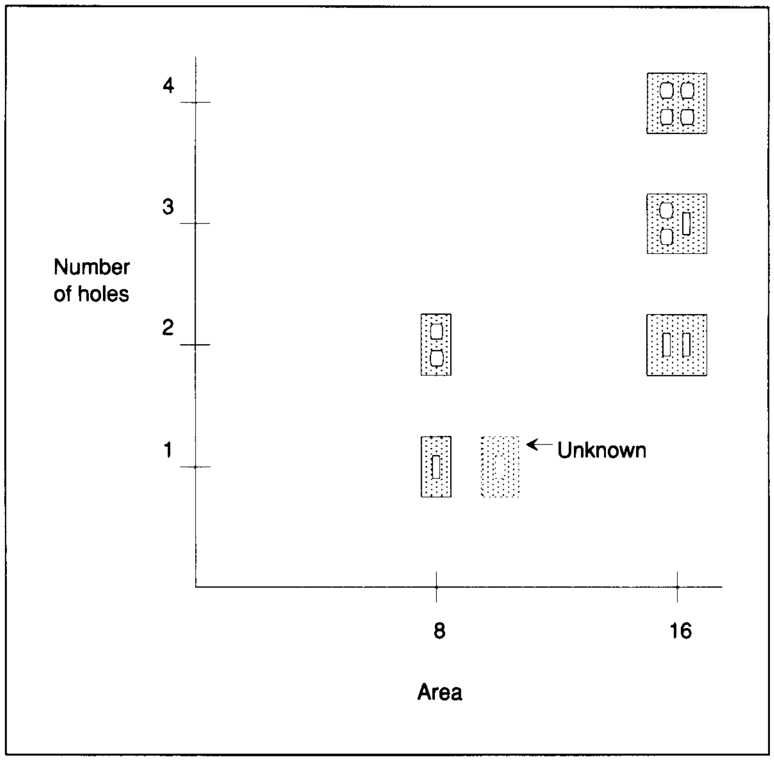


Figure 2.5 A feature space. An unknown object is identified according to the distances between its feature point and those of various models. Evidently the unknown is most likely to be a single-hole switch plate.

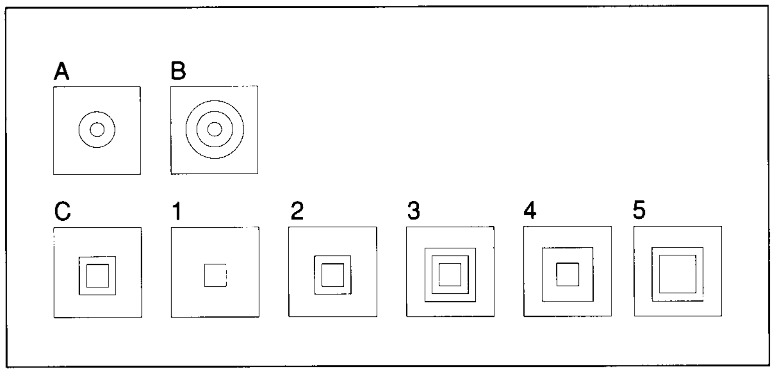
Many good programmers use a notation much like procedural English at the design stage, when they are deciding what a procedure will do. Much of the procedural English then survives in the form of illuminating comments.

Feature-Based Object Identification Illustrates Describe and Match

Feature-based object identification is one of the simplest applications of the describe-and-match method. Feature-based object identifiers consist of a **feature extractor** and a **feature evaluator**. The feature extractor measures simple characteristics such as an object's area. Values obtained by the feature extractor become the coordinates of a **feature point** in **feature space**, a multidimensional space in which there is one dimension for each feature measured. To identify an unknown object, you compare the distances between its feature point and the feature points of various idealized objects. The most likely identity of the unknown object is determined by the smallest distance. Figure 2.5 shows the points corresponding to an unknown object and a family of idealized electrical-box covers in a box-cover feature space.

Generally, speed and discrimination considerations determine which features are used in particular situations. Candidate features for objects

Figure 2.6 An easy problem for the ANALOGY program.



such as electrical-box covers include total object area, hole area, hole count, perimeter length, minimum distance from center of area to edge, maximum distance from center of area to edge, average distance from center of area to edge, length of major axis of ellipse of equal inertia moment, length of minor axis of ellipse of equal inertia moment, total area minus hole area, ratio of hole area to total area, and ratio of perimeter squared to area.

THE DESCRIBE-AND-MATCH METHOD AND ANALOGY PROBLEMS

In this section, you learn that the describe-and-match method, working in harness with a semantic-net representation, produces impressive performance on geometric analogy problems in which the problem, as shown in figure 2.6, is to select an answer figure, X, such that A is to B as C is to X gives the best fit.

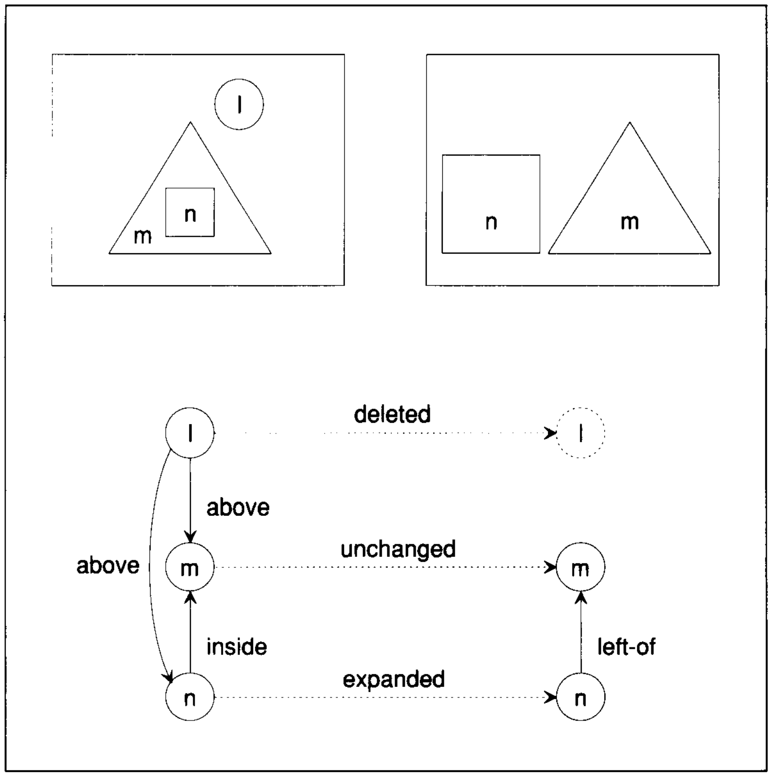
One way to start is to *describe* rules that explain how A becomes B and how C becomes each of the answer figures. Then, you can *match* the rule that explains how A becomes B to each rule that explains how C becomes an answer. The best match between rules identifies the best answer. Thus, the describe-and-match paradigm can be used to solve analogy problems.

The key to solving such problems lies in good rule descriptions. The ANALOGY program, described in this section, does its job by matching rule descriptions together and measuring description similarity.

Geometric Analogy Rules Describe Object Relations and Object Transformations

ANALOGY uses two-part rules. One rule part describes how the objects are arranged in the source and destination figures. One object may be above, to the left of, or inside of another. The other rule part describes how the objects in the source figure are transformed into objects in the destination figure. An object may be scaled, rotated, or reflected, or may be subject

Figure 2.7 A rule described as a geometric analogy net, which is a kind of semantic net. Rule descriptions consist of object-relation descriptions and object-transformation descriptions. Links shown solid describe relations among source objects and among destination objects. Links shown dotted describe how objects are transformed between the source and the destination.



to some combination of these operations. Also, an object may be added or deleted.

A typical rule can be described using a semantic-net representation, as illustrated in figure 2.7. This representation is not just any semantic net, of course—it is one that is specialized to describe rules:

A **geometric analogy net** is a representation

That is a semantic net

In which

▷ The nodes denote dots, circles, triangles, squares, rectangles, and other geometric objects.

▷ Some links denote relations among figures objects, specifically inside, above, and to the left of.

▷ Other links describe how figure objects are transformed. The possibilities are addition, deletion, expansion, contraction, rotation, and reflection, and combinations of these operations.

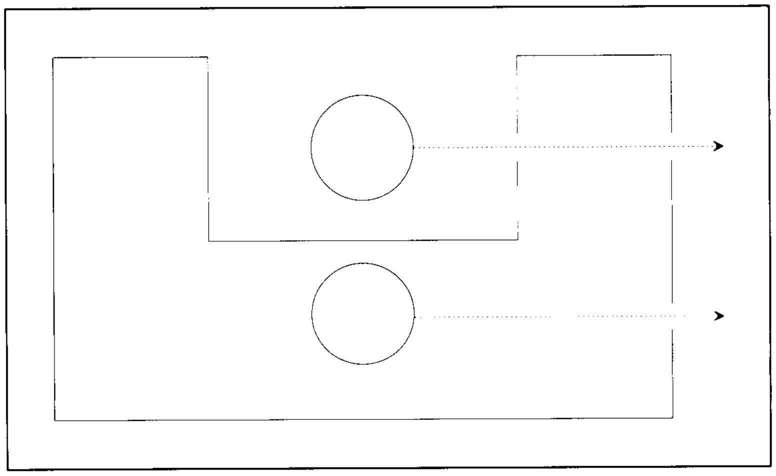


Figure 2.8 Two circles, one of which is inside a polygon. One object is inside another if a line drawn to infinity crosses the boundary of the potentially surrounding object an odd number of times. Thus, one circle is inside; the other is not.

You could write this specification for geometric analogy nets, of course, without any reference to semantic nets, by importing all the descriptive elements from the semantic-net specification. The alternative shown is better, not only because it saves space, but also because it focuses on exactly what you need to add to transform the general concept into a representation tailored to a particular circumstance. As you can see, transforming the semantic-net concept into a geometric analogy net requires only the application-specific recitation of which link labels are allowed and what the nodes and links denote.

ANALOGY uses a simple idea, illustrated in figure 2.8, to decide whether Inside, rather than either Left-of or Above, is the appropriate relation between objects. First, ANALOGY makes sure that the objects do not touch. Then, ANALOGY constructs a line from any point on one figure to infinity, as shown in figure 2.8. If the line crosses the second figure an odd number of times, then the second figure surrounds the first. Happily, this method involves only simple line-crossing tests, and it works even if the figures are extremely convoluted.

As shown in figure 2.9, ANALOGY uses another simple procedure to compute the spatial relationship between two objects. ANALOGY computes the center of area of each of the two objects, constructs diagonal lines through the center of area of one of them, and notes which region contains the center of area of the other object. Because the relations used are symmetric, it is not necessary to note both left and right relations.

Finally, ANALOGY uses a matching procedure to decide if an object in one figure can be transformed into an object in another figure by a combination of scaling, rotation, and reflection operations. The dotted links in figure 2.7 mark objects that pass this transformation test.

Now that you have seen how rules are constructed, you can see that the example in figure 2.10 is contrived so as to depend on only relations

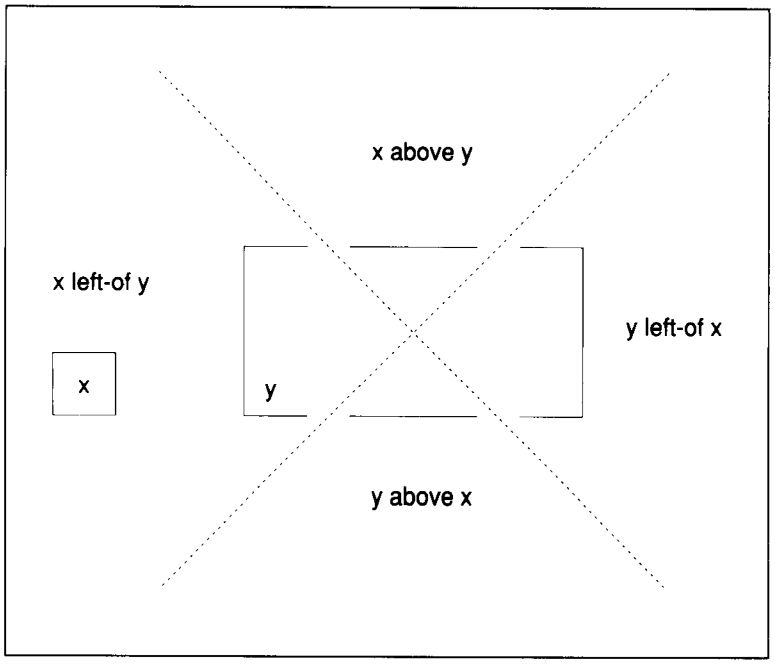


Figure 2.9 A square to the left of a rectangle. Relations between objects are determined by comparing centers of area.

between objects. No object transformations can influence the solution, because no objects are transformed in the move from the source figure to the destination figure.

It is clear that the C-to-3 rule best matches the A-to-B rule because, with l associated with x and m associated with y, the two rules match exactly.

Note, however, that there is no a priori reason to associate l with x rather than with y. In going from the source figure to the destination figure, you want to be sure that squares go to squares, circles to circles, triangles to triangles, and so on. But this need to match one object to a geometrically similar object does not hold in comparing two rules. In the example, answer 3 is to be selected even though the objects in A and B are a triangle and a square, whereas in C and in all the answer figures, the objects are a circle and a dot. In general, ANALOGY must try all possible ways of associating the nodes when matching rules.

This one-for-one association of variables implies that the number of objects that move from the source figure to the destination figure must be the same in both of the two rules. The number of additions and deletions must be the same as well. Any attempt to match two rules for which the numbers are different fails immediately.

If n objects move from the source figure to the destination figure in each of two rules being compared, there will be $n!$ ways of associating the variables in searching for the best way to match the rules. More generally,

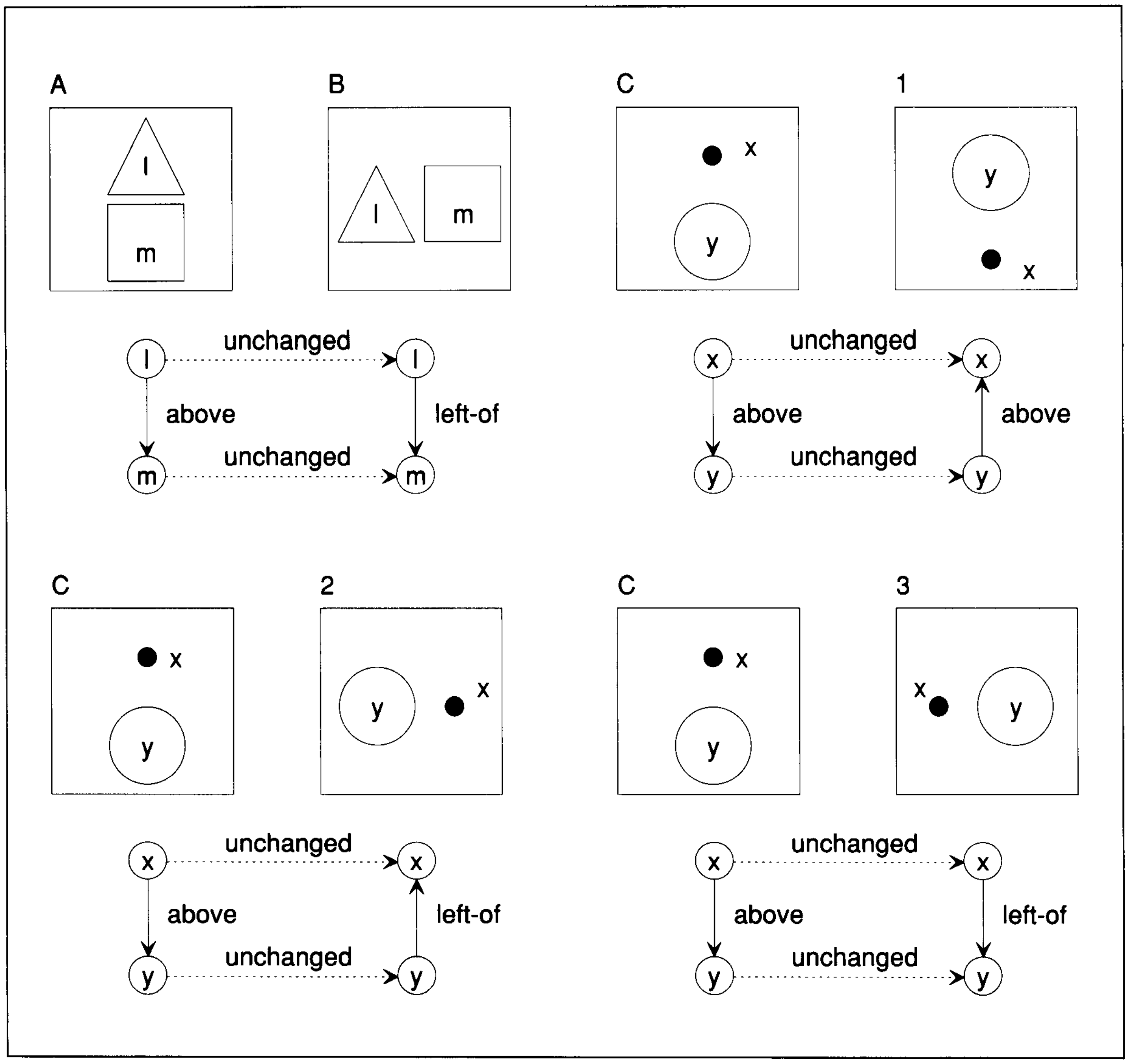


Figure 2.10 A problem whose solution is determined by relations only. Comparison of the rule descriptions verifies that the C-to-3 rule matches best.

if n_1 objects move, n_2 are added, and n_3 deleted, in going to the destination figure, then $n_1!\,n_2!\,n_3!$ is the number of possible associations. All must be tried.

In the example, there are two possible ways to associate the objects, because $n_1 = 2$, $n_2 = 0$, and $n_3 = 0$. Specifically, ANALOGY can associate l with x and m with y, or ANALOGY can associate l with y and m with x.

The previous example involves only relations between objects, because no objects are transformed. Symmetrically, for the problem in figure 2.11, there is only one object in each figure, so there are no relations between

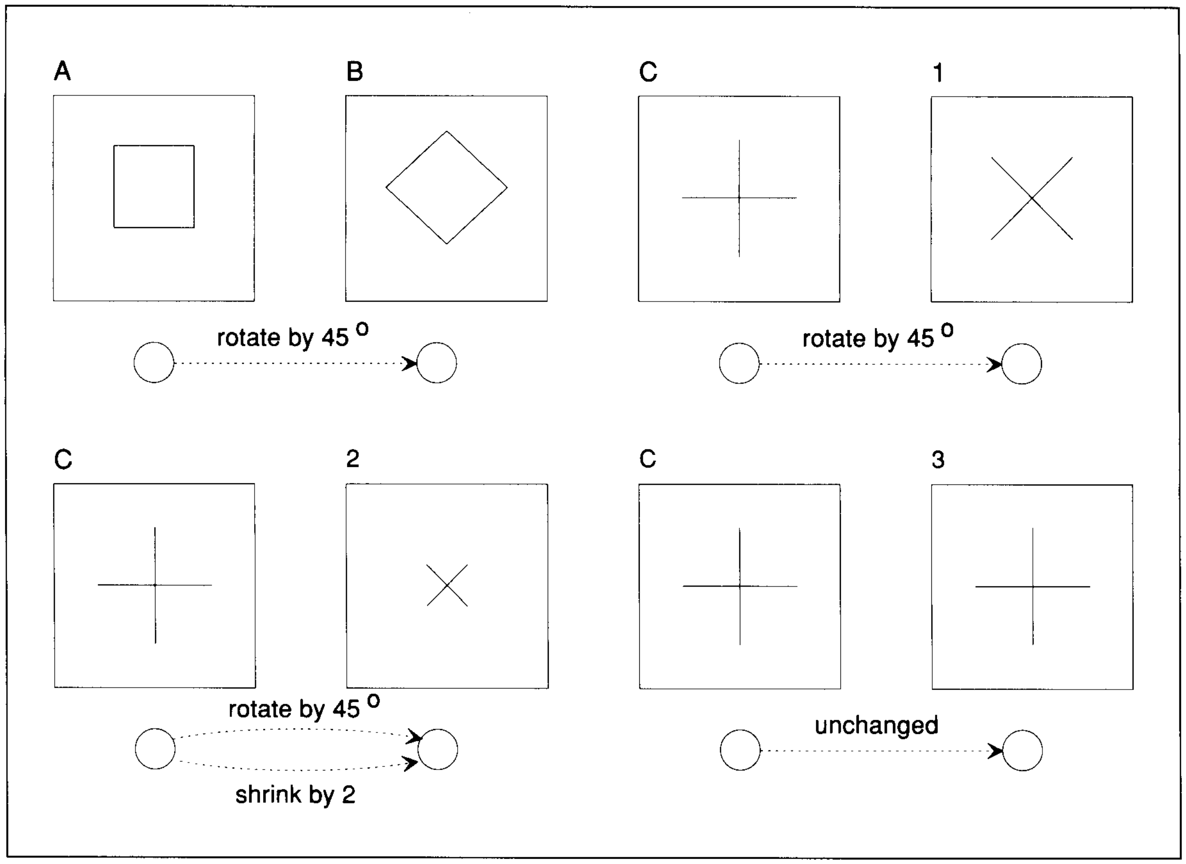


Figure 2.11 A problem whose solution is determined by transformations only. Because each figure has only one object, relations between objects are not relevant. Comparison of the rule descriptions verifies that the C-to-1 rule provides is the best match.

objects, and only object transformations can matter. ANALOGY concludes that the C-to-1 rule best matches the A-to-B rule, because only answer 1 corresponds to a simple 45° rotation with no reflection or scale change.

Scoring Mechanisms Rank Answers

How should ANALOGY measure the similarity of two rules? So far, the examples have been so simple that the best answer rule matches the A-to-B rule exactly. But if an exact match cannot be found, then ANALOGY must rank the inexact matches. One way to do this ranking is to count the number of matching elements in the two rules involved in each match, as shown in figure 2.12.

To tune the counting a bit, you can weight relations describing object transformations less heavily than you weight relations describing relations among objects. Assuming that relations among objects each add one point to the total score, then less than one point should be added for each object-transformation relation. Experimentally, the numbers shown in figure 2.13 work well. A radically different set of numbers would reflect

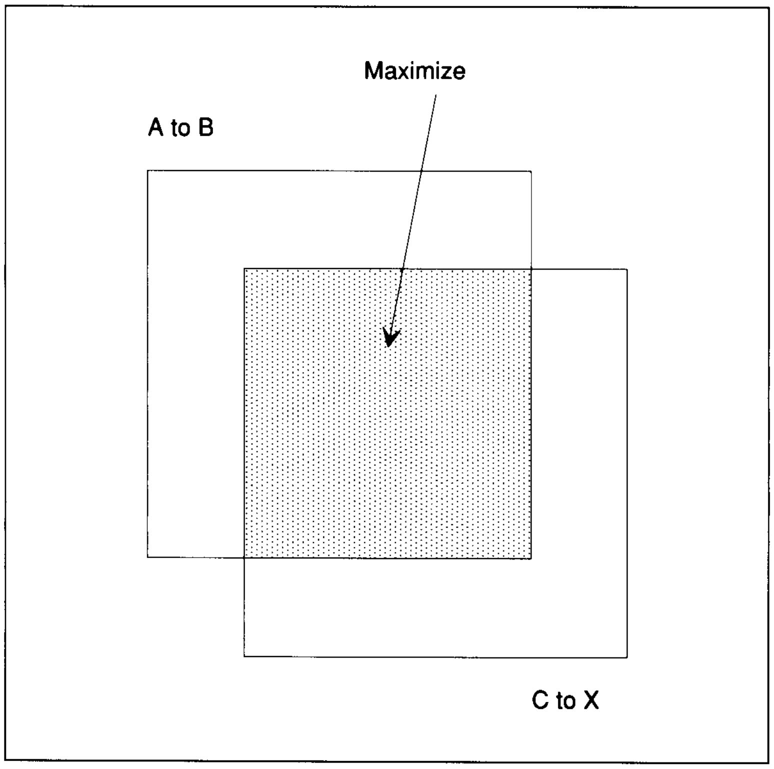


Figure 2.12 Rule similarity, measured by degree of overlap. You determine the answers by finding the C-to-X rule with the maximum number of elements in common with the A-to-B rule.

a different judgment about how the various possibilities should be ordered. The given set is biased toward rotations and against reflections. A different set might indicate the opposite preference. The corresponding variations on ANALOGY would occasionally disagree with one another about the answers.

Of course, it is possible to elaborate the measure of similarity in other directions. Suppose, for example, that $\mathcal{S}_{\text{AB}}$ is the set of elements in the A-to-B rule, and that $\mathcal{S}_{\text{CX}}$ is the set of elements in the C-to-X rule. Then, $\mathcal{S}_{\text{AB}} \cap \mathcal{S}_{\text{CX}}$ is the set of elements that appear in both rules, $\mathcal{S}_{\text{AB}} - \mathcal{S}_{\text{CX}}$ is the set of elements appearing in the A-to-B rule but not in the C-to-X rule, and $\mathcal{S}_{\text{CX}} - \mathcal{S}_{\text{AB}}$ is the set of elements appearing in only the C-to-X rule. With these sets in hand, you can use the following formula to measure similarity:

$$\begin{aligned}\text{Similarity} =& \alpha \times \text{Size}(\mathcal{S}_{\text{AB}} \cap \mathcal{S}_{\text{CX}})\\ &- \beta \times \text{Size}(\mathcal{S}_{\text{AB}} - \mathcal{S}_{\text{CX}})\\ &- \gamma \times \text{Size}(\mathcal{S}_{\text{CX}} - \mathcal{S}_{\text{AB}})\end{aligned}$$

where α, β, and γ are weights, and Size is the function that computes the number of elements in a set. If $\beta = 0$, $\gamma = 0$, and $\alpha = 1$, the formula reduces to counting the common elements. If β and γ are not the same, the formula gives asymmetric similarity judgments, allowing, for example,

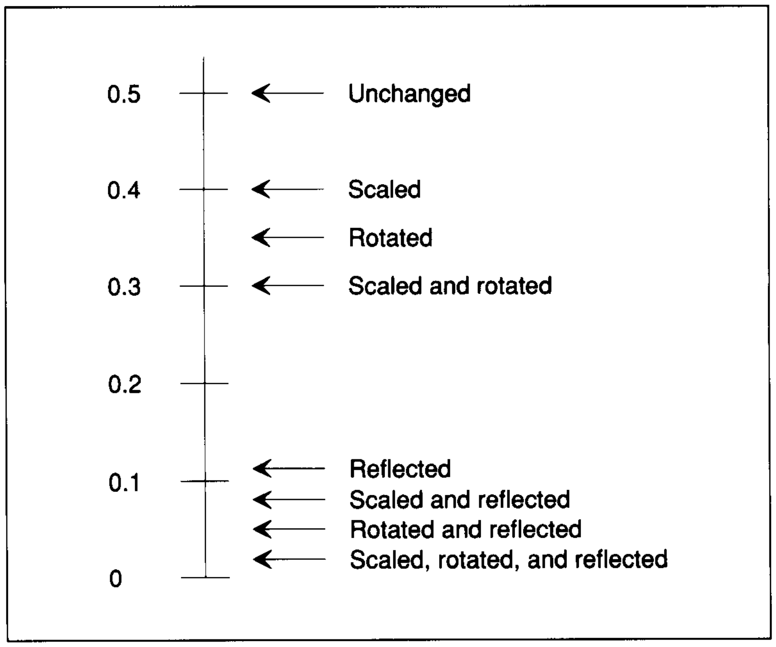


Figure 2.13 Weights determine transformation-description contributions to similarity scores.

the A-to-B rule to be more similar to the C-to-X rule than the C-to-X rule is to the A-to-B rule.

Be skeptical about such formulas, however. Viewed as a representation for importance, a set of weights is not explicit and exposes little constraint.

Ambiguity Complicates Matching

So far, all the objects in the source and destination figures have distinct shapes. Consequently, it is easy to decide how to form the rule describing the transformation. In situations such as the one shown in figure 2.14, however, there is ambiguity because there is no way to know which of the two triangles has disappeared. Perhaps the larger one is gone; alternatively the smaller one may have been deleted and the larger one may have shrunk.

In fact, you cannot judge either explanation to be superior without considering the other figures given in the problem. Consequently, ANALOGY must construct two rules, one corresponding to each way the triangles in the source figure can be identified with triangles in the destination. In general, for each source and destination pair, many rules are possible; and, for each rule, there may be many ways to match it against another rule.

Good Representation Supports Good Performance

Examine figure 2.15. It shows three examples, drawn from intelligence tests, that are well within the grasp of the ANALOGY procedure. In the first example, the most reasonable theory about the rule for going from A to B is that the inside object is deleted. The C-to-3 rule is the same, and

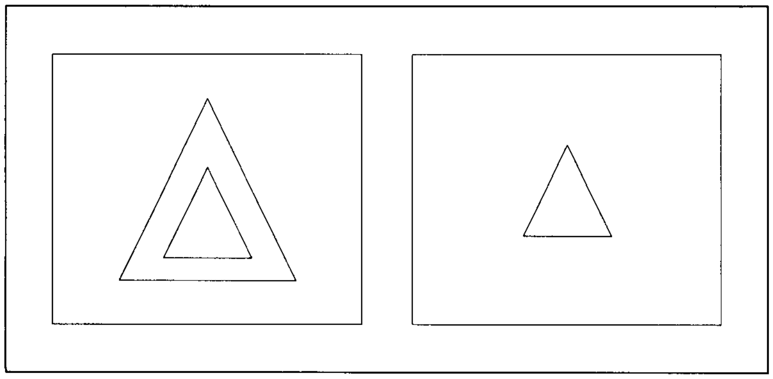


Figure 2.14 An ambiguous change. The large triangle may have been deleted; alternatively, the small one may have been deleted and the large one shrunk.

answer 3 is the best answer, with answer 4 a close second. Answer 4 would be the clear winner if answer 3 were not present.

In the second example, answer 3 is the correct answer. Actually, answer 3 is the only answer figure that ANALOGY considers seriously, because among the answer figures, only answer 3 has the same number of objects as B has. Remember that requiring the same number of objects is an indirect consequence of permitting a match only between rules for which the numbers of movements, additions, and deletions are the same.

In the third example, the A-to-B rule could be described as either a rotation or a reflection, with answer 2 being the best answer if the process prefers rotations, and with answer 1 being the best answer if it likes reflections better. ANALOGY prefers rotations, and judges answer 2 to be best.

THE DESCRIBE-AND-MATCH METHOD AND RECOGNITION OF ABSTRACTIONS

In this section, you learn that the describe-and-match method, again working in harness with a semantic-net representation, can be used to recognize abstractions in story plots. In combination with what you have already learned about the describe-and-match method and semantic nets, you see that both have a broad reach.

Story Plots Can Be Viewed as Combinations of Mental States and Events

To describe plots using a semantic net, you need a vocabulary of node types and link labels. Happily, you soon see that you can do a lot with a vocabulary of just three node types and three link labels. The three node types are mental states, denoted by MS in diagrams; positive events, denoted by +; and negative events, denoted by −. The link labels are *i*, an acronym for initiates, meaning that the mental state or event at the tail of an *i* link leads to the one at the head of the link; *t*, for terminates,

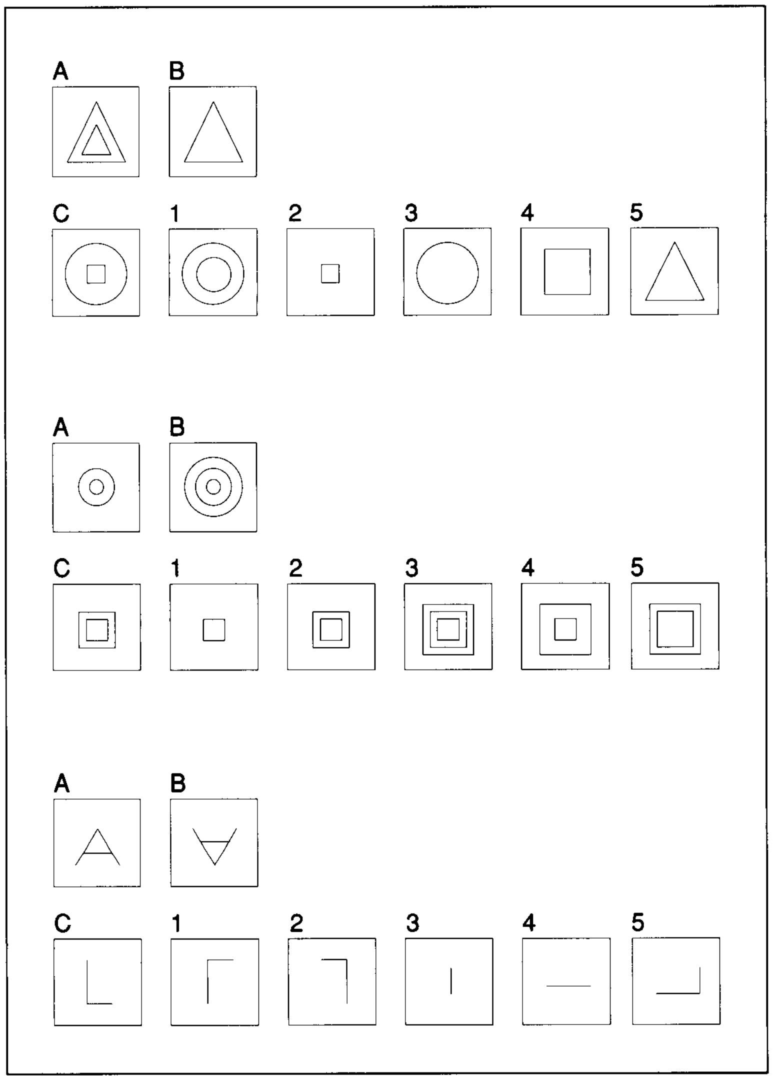
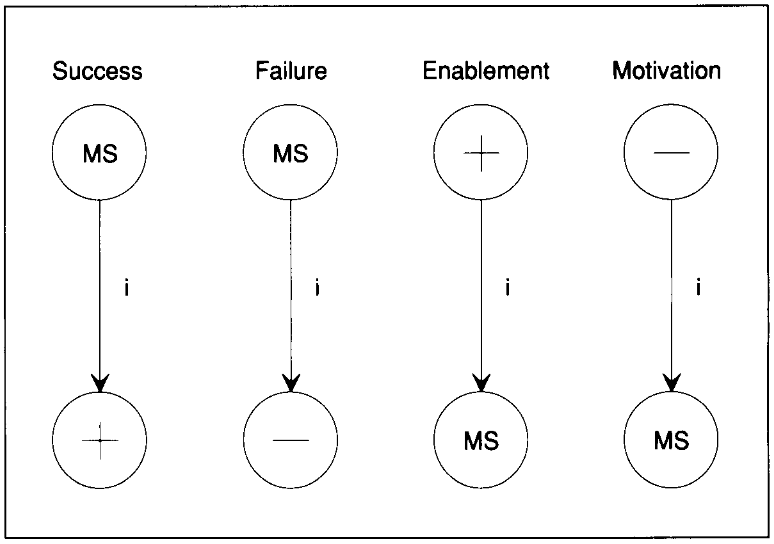


Figure 2.15 Three problems solved successfully by ANALOGY.

meaning that the mental state or event at the tail turns off the one at the head; and *c*, for corefers, meaning that the mental state or event at the tail refers to the same mental state or event as the one at the head. Links labeled with *c* have two heads: **double-headed links** are a notational shorthand for pairs of identically labeled single-headed links pointing in opposite directions.

With three node types and three link labels, there could be as many as 3 x 3 x 3 = 27 node–link–node combinations. Of these 27 possibilities, 15 have a natural, easily stated interpretation, and each of these is called a **base unit**. In figure 2.16, for example, four base units are exhibited,

Figure 2.16 Combinations in which mental states and events initiate one another. Mental states may initiate positive events or negative events and vice versa. The four possible combinations constitute instances of *success*, *failure*, *enablement*, and *motivation*, all of which are base units.



each of which involves a mental state that initiates an event, or vice versa. As shown, if a mental state initiates an event, you have what is casually called a *success* or a *failure*, depending on the sign of the event. If an event initiates a mental state, we witness *enablement* or *motivation*, again depending on the sign.

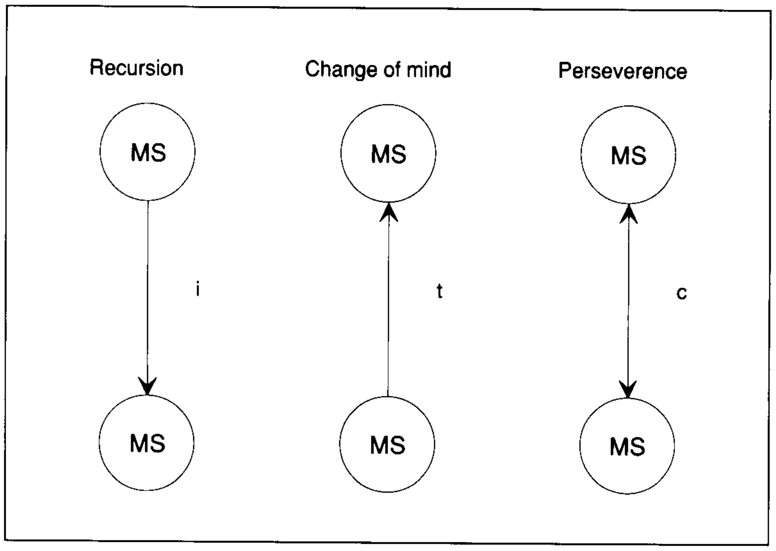
Another group of base units is shown in figure 2.17. This time, each of the base units involves two mental states. When one mental state initiates another, we say that *recursion* has occurred. When one terminates another, we have a *change of mind*. If a mental state persists over a period of time, the individual involved is exhibiting *perseverance*.

The final group of base units is shown in figure 2.18. Now there are no mental states at all; there are only events joined by termination or coreference. The eight combinations are *positive tradeoff* and *negative tradeoff*, *positive coreference* and *negative coreference*, *loss* and *resolution*, and *mixed blessing* and *hidden blessing*.

In descriptions, base units often overlap, producing recognizable aggregates. Let us call these aggregates **composite units**. Together, base units and composite units constitute **abstraction units**. Figure 2.19 shows a composite unit consisting of a *success* base unit joined, by its positive event, to a *loss* base unit. When a success is followed by a loss, in normal language we often say that "the success was fleeting," or use words to that effect. Hence, this composite unit is called *fleeting success*.

Other examples of composite units are shown in figure 2.20. Each composite unit in the figure consists of a negative event, followed by a mental state, followed by a positive event. The composite units differ because they involve different links, and, hence, different base units. Motivation followed by success yields *success born of adversity*; motivation followed by

Figure 2.17 Mental states joined by *initiate, terminate,* or *corefer* links. The three possible combinations constitute instances of *recursion, change of mind,* and *perseverance* base units.



a positive event that terminates the motivation-producing negative event is a matter of *fortuitous success*; and finally, motivation followed by a success involving a positive event that terminates the motivation-producing negative event is *intentional problem resolution.*

When more than one person is involved, more elaborate arrangements are possible. In figure 2.21, for example, the situation from one person's perspective is a *success born of adversity.* In addition, however, the negative event from that person's perspective corefers to a positive event from the other person's perspective. Similarly, the positive event corefers to a negative event. These additions, together with the *success born of adversity*, constitute *retaliation.*

Abstraction-Unit Nets Enable Summary

To recognize abstractions in a story plot, you first *describe* the story plots in terms of nodes, representing mental states and events, and links, representing relations among those mental states and events. Then you *match* the nodes and links with items in a catalog of named abstraction units. Consider, for example, the following story about Thomas and Albert:

Thomas and Albert

Thomas and Albert respected each other's technical judgment and decided to form a company together. Thomas learned that Albert was notoriously absentminded, whereupon he insisted that Albert have nothing to do with the proposed company's finances. This angered Albert so much that he backed out of their agreement, hoping that Thomas would be disappointed.

Figure 2.18 Positive events and negative events joined by terminate or coreference links. The possible combinations constitute instances of eight base units: *positive tradeoff, loss, resolution, negative tradeoff, positive coreference, mixed blessing, hidden blessing,* and *negative coreference.*

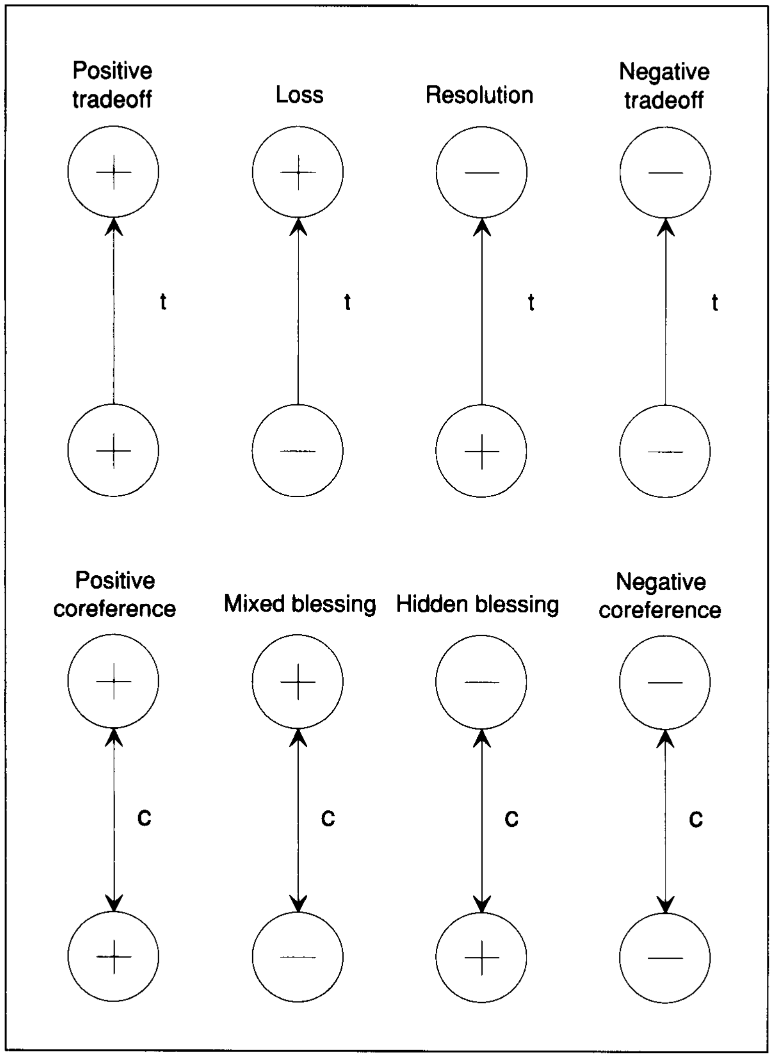
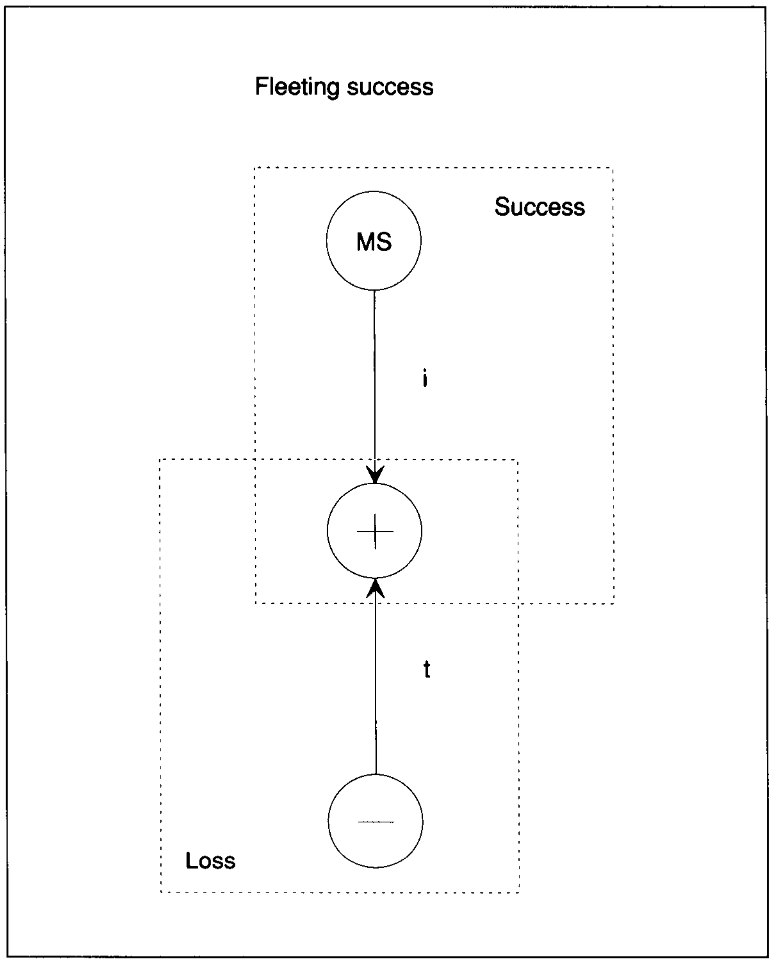


Figure 2.22 shows what "Thomas and Albert" looks like in semantic-net terms. The *respect* abstraction is captured by the two mental states at the top. Those mental states initiate the decision to form a company, a positive event from both Thomas's and Albert's points of view. Thomas's discovery about Albert is a negative event, which leads to a mental state in which Thomas thinks about the company's finances, which leads to his insistence that Albert keep out of them, a positive event as far as Thomas is concerned. The insistence is a negative event from Albert's perspective, however. For Albert, the insistence leads to a mental state that leads to backing out of the agreement, which Albert views now as a positive event and Thomas views as a negative one.

Figure 2.19 Base units joined to produce larger, composite units. In this illustration, a *success* unit and a *loss* unit, both basic, join to produce a *fleeting success* composite unit.

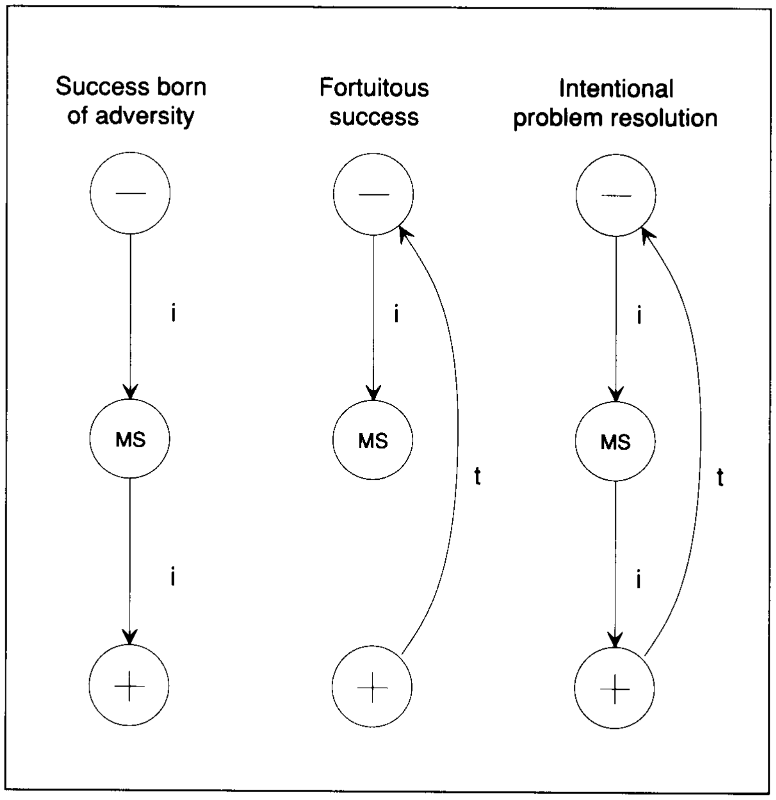


Now think of the diagram as a mine for abstraction units. Digging a little reveals that there are six abstraction units that are not wholly contained in some higher-level abstraction unit. These are called **top-level units**. In this particular example, the top-level units are connected to each other by exactly one shared mental state or one shared event.

Figure 2.23 shows the resulting arrangement of top-level units, in the form of a top-level abstraction net, with the top-level units shown in the same relative positions that their pieces occupied in figure 2.22.

To summarize a plot using a top-level abstraction net, you describe the central top-level unit first. Then, you describe the surrounding top-level units and explain how those top-level units are related to the central top-level unit. For the "Thomas and Albert" story, you would produce the following result:

Figure 2.20 Three different composite units. In this illustration, a negative event always is followed by a mental state that is followed by a positive event. In the first case, a *motivation* base unit is followed by a *success* base unit producing an instance of *success born of adversity*. In the second, the *success* unit disappears and a *resolution* unit appears, producing a *fortuitous success*. In the third, *success* reappears, joining the other two, producing an *intentional problem resolution*.



A Summary of Thomas and Albert

Albert *retaliated* against Thomas because Thomas went through an *intentional problem resolution* that was bad for Albert. The *retaliation* caused a *loss* for Thomas and a *positive tradeoff* for Albert. The *loss* reversed Thomas's previous *success*, and the *positive tradeoff* reversed Albert's previous *success*.

In addition to enabling summary, top-level abstraction nets allow you to compare and contrast two situations, even when those two situations are superficially quite different. Consider the following story about John and Mary, for example:

John and Mary

John and Mary loved each other and decided to be married. Just before the wedding, John discovered that Mary's father was secretly smuggling stolen art through Venice. After struggling with his conscience for days, John reported Mary's father to the police. Mary understood John's decision, but she despised him for it nevertheless; she broke their engagement knowing that he would suffer.

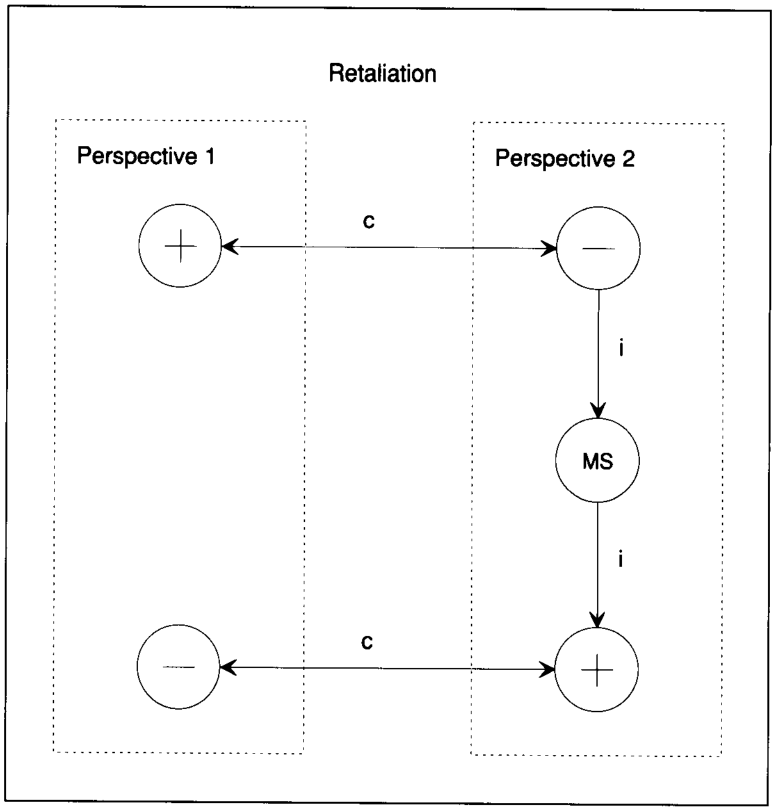


Figure 2.21 Mental states, positive events, and negative events linked across perspectives. In this illustration, there are two perspectives. Each perspective involves a positive event that is seen as a negative event in the other. This particular combination of perspectives, events, and a mental state is called a *retaliation*.

On the surface, "John and Mary" seems to have little resemblance to "Thomas and Albert." More abstractly, however, both involve a central *retaliation* brought on by an *intentional problem resolution* leading eventually to a *loss* and a *positive tradeoff*, both of which finish off a previous *success*. Such similarities are easy to see, once top-level abstraction nets are constructed: the stories' diagrams are exactly the same.

Of course, more complicated stories will have more complicated top-level abstraction nets. Consider, for example, "The Gift of the Magi," a story by O. Henry, with the following plot:

The Gift of the Magi

Della and her husband, Jim, were very poor. Nevertheless, because Christmas was approaching, each wanted to give something special to the other. Della cut off and sold her beautiful hair to buy an expensive watch fob for Jim's heirloom gold watch. Meanwhile, Jim sold his watch to buy some wonderful combs for Della's hair. When they found out what they had done, they were sad for a moment, but soon realized that they loved each other so much, nothing else mattered.

Figure 2.22 Two stories viewed as aggregates of abstraction units, both base and composite. In this illustration, there are two perspectives, Thomas's and Albert's, and six top-level abstraction units.

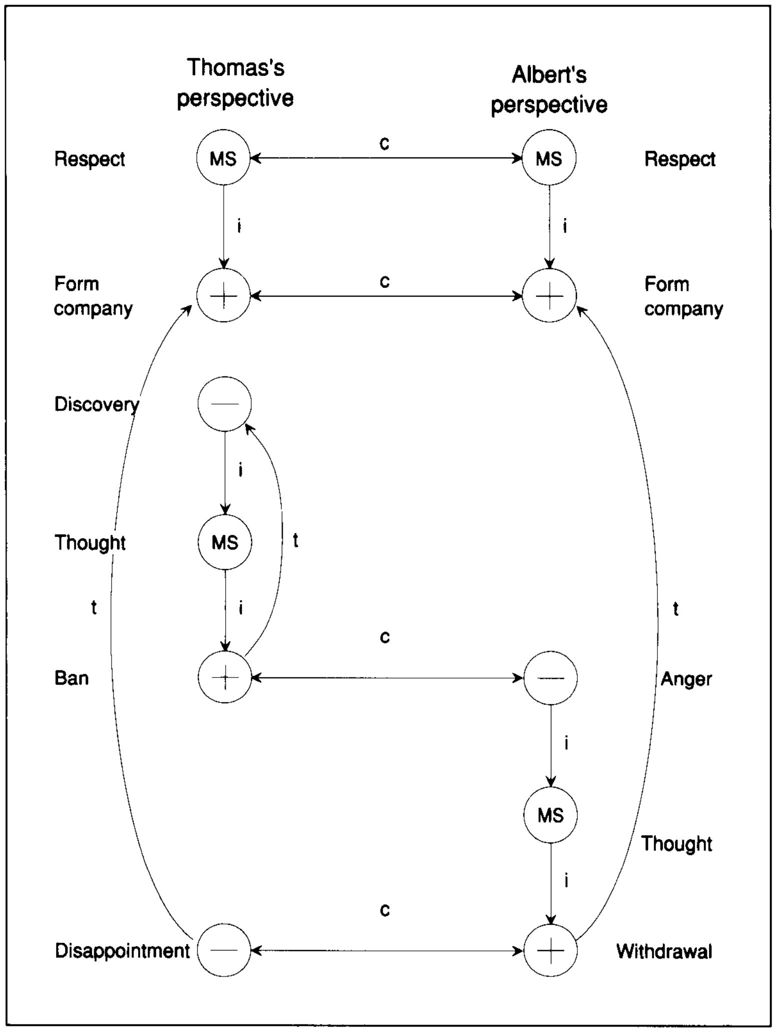


Figure 2.24 shows one plausible set of nodes and links for the story; those nodes and links lead to the top-level abstraction net shown in figure 2.25.

Abstraction Units Enable Question Answering

Abstraction units allow you to answer certain questions by matching. Here are examples:

- What is the story about? Answer by naming the central abstraction unit in the top-level abstraction net. For example, "Thomas and Albert" is about *retaliation.*
- What is the result? Answer by naming the abstraction units that are joined to earlier abstraction units in the top-level abstraction net, but

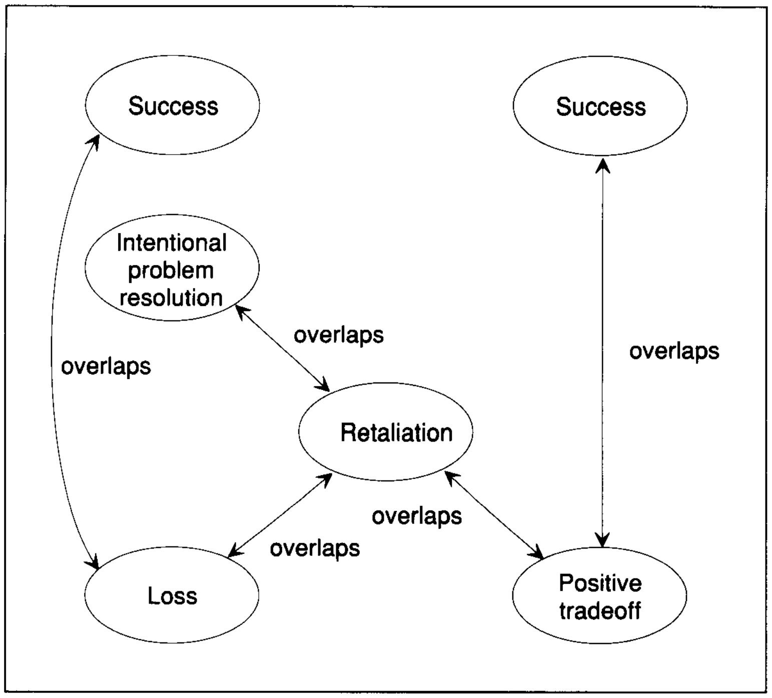


Figure 2.23 A top-level abstraction net formed from the top-level units of figure 2.22. This top-level abstraction net enables you to build a summary description around the most highly linked top-level unit, the *retaliation*.

not to later abstraction units. For example, the result in "Thomas and Albert" is a *loss* and a *positive tradeoff*.

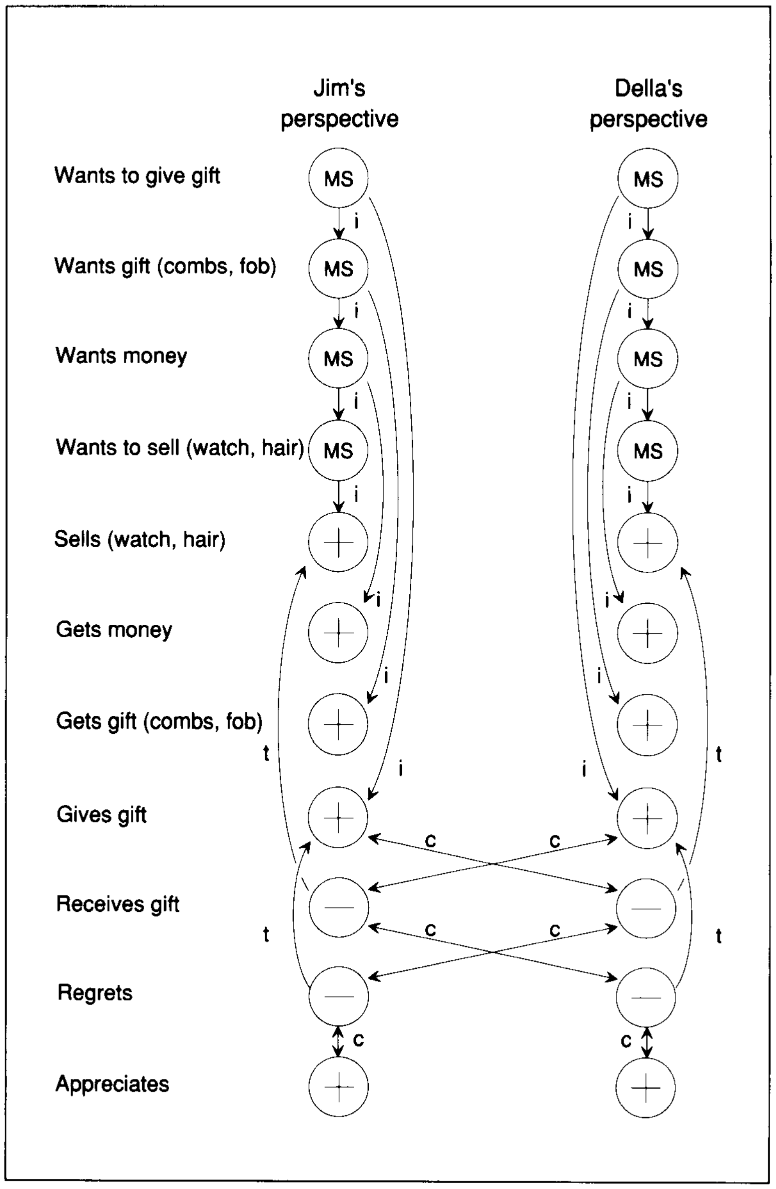
- Does the story involve a certain abstraction? Answer by checking for the appropriate abstraction unit. For example, "Thomas and Albert" does contain an instance of *intentional problem resolution*.
- In what way is one story like another? Answer by naming the most highly connected abstraction unit that appears in both top-level abstraction nets. If hard pressed, enumerate the other abstraction units that appear in both. For example, "Thomas and Albert" is like "John and Mary" in that both involve *retaliation*. Moreover, both involve *success*, *intentional problem resolution*, *loss*, and *positive tradeoff*.

In all these examples, you could give more detailed answers by naming the people and the events involved in the abstraction units mentioned.

Abstraction Units Make Patterns Explicit

In this section, you have seen how a base-unit semantic net facilitates similarity analysis and summary by making mental states, events, and links between them explicit. Thus, the first criterion of good representation—that something important is made usefully explicit—is satisfied. Some people argue, however, that a base-unit semantic net does not yet pass the computability criterion for good representation because there is no fully specified way to translate text into abstraction-unit patterns.

Figure 2.24 The mental states, positive events, and negative events of "The Gift of the Magi."



PROBLEM SOLVING AND UNDERSTANDING KNOWLEDGE

When approaching a new class of problems, to be solved either by you or by a computer, you always should start by asking yourself certain **questions about knowledge**. This section discusses a few questions about knowledge that are particularly important.

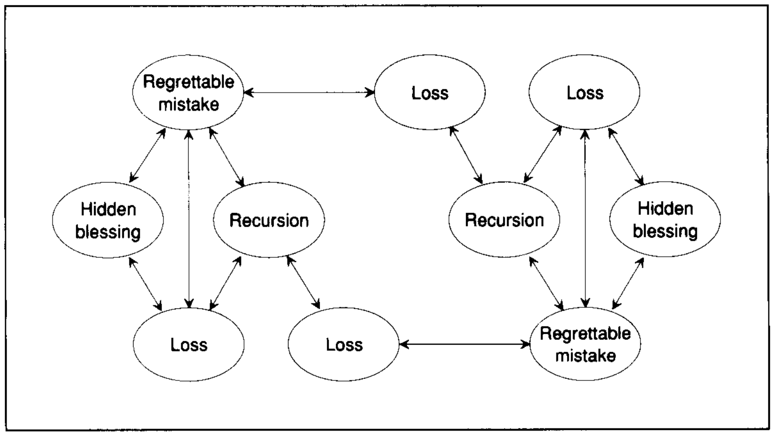


Figure 2.25 The top-level abstraction unit net for "The Gift of the Magi." All links are *overlaps* links.

- What kind of knowledge is involved?

Perhaps the important knowledge concerns the description of concrete or abstract objects. Alternatively, perhaps the important knowledge is about a problem-solving method.

- How should the knowledge be represented?

Some knowledge may, for example, fit nicely within the semantic-net framework. Other knowledge is best embedded in a collection of procedures. There are many possibilities.

- How much knowledge is required?

After learning what kind of knowledge is involved in a task, this question should be the one you ask. Are there 40 things to know, or 400, or 4,000?

One reason to ask about quantity is that you must consider the demand for sensible resource allocation among the various chores required. Another is that knowing the size of a problem builds courage; even if the size is large, digesting bad news is better than anticipating even worse news unnecessarily.

In any event, the tendency is to overestimate grossly; after seeing that a task is reasonably complicated, it is easy to suppose that it is unimaginably complicated. But many tasks can be performed with human-level competence using only a little knowledge.

- What exactly is the knowledge needed?

Ultimately, of course, you need the knowledge. To do geometric analogy problems, you need to know what relations are possible between figure parts, and you need to know how parts can change. To recognize abstractions, you need a library of base and composite abstraction units. Much of learning any subject, from electromagnetic theory to genetics, is a matter of collecting such knowledge.

SUMMARY

- Once a problem has been described using an appropriate representation, the problem is almost solved.
- A representation consists of a lexical part, a structural part, a procedural part, and a semantic part.
- There are many schools of thought about the meaning of semantics. Ultimately, meaning always seems to be rooted in human perception and human intuition.
- Feature-based object identification illustrates the describe-and-match method. An unknown object is identified with an idealized object if their feature points are nearest neighbors in feature space.
- Geometric analogy rules describe object relations and object transformations. You solve geometric analogy problems by determining which rules are most similar.
- Story plots can be viewed as combinations of mental states and events.
- Abstraction-unit nets enable certain kinds of summary and question answering.
- Good representations make important objects and relations explicit, expose natural constraints, and bring objects and relations together.

BACKGROUND

The discussion of geometric analogy problems is based on work by Thomas Evans [1963]. The discussion of plot units is based on the work of Wendy Lehnert; the "John and Mary" story and the analysis of "The Gift of the Magi," in particular, are adapted from one of her highly influential papers [1981].

Feature vectors, and object identification using feature vectors, are described in more detail in *Robot Vision*, by Berthold K. P. Horn [1984].