MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

AIM 520

April 1979
Revised June 1979
Revised January 1980
Revised May 1980

LEARNING AND REASONING BY ANALOGY: THE DETAILS

by

Patrick H. Winston

This paper, in shorter form, has appeared as "Learning and Reasoning by Analogy," *CACM*, vol. 23, no. 12, December 1980.

Abstract

We use analogy when we say something is a Cinderella story and when we learn about resistors by thinking about water pipes. We also use analogy when we learn subjects like Economics, Medicine, and Law.

This paper presents a theory of analogy and describes an implemented system that embodies the theory. The specific competence to be understood is that of using analogies to do certain kinds of learning and reasoning. Learning takes place when analogy is used to generate a constraint description in one domain, given a constraint description in another, as when we learn Ohm's law by way of knowledge about water pipes. Reasoning takes place when analogy is used to answer questions about one situation, given another situation that is supposed to be a precedent, as when we answer questions about *Hamlet* by way of knowledge about *Macbeth.*

The input language used and the treatment of words implying CAUSE have been improved. AIM 632, "Learning New Principles from Precedents and Exercises," describes these improvements and subsequent work. It is, at this writing, in publication in the *Artificial Intelligence Journal.*

This research was done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Office of Naval Research under contract N00014-77-C-0389 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

ANALOGY

Much thinking is done by analogy. We face a situation, we recall a similar situation, we match them up, we reason, and we learn. We use analogy when we say some situation is likely to be a Cinderella story and when we learn about resistors by thinking about water pipes. Experts in Economics, Medicine, and Law use analogy to relate new situations to case studies.

This paper presents a theory of analogy and describes an implemented system that embodies the theory. The paper begins with a presentation of some examples that further illustrate the sort of reasoning and learning to be understood, followed by a discussion of how one can tell that there has been some success. Next, there is a specification of a representation and an exploration of principles that seem to enable reasoning and learning to be done with the aid of the representation. And finally, an implemented system is presented that actually does reasoning and learning.

The implemented system has a number of key ingredients, the following in particular:

- **Extensible-relations representation.** Situations are represented using relations between pairs of parts. Supplementary descriptions can be attached to the relations when elaboration is needed.

- **Importance-dominated matching.** The similarity between two situations is measured by finding the best possible match according to what is important in the situations as exhibited by the situations themselves. Various kinds of constraint relations help determine importance. Cause is a common importance-determining constraint.

- **Analogy-driven constraint learning.** A constraint such as Ohm's law is learned as a by-product of mapping the parts of a situation in a well-understood domain into the parts of another situation in an ill-understood domain.

- **Analogy-driven reasoning.** Some questions about a situation ask if a particular relation holds. Causes found in a remembered situation can supply suggestive precedents.

- **Classification-exploiting hypothesizing.** Memory is searched for situations that are likely to be similar to a new, given situation. The search assumes that the useful remembered situations will involve the same sorts of things as the new one at some level of classification.

The system is presumed to work with situations that are subject to certain restrictions:

- **Symbolic sufficiency.** A situation can be described by using a repertoire of classes, properties, acts, and other relations. The repertoire need not be so large as to make matching complicated.

- **Description-determined similarity.** A situation is similar to another if the important relations in their descriptions can be placed in correspondence.

- **Constraint-determined importance.** The important relations of a situation are the ones explicitly said to be important by some teacher or implicitly known to be important by being involved in constraint relations. Often the constraint relations have to do with various forms of cause.

- **Historical continuity.** A situation that is similar to a past situation generally leads to similar results or conclusions.

THE CRITERIA FOR SUCCESS AND THE COMPETENCE TO BE UNDERSTOOD

In any scientific work, it is necessary to have some way of determining success. For this work, claims for success are with respect to the following criteria:

- There must be an implemented program that performs a specified task.

- The implemented program must perform by virtue of identifiable principles.

In Artificial Intelligence, having such criteria for success in mind helps avoid tendencies either to be romantically speculative about the power of vague ideas or to be overcome by the performance of working, but *ad hoc* programs. For this particular work, part of the specified task is to do reasoning and learning by analogy as required by the following representative scenario:

- A teacher tells a student that the voltage across a resistor can be calculated by thinking about the water pressure across a length of pipe. The student correctly finds the voltage without knowing Ohm's law.

- The teacher instructs the student in two different voltage-resistance-current situations and tells the student to formulate a law. The student invents Ohm's law.

- The teacher suggests generalizing from the water-pipe law and Ohm's law. The student formulates a linear constraint that involves forces and flows.

Thus, practice with some specific situations in one domain enables the invention of a specific law. In the other direction, once several forms of the same sort of law are known, comparison enables generalization.

Another part of the specified task is to reason by analogy, as required in the following representative examples:

- A plot outline is given in terms of 40 or 50 facts. The plot appears reminiscent of several of Shakespeare's tragedies. Analysis suggests that it is most like *Macbeth*. Someone asks if the person that corresponds to Macbeth will end up dead. Reasoning by analogy suggests asking if the person that corresponds to Lady Macbeth persuaded the Macbeth equivalent to murder the Duncan equivalent.

- The case of Smith versus Wesson establishes a precedent for assault cases. Smith pointed a rifle at Wesson to frighten him. The rifle was not loaded, and it was therefore harmless. Nevertheless, an assault has taken place because Wesson did not know that the rifle was not loaded. Subsequently, Smith versus Wesson is retrieved when the case of Villain versus Victim is considered. Villain pointed a pistol at Victim in order to frighten him. The pistol was harmless toy. Reasoning by analogy suggests asking if Victim knew that the pistol was harmless.

All of these examples, having to do with both learning and reasoning, have been handled successfully in a series of experiments using an implemented system. Given that the examples suggest an interesting level of competence, it remains to be more precise by showing the input to and the output from the implemented programs and to demonstrate that the implemented programs work by virtue of identifiable principles.

The term *competence*, incidentally, is used in the sense intended by Chomsky, namely to refer to the *knowledge* necessary for any particular set of algorithms to exhibit stated behavior. *Performance* has to do with the *use* of knowledge to exhibit stated behavior. Implementation of a particular set of algorithms, from this point of view, is done so that performance can help evaluate progress toward understanding competence.

REPRESENTING SITUATIONS USING EXTENSIBLE RELATIONS

Broadly speaking, a representation is a vocabulary of symbols together with some conventions for arranging them. A good representation is one that has the following coupled characteristics:

- It makes the important facts explicit.
- It suppresses irrelevant detail.
- It is perspicuous.
- It exposes constraint.
- It can be computed from a natural input.

In this section, a particular representation will be explained. It is based on two key assumptions: first, that what needs to be represented can be expressed in simple English; and second, that much of what can be expressed in simple English can be thought of as consisting of an act-specifying verb and some noun-centered word groups that the act ties together.

For the simplest sentences, the nouns involved are just the agent and the object of the act. For others, the agent and object are supplemented by other things that participate in act description, such as an instrument, a time or location, or perhaps a source or destination if the act involves motion. In the following, for example, an instrument is specified:

Prince Charming found Cinderella with her glass shoe.

Knowing the kinds of things that are involved in describing an act and understanding how to recognize those things in sentences is the objective of *case-grammar theories* of sentence meaning. Agents, objects, instruments, and similar terms are used as names for *case slots*. Sentence analysis is viewed as the job of filling case slots using sentences as raw material.

Extensible-relation Representation

From the perspective of case grammar, the most obvious way to strip a situation description of its syntactic nuances and to represent its approximate meaning is to use the material supplied by the sentences in the description to fill in an act-oriented schema, as illustrated by the following rendering:

Action: Find
Agent: Charming
Object: Cinderella
Instrument: <name for Cinderella's shoe>

The disadvantage of such a representation is that it favors situation matching by finding correspondences between act-oriented schemata, rather than between situation parts. Introspectively, I feel that I form analogies by pairing off situation parts, using acts and other relations as evidence, not the other way around.

Consequently, I use an alternative, object-oriented representation. Parts of situations are represented as nodes that are tied together with relations forming a kind of semantic net. When more than an agent and an object is involved in an act, a supplementary description is tied to the act-specifying relation, making the representation just as capable of bearing case-like information. The supplementary description itself consists of a node related to other nodes as illustrated by figure 1. As a

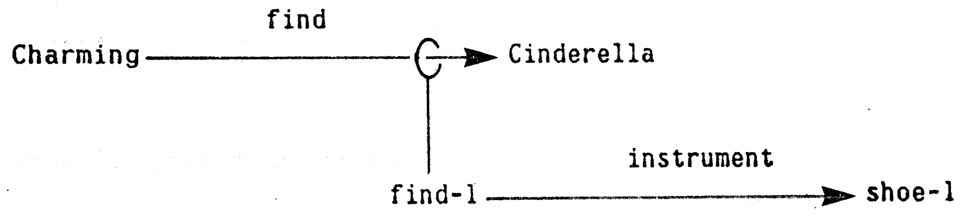


Figure 1: A FIND relation further described by a supplementary description node.

consequence of the ability of the supplementary description node to say a lot about a relation, this is called an *extensible-relation* representation.

Using this extensible-relation representation, the agent and the object involved in an act have an explicit prominence. Since agents and objects are generally among the parts of a situation, the representation favors situation matching by finding correspondences between situation parts, which I think seems natural.

Recall that this discussion began with the assumption that much of simple English consists of sentences whose information content can be captured by a representation that views sentences from the perspective of case grammar. There are other kinds of sentences, however. Some indicate that an object belongs to some class; some indicate that an object has some property; and some indicate that a particular relationship holds between two objects:

Charming is a prince.
Cinderella is beautiful.
Cinderella loves Charming.

Information about classes and properties is easily conveyed by using A-KIND-OF and HAS-PROPERTY relations. Information about relationships between two objects is handled by state relations such as LOVES just as if the relationship were an act. Note that A-KIND-OF, HAS-PROPERTY, and state relations can be tied to supplementary description nodes although no such nodes are demanded by the examples just given.

Importantly, the relations from and to a supplementary node all come from a limited vocabulary because the purpose of a supplementary node is to tie together all the participants in a constraint. This limited vocabulary consists of the following: first, case names, such as *instrument*, used in describing constraints couched in the form of simple act-centered sentences; second, case-like names, such as *multiplier*, used in describing algebraic constraints; and third, names like *cause*, used to show how constraints themselves are constrained.

The actual implementation was done using a version of FRL, an acronym for a LISP-based Frame Representation Language, developed by Bruce Roberts and Ira Goldstein [14, 15]. FRL was used because FRL has handy mechanisms for asserting relations and attaching supplementary descriptions to them. In FRL terms, an agent-act-object combination is expressed as a *frame*, a *slot* in the frame, and a *value* in the slot. A supplementary description node for an agent-act-object combination is expressed in the form of a so-called *comment frame* attached to the frame-slot-value combination.

■ As a consequence of the FRL implementation, my habit is to use the terms *frame* and *value* when referring to nodes, to use the term *slot* when referring to relations of all kinds, and to use the term *comment frame* when referring to the nodes which bear supplementary descriptions. Note that comment frames are part of the knowledge representation -- they are not notes to human programmers.

As it stands, many standard questions have been put off, particularly those involved in the recording of information about quantification, negation, disjunction, and perspective. These questions were put off since dealing with them was not forced by the situations considered in this work so far.

English-like Input

It was pointed out that one characteristic of a good representation is that it can be computed from a natural input. This means that the extensible-relation representation of a situation should be computable from a simple English description of the situation or from something that is at least close to simple English.

In fact, a translator that translates English-like situation descriptions into descriptions in extensible-relation representation has been designed and implemented. It was designed with a view toward compromise between maximum input transparency and minimum translator complexity. It requires only minimal syntactic, semantic, and reference-finding machinery since it accepts a kind of parenthesized English notation in which the case names are explicitly given. For example, part of the sentence "Prince Charming found Cinderella with her glass shoe," is expressed as follows:

> Charming find Cinderella [instrument shoe-1].

This sentence illustrates the basic form for conveying an agent-act-object combination with case information. Material enclosed in square brackets gives case-like information to be hung on the comment frame that further describes the agent-act-object combination just asserted. Thus the expression [instrument shoe-1] generates a comment frame for the Charming-find-Cinderella combination and hangs a case-filler combination on it. More about the shoe is conveyed as follows:

> Shoe-1 a-kind-of shoe prop - raw-material glass - owner Cinderella.

This sentence illustrates that the input language permits a kind of elision. When the same agent-act combination is used with several objects, the objects are just enumerated one after the other. When the same agent is involved with several acts, hyphens keep things separated.

A minimal reference feature makes it possible to use the following, simpler forms, in which specific names are avoided:

> Charming find Cinderella [instrument a shoe].

> The shoe is a prop - raw-material glass - owner cinderella.

Expressions like "a shoe" generate instances attached to the named class, SHOE in this case. Expressions like "the shoe" simply refer to the last instance created of the named class. Expressions like "is a prop" are taken to mean that an A-KIND-OF relation is implied.

Finally, it is often useful to have a direct way to refer to an agent-act-object comment frame. This is done by embedding the agent, act, and object involved in curly brackets. Thus either of the following indicates that Charming's love for Cinderella causes him to kiss her:

> Charming love Cinderella [cause {Charming kiss Cinderella}].

> {Charming love Cinderella} cause {Charming kiss Cinderella}.

Figure 2 shows what all of this information looks like when translated into the graphical form of the extensible-relation representation.

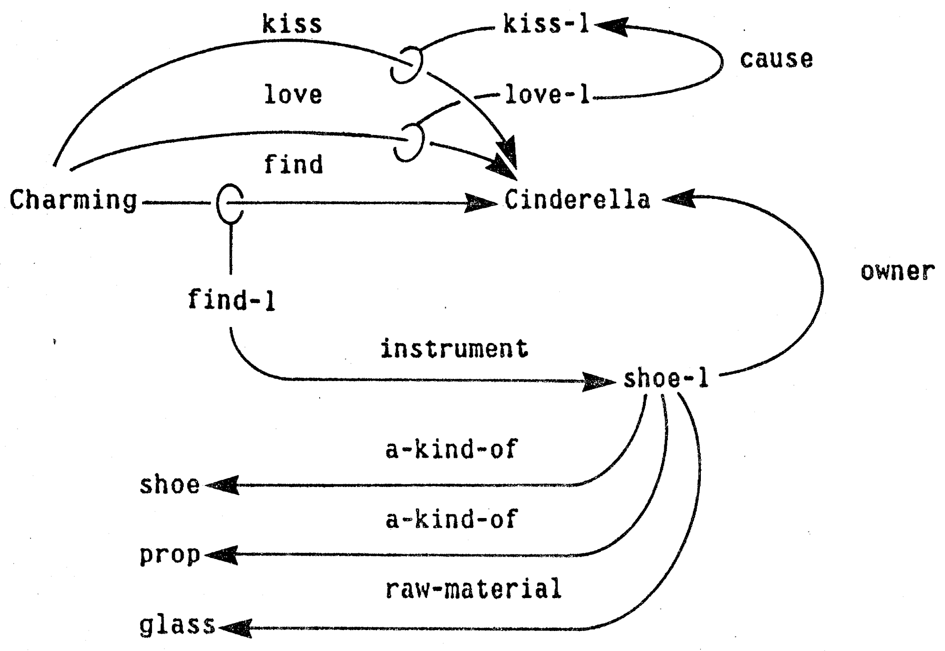


Figure 2: The input translator produces a constellation of nodes and relations from a few English-like sentences.

Demons

In addition to English-like input, it is good to have some deductions made automatically, reducing the need for tedious attention to details. For example, given that Cinderella is married to Prince Charming, Prince Charming is clearly married to Cinderella. Similarly when one person kills another, it is clear that the killed person is dead.

One way to arrange for such obvious deductions is to use procedures that are invoked when relations are inserted in the data base. Such procedures are commonly called *if-added demons*. In the version of FRL used, if-added demons that trigger on the use of a particular relation are placed in a frame describing that relation.

In the end, demons prove so important that there has to be some concern about whether they can be learned. In fact they can be because using a demon is like doing an analogy in miniature. The ideas that make it possible to accumulate big chunks of experience are the same as the ideas that explain how it is possible to remember cause-effect relations at the level of demons.

An immediate question is whether demon use should be limited, and if so, by what control scheme. At the moment, there are no answers.

Capturing Story Plots Requires Attention to Cause

Using extensible relations, ten story plots were set down. The following is what one version of one of these, namely *Macbeth*, looks like in the English-like notation:

```
MA is a story.

{Macbeth is a noble} before {Macbeth is a king}. Macbeth
marry Lady-macbeth. Lady-macbeth is a woman - has-property
greedy ambitious. Duncan is a king. Macduff is a noble -
has-property loyal angry. Weird-sisters is a hag group -
has-property old ugly weird - number 3.

Weird-sisters predict {Macbeth murder Duncan}. Macbeth
desire {Macbeth a-kind-of king} [cause {Macbeth murder
Duncan}]. Lady-macbeth persuade {Macbeth murder Duncan}.
Macbeth murder Duncan [coagent Lady-macbeth - instrument
knife]. Lady-macbeth kill Lady-macbeth. Macbeth murder
Duncan [cause {Macduff kill Macbeth}].
```

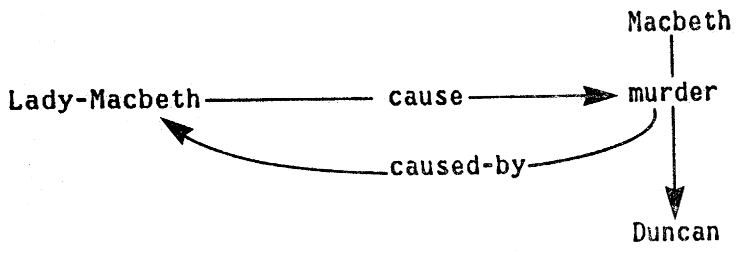
In addition to *Macbeth*, the plots set down include three other Shakespearean tragedies, one of his comedies, two plays by Ibsen, and some random things, all selected by thumbing through an encyclopedia of plots. The purpose was to discover if the representation and accompanying vocabulary are adequate for reasoning with plots by analogy. The following observations were made:

- A few basic English words adequately supplied by far the bulk of those needed for slot names and classes. Most of the ones I use are in Ogden's thousand-word Basic English vocabulary [12] and in the first thousand or two most frequent English words [2].
- Cause is important because cause constrains relation pairs.

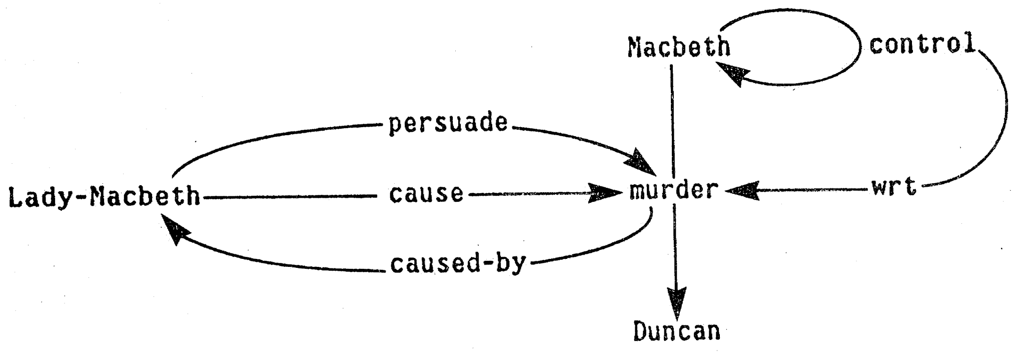
Many people have attended to the role of cause, particularly Schank [17] and Wilks [19]. To handle cause here, the following conventions were honored, based on the work of Givon [5]:

- Acts, relations, and people can cause or prevent acts and relations. The inverses of CAUSE and PREVENT are CAUSED-BY and PREVENTED-BY. See figure 3a for what happens given this fragment:

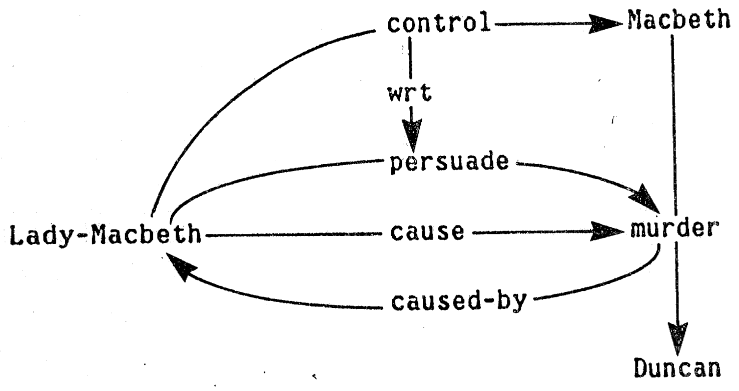
```
Lady-Macbeth cause {Macbeth murder Duncan}.
```



3a: Lady-Macbeth cause {Macbeth murder Duncan}.



3b: Lady-Macbeth persuade {Macbeth murder Duncan}.



3c: Lady-Macbeth force {Macbeth murder Duncan}.

Figure 3: The relations placed by CAUSE, PERSUADE, and FORCE.

- People can persuade and dissuade. Persuade indicates cause and a CAUSE relation is therefore generated by a demon whenever PERSUADE is used. Dissuade similarly indicates prevent. Since persuaders and dissuaders normally intend for something to happen, INTEND relations are generated too, again by demons. Since the person persuaded or dissuaded retains control of what is happening, a demon-placed CONTROL relation so indicates. See figure 3b for an example showing what happens given this:

  Lady-Macbeth persuade {Macbeth murder Duncan}.

  People can also order and forbid. For the moment ORDER and FORBID simply carry demons that place PERSUADE and DISSUADE relations and trigger their demons in turn.

- People can force. Force is like order and persuade, except that the person forcing has control of what is happening. See figure 3c for an example showing what happens given this use of a FORCE relation:

  Lady-Macbeth force {Macbeth murder Duncan}.

DETERMINING ANALOGOUS PARTS USING IMPORTANCE-DOMINATED MATCHING

Analogy is based on the assumption that if two situations are similar in some respects, then they must be similar in other respects as well.

To determine if two situations are similar, the parts of the situations must be placed in correspondence. The purpose of matching is to establish the best way to do this. In general, the best way will leave some of the parts' classes, properties, acts, and other relations unpaired, because if two situations were exactly alike, nothing could be inferred about one by using the other. A matcher for use in analogy must be flexible, not rigid.

It is easy to be seduced into worrying about matching for its own sake, without attention to the sorts of things to be matched. This typically leads to the invention of all sorts of mechanisms of doubtful value in practice. Consequently, the matcher described here was developed by implementing only those mechanisms needed to attend to the factors that demonstrably influence similarity.

Recall that the parts of a situation are the new instances generated by the input interface whenever the articles *a* and *an* are used. Given that matching is to pair the parts of two situations, three general issues require thought: first, how is the space of all possible matches of the parts of two situations to be searched; second, what is to constitute a quantum of evidence for a particular match, and third, how are the quanta of evidence for a particular match to be combined to produce an overall measure of similarity.

The issue of how to search the space of all possible matches seemed the best issue to neglect in the early stages of this research. The reason is that efficiency is best addressed after it is established that there is something worthwhile to be efficient about. Consequently, the implemented matcher does a brute-force search of the space of possible matches, calculates how good each is, and announces the best.

The price paid is that the situations matched may not involve more than a handful of parts. In general, if there are N1 parts in one situation and N2 in another, then the number of ways the situations can be paired up is N1!/(N1-N2)!, given that N1 is equal to or greater than N2.

At first thought, trying all possible pairings seems hopeless since the number of possible pairings gets big fast. For small N1 and N2, the number is manageable. The implemented matcher, when compiled, handles 100 or so pairing possibilities without excessive strain. For larger numbers, something must be done to constrain the number of pairings considered.

Given a way of searching the space of all possible matches, the next issues have to do with finding and combining evidence so that the best match can be identified. Thought about these issues led to the following conclusions:

- To exploit an analogy between two situations starts with establishing the best correspondence between the parts of the situations using the parts' classes, properties, acts, or other relations as evidence, depending on what is important.

- To establish what is important in a situation may require attention to constraint. This in turn usually means looking at the causal relations exhibited in the situation.

- To establish enough of what is important in a situation may require expansion of some facts in the direction of more detail or abstraction of some facts in the direction of more generality.

- To establish enough of what is important in a situation may require asking some questions.

- To combine evidence, simply counting the individual items of evidence is sufficient to handle the particular tasks involved in determining success.

Plainly, the issue of what constitutes an item of evidence requires further discussion. Examples will deal with both story plots and natural laws.

Finding Correspondence can Require Attention to Properties and Relations

Suppose, for example, that Prince Charming and Cinderella are the characters in one plot and Romeo and Juliet are the characters in another. (These names were picked so that the combinations are mnemonic -- the plots are not developed in this illustration.)

> Charming job entertaining - has-property brave strong - love Cinderella - kiss Cinderella.
>
> Cinderella job cleaning - has-property beautiful.
>
> Romeo job fighting cleaning - has-property strong - love Juliet - kiss Juliet.
>
> Juliet has-property beautiful.

Knowing just these facts intuitively indicates that CHARMING corresponds to ROMEO and CINDERELLA to JULIET. The implemented matcher agrees because it tries all possible matches and because its simple scoring module awards one point to each shared relation.

In doing its job, the matcher produces a set of paired frames for each possible match. Each set of paired frames is referred to as a list of linked pairs. Each list of linked pairs is evaluated by calculating a similarity score for each linked pair in the list and adding the results together. The score for each linked pair is calculated by scoring one point if the two linked frames contain the same value in some particular slot or if two linked frames contain the two parts of another linked pair in some particular slot.

The matcher therefore produces a score of four for the intuitively correct match of CHARMING and CINDERELLA with ROMEO and JULIET, but only one for the incorrect one.

The details of the matcher's scoring module are not important. The important thing is that it seems reasonable to use both relations and properties as evidence in lieu of specific information about what is important. We will see that relations and properties are not sufficient, however.

Finding Correspondence can Require Attention to Corresponding Comments

For the sake of illustration, suppose Prince Charming's love for Cinderella causes him to kiss her and Romeo's love for Juliet has the same result, indicated by the following:

{Charming love Cinderella} cause {Charming kiss Cinderella}.

{Romeo love Juliet} cause {Romeo kiss Juliet}.

Certainly there is evidence for pairing CHARMING with ROMEO and CINDERELLA with JULIET since such a pairing places the LOVE and the KISS slots in correspondence. However it is evident that pairing the characters in this way also puts LOVE-1 and KISS-1 in correspondence with LOVE-2 and KISS-2, which puts a CAUSE and a CAUSED-BY relation in correspondence, producing additional evidence of similarity.

To account for the additional similarity available in comment frames, the implemented matcher adds corresponding comment frames to each possible list of linked pairs after it is formed but before it is scored. Thus the corresponding comment frames are considered linked when scoring other frames, and they themselves are scored.

Thus the total score should be six for the two situations under consideration. Figure 4 illustrates the combinations that lead to this score in graphic form.

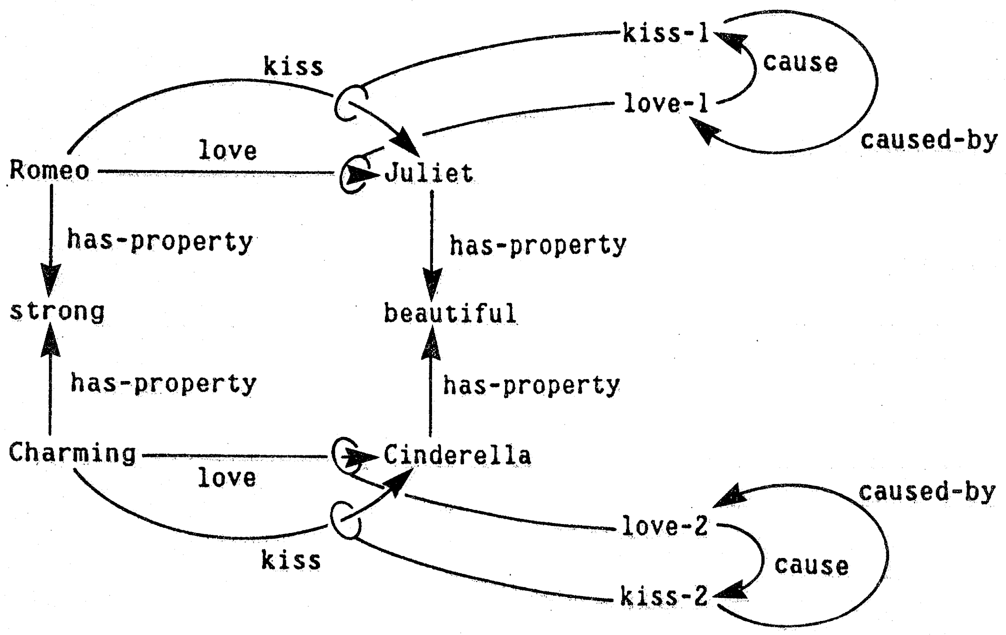


Figure 4: Matching *Cinderella* with *Romeo and Juliet* produces a match score of six. Having the same properties accounts for two points; having the same relations between the people accounts for two more; and having corresponding causal relations between the comment frames accounts for the final two.

Finding Correspondence can Require Attention to Classification Information

So far it would not help to add the following information:

Charming is a prince. Romeo is a boy.

Cinderella is a princess. Juliet is a girl.

Knowing that CHARMING is A-KIND-OF PRINCE and ROMEO is A-KIND-OF BOY lends no direct strength to their similarity, nor does it help to know that CINDERELLA is A-KIND-OF PRINCESS and JULIET is A-KIND-OF GIRL. These facts do lend indirect strength to the two pairs because the given classifications indicate common classification at higher levels. After all, a prince and a boy are the same sex as are a princess and a girl.

To account for this indirect evidence of similarity, the implemented matcher treats all A-KIND-OF slots as if they contained everything that is found by tracing through the A-KIND-OF hierarchy that leads from them. The A-KIND-OF slot of CHARMING contains only PRINCE, but it is treated as if it contained PRINCE, MAN, and PERSON. Similarly ROMEO's A-KIND-OF slot is treated as if it contained not only BOY, but also MAN and PERSON. Consequently the A-KIND-OF slot contributes a score of two when CHARMING and PRINCE are paired. Similarly CINDERELLA is a PRINCESS, a WOMAN, and a PERSON while JULIET is a GIRL, a WOMAN, and a PERSON. Their A-KIND-OF slots contribute a score of two as well. Thus the A-KIND-OF slots of all the parts lead to a match score of ten, four more than before. Figure 5 shows the A-KIND-OF hierarchy that gives these extra points.

An objection to this type of scoring is that the A-KIND-OF hierarchy might involve long chains that would tend to cause classification information to dominate matching. Happily, long A-KIND-OF chains do not occur in the situations considered in this work so far, so the objection has been noted, but not thoroughly studied.

If long A-KIND-OF chains should prove to be a problem, the work of Rosch *et al.* may be relevant [16]. They argue that the world of human experience is such that there is a so-called basic level of class abstraction in the A-KIND-OF hierarchy. At this basic level, two things are true: at the next level up, the members of the classes share substantially fewer properties than at the basic level; and at the next level down in the hierarchy, the members of the classes share about the same number of properties as at the basic level. Concepts like guitar, apple, hammer, shirt, table, and car are at the basic level. Musical instrument, fruit, tool, clothing, furniture, and vehicle are higher. Grand piano, Mackintosh apple, ball-peen hammer, dress shirt, kitchen table, and sports car are lower.

Above the basic level, common class membership means little. Below the basic level, common class membership adds little. Consequently, it might be reasonable to score matching points only for common classification

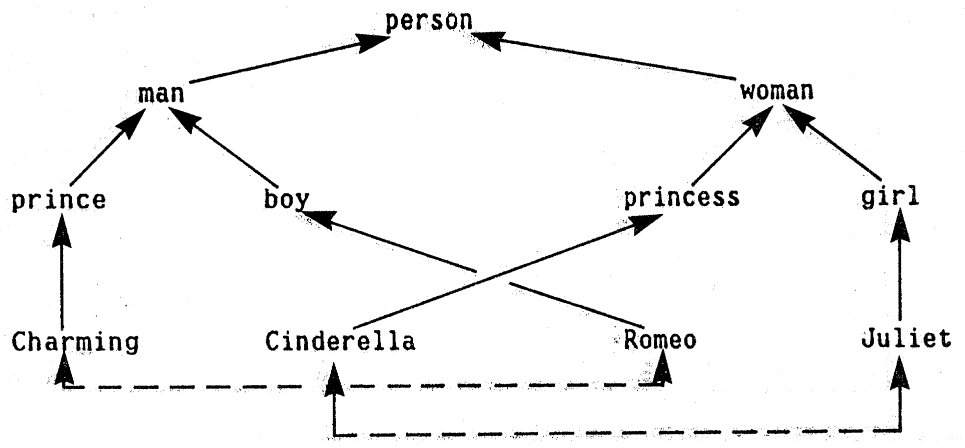


Figure 5: The A-KIND-OF hierarchy is exploited in matching. The A-KIND-OF slots contribute four points to the scoring of the match with ROMEO paired with CHARMING and CINDERELLA with JULIET even though each has something different in its A-KIND-OF slot.

at the basic level, ignoring common classification above and below, finessing the problem of long A-KIND-OF chains.

Constraint Makes some Relations more Important than Others

There is some debate about whether a matcher should distinguish among the kinds of information available for matching. One view is that classification information is the most important. Another view favors properties.

Still another, milder view is that all information is important, but to a varying degree that has to be accounted for by a weighting scheme, possibly context dependent. This can quickly give the matcher an *ad hoc* feel. It is disturbing when a program must be tuned up by fooling with a system of parameters.

Nevertheless, some relations are more important than others because they lead to the conclusions that are to be exploited in the analogy process. The important relations are sometimes A-KIND-OFs, sometimes HAS-PROPERTYs, and sometimes other relations, some of which are normally incidental.

Thankfully, importance tends to be taught by teachers, either explicitly or implicitly. Explicit teaching is done when a teacher says, perhaps without justification, that some fact is important. Implicit teaching is done when a teacher includes some fact in a constraint, typically in the form of a causal chain. Thus I take the following position:

- Any relation can be important in matching. The importance of a particular relation can be determined by remembering what teachers have said about it or by noting whether it is involved in constraining something. Causing something is an important way of constraining.

For the most part, the examples in this paper assume a beneficent teacher who gives only the relevant facts and who does not deliberately try to confuse the system by shoveling detritus at it. It is important, however, to understand that mechanisms have been implemented that pay attention to importance on demand.

In particular, the matcher can be told to use only relations that have comment frames with IMPORTANT in the HAS-PROPERTY slot. The HAS-PROPERTY slot of a comment frame can have IMPORTANT placed in it directly as in the following example:

Macbeth kill Duncan [has-property important].

Alternatively, the HAS-PROPERTY slot can have IMPORTANT put in by a demon placed in the CAUSE frame. Using this demon, all frames at either end of a cause relation are noted to be important, as well as the cause relation itself.

Actually, the implemented strategy represents one end of a spectrum of possibilities. As it stands, relations never become globally important. A looser strategy would make a relation important everywhere in a situation if it is determined to be important somewhere in the situation. And a still looser strategy would make a relation important everywhere in a situation if it is important somewhere in some other situation of the same general class.

Matching Large Groups may Require some Preliminary Classification

As the size of two groups to be matched becomes large, trying all possibilities becomes intractable. There are two choices: throw away the exhaustive matcher and do something else, or somehow prune the collection of matching alternatives that the matcher generates. The implemented matcher prunes:

- One way to limit the matching alternatives is to restrict the pairings to those that link together only frames of the same class, as specified by instructions to the matcher.

For example, if there are two groups of people to be matched, and each contains, say, three men and four women, then the total number of match alternatives is:

$N1!/(N1-N2)! = 7! = 5040$

But if the matcher is instructed to link men only with men and women only with women, then the number is:

$M1!/(M1-M2)! \times W1!/(W1-W2) = 3! \times 4! = 6 \times 24 = 144$

The smaller number is only 3% of the larger. Of course it is no longer possible to discover a male Cinderella, a defect that may suggest a similar difficulty when people must deal with analogies involving many parts. To prevent too many blunders of this sort requires some way of selecting a good set of classes for the matcher. There may be some way of doing this by inspecting the A-KIND-OF hierarchy in the vicinity of the frames involved in the match.

Matching finds Corresponding Parts in Plots

Here are some results showing the match scores between the four Shakespearean tragedies and one comedy:

	MA	HA	JU	OT	TA
MAcbeth	78	49	45	21	9
HAmlet	49	108	35	22	9
JUlius Caesar	45	35	91	28	8
OThello	21	22	28	71	10
TAming of The Shrew	9	9	8	10	50

The choice of Shakespearean tragedies was somewhat ill-advised since they lean toward the macabre. Nevertheless, it is interesting that the tendency to have evil, murder, and death everywhere in sight does make them more similar to each other than to the comedy.

The average score on the diagonal is 80. Evidently the average number of facts known about each plot is therefore 80. Some of the facts are derived by demons and others are implied by the A-KIND-OF connections. A demon on MURDER creates a KILL and links the MURDER and the KILL together with a CAUSE relation. A demon on KILL leads to instances of HAS-PROPERTY and CAUSE because killing someone causes that person to have the property of being dead. Also, murderers are noted to have the EVIL property.

It is instructive to look at the best and worst off-diagonal matches to see if they make sense. Evidently Macbeth and Hamlet show the most similarity. The matcher announces its view as follows:

```
49. values match with 78. and 108. possible -- Best match
is 13. better than next best.

(49. (10. MACBETH CLAUDIUS)
     (5. DUNCAN GHOST)
     (4. LADY-MACBETH GERTRUDE)
     (4. MURDER-1 MURDER-2)
     (4. KILL-1 KILL-4)
     (4. KILL-3 KILL-5)
     (3. MACDUFF HAMLET)
     (3. KILL-2 KILL-8)
     (2. HQ-3 HQ-5)
     (2. HQ-2 HQ-9)
     (2. HQ-1 HQ-4)
     (1. WIERD-SISTERS LAERTES)
     (1. CAUSE-6 CAUSE-11)
     (1. CAUSE-5 CAUSE-20)
     (1. CAUSE-1 CAUSE-8)
     (1. CAUSE-2 CAUSE-9)
     (1. CAUSE-7 CAUSE-12)
     (0. AKO-2 AKO-3))
```

This makes some sense. Macbeth and Claudius both kill a king so as to become king and both are killed in turn. Their victims are Duncan and the Ghost. Macduff and Hamlet kill them. Their wives are Lady Macbeth and Gertrude.

On the other hand, *The Taming of the Shrew* and *Julius Caesar* show little similarity. In fact, there are only two points of similarity beyond the fact that there are four people to pair up and two of the four have the same sex. The score of eight is at the level of background noise.

Evidently, the plots as given to the matcher do not exhibit much diverting detail because the general shape of the table is the same when the matcher counts only the relations that are demonstrably important. The following revised table shows this:

	MA	HA	JU	OT	TA
MAcbeth	16	12	11	5	0
HAmlet	12	28	9	5	0
JUlius Caesar	11	9	21	7	0
OThello	5	5	7	21	0
TAming of The Shrew	0	0	0	0	11

The scores are much reduced, but still *Macbeth* and *Hamlet* are similar while *The Taming of the Shrew* and *Julius Caesar* are not.

In producing the scores in this table, the matcher counted only relations marked as important. Demons on CAUSE and PREVENT do the marking.

Abstraction may be Necessary before Matching

Matching may require some preliminary act abstraction, exploiting the fact that acts like KILL imply more abstract acts like HURT. Suppose, for example, that two situations are proposed, one involving a tragic event, and the other, a person in conflict with himself:

> TRAGIC-EVENT is a situation.
>
> Evil-person is a person. Good-person is a person. Evil-person hurt Good-person. Evil-person has-property evil. Good-person has-property good.
>
> SELF-CONFLICT is a situation.
>
> Grubbla is a person. Grubbla has-conflict-with Grubbla.

As they stand, these situations are abstractions of some of the things that go on in the ten experimental plots, but neither has anything explicitly in common with any of them, other than that people are involved and that some explicit hurting goes on in Ibsen's *A Doll's House* and Shaw's *Pygmalion*. Good and evil are nowhere to be found.

A kind of abstraction can make good matches happen anyway. It is easy to put demons on HAS-PROPERTY and MURDER so that loyal people are noted to be GOOD and murderers are noted to be EVIL. Other demons can generate HURT and HAS-CONFLICT-WITH relations, given KILL. For the tragic event situation, these demons enable the matcher to produce the following results, where the numbers give the actual matching scores and the percentages of the maximum possible scores:

```
The matches, in order of quality, are:
5.        83. %   Macbeth
5.        83. %   Hamlet
5.        83. %   Othello
5.        83. %   Julius Caesar
4.        66. %   Dolls House
4.        66. %   Pygmalion
4.        66. %   Adam and Eve
3.        50. %   Cinderella
2.        33. %   Taming of the Shrew
2.        33. %   Hedda Gabbler
```

The act of murder made Macbeth an evil person and established that he hurt Duncan. The plot lacks perfect match with the description of a tragic event because it was not stated that Duncan was a good person, nor was it deduced. Hamlet and Julius Caesar similarly fail to match perfectly. The Ghost and Caesar are not known to be good.

Othello fails because Othello kills Desdaemona, but killing, unlike murdering, does not imply a person is evil. Desdaemona, however, is good because she is loyal.

Now consider the other situation, the one that involves self-conflict:

```
The matches, in order of quality, are:
2.        100. %  Hamlet
2.        100. %  Othello
2.        100. %  Julius Caesar
2.        100. %  Hedda Gabbler
1.        50. %   Macbeth
1.        50. %   Taming of the Shrew
1.        50. %   Dolls House
1.        50. %   Pygmalion
1.        50. %   Cinderella
1.        50. %   Adam and Eve
```

To kill means to hurt which means to have conflict with. Evidently the good matches are the ones with suicides.

A form of irony, incidentally, could be found the same way. But there were no such incidents, at least as described:

```
IRONIC-EVENT is a situation.

Unfortunate-person is a person. Unfortunate-person want a
desired-act - attempt the desired-act [cause an actual-
act]. The desired-act opposite the actual-act.
```

All these examples argue for the following conclusion:

- Simple deduction in the direction of abstraction facilitates some matches.

Many Similarity Measures are Possible

The similarity measure used in the implemented matcher is just the total number of points of evidence exhibited when the parts in two situations are optimally paired. Thus the similarity is a measure of overlap.

Many other authors have considered the question of similarity measurement, although not in the context of the representation used in this paper. In particular, Tversky considers situations in which two objects defined by feature sets are to be compared [18]. He argues persuasively that similarity should be determined not only by the features that correspond, but also by those that do not. For determining the similarity of feature set A to feature set B, he recommends this formula:

$$\text{SIMILARITY}(A,B) = \theta\, f(A \cup B) - \alpha\, f(A - B) - \beta\, f(B - A)$$

For some θ, α, and β

Where f is typically a function that satisfies additivity:

$$f(X \cup Y) = f(X) + f(Y)$$

In this paper similarity is measured between groups of frames rather than feature sets. $A \cup B$ is analogous to the slot-value combinations that are in one or both of the groups of frames; A - B and B - A would be analogous to the slot-value combinations that are in one group of frames but not in the other; f is just a function which counts; θ is 1; and both α and β are 1. If the analogs to A - B and B - A were used with unequal α and β, the measure would be unsymmetric -- one situation would be more similar to a second than the second would be to it.

Matching works on Physical Laws as well as on Plots

Consider the relation between the water pressure and water flow in a pipe. It is possible to describe the kind of thing each is, as well as about how each is related to the other and to the resistance of the pipe, using the same symbol-arrangement conventions that we have been working with. A new vocabulary is needed, however:

> PIPE-LAW is a constraint - dependent-variable pressure-pipe-law - independent-variable flow-pipe-law - multiplier resistance-pipe-law.

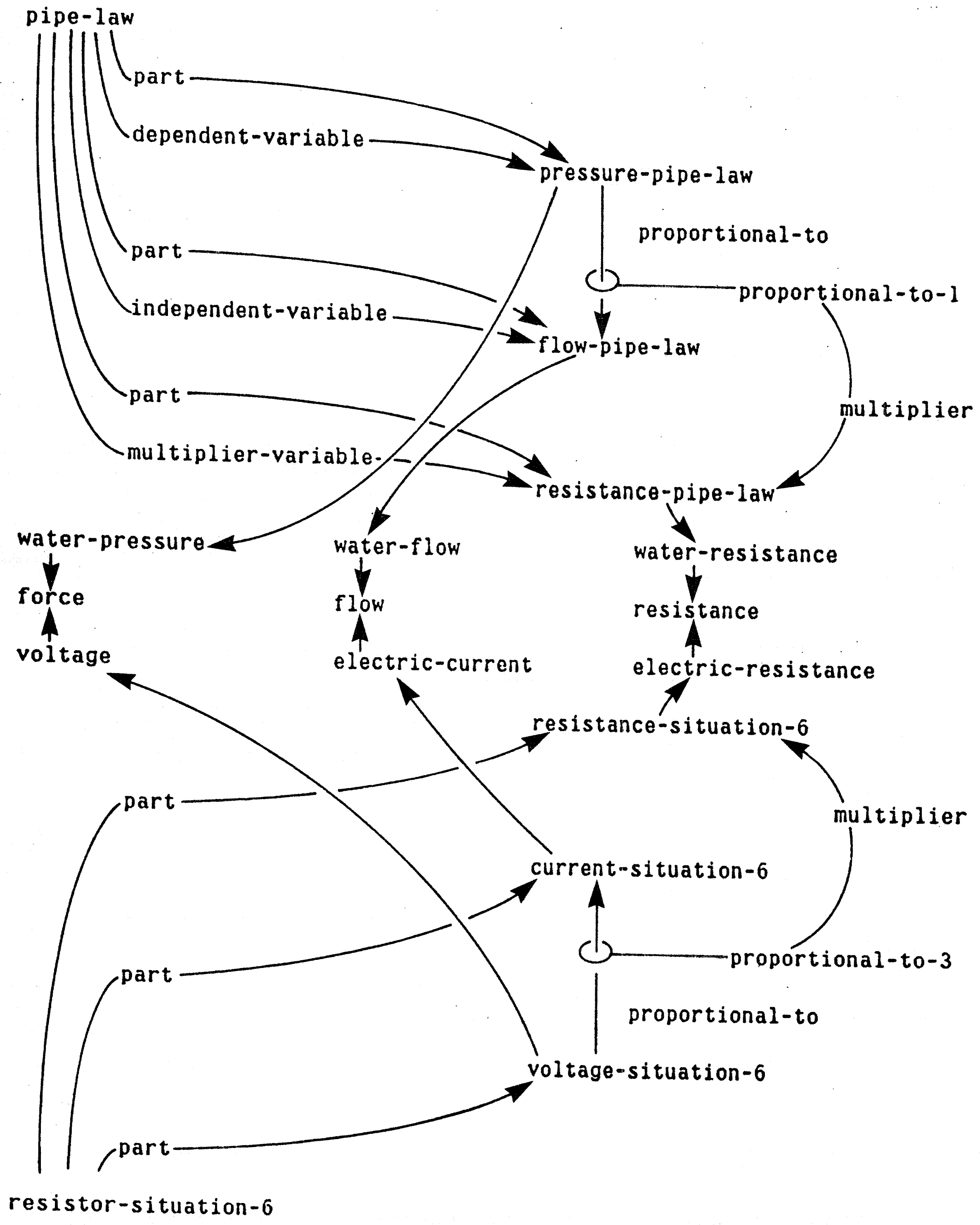


Figure 7: Resistor-situation-6, a situation involving a resistor, can be matched to the general water-pipe description.

and other information to insure match.

Matching Enables Computation in Analogous Domains

So far we have looked at an example in which it was necessary to match the parts of a specific situation against those of a general description within one domain. Now we look at an example in which the specific situation and the general situation are in different domains.

Figure 7 illustrates an example in which it is desired to calculate the voltage across a resistor. As shown, the parts of the specific resistor situation are described by classification, property, and other information, but it is assumed that there is no general description of resistor situations and no procedure for computing the voltage across a resistor.

If a teacher announces that resistors are like water pipes, however, the voltage can be computed. It is only necessary to match the parts of the specific resistor situation with those of the general water pipe situation and to use the procedure for calculating water pressure on the resistor-situation parts identified as the INDEPENDENT-VARIABLE and the MULTIPLIER. As it stands, there is enough evidence to insure the correct match of the parts, although the strength of the match is naturally weaker than it was when the specific situation and the general situation both involved water.

Specific Laws can be Learned

It is possible to generate a general description of resistor situations once two specific situations have been analyzed through the water-pipe analogy. The idea is simple: copy the classification, property, and other information that is common to both specific situations. Figure 8 illustrates the result, given two analyzed resistor situations.

With the new description of resistor situations, the procedure that computes water pressure can be applied to resistor situations with more confidence, since resistor situations will match the general resistor-situation description better than they will match the general water-pipe situation.

The newly generated, general description of resistor situations, together with the same procedure that worked with water pipes, constitute Ohm's law. Generating the description constitutes a kind of learning. Let us recapitulate the steps:

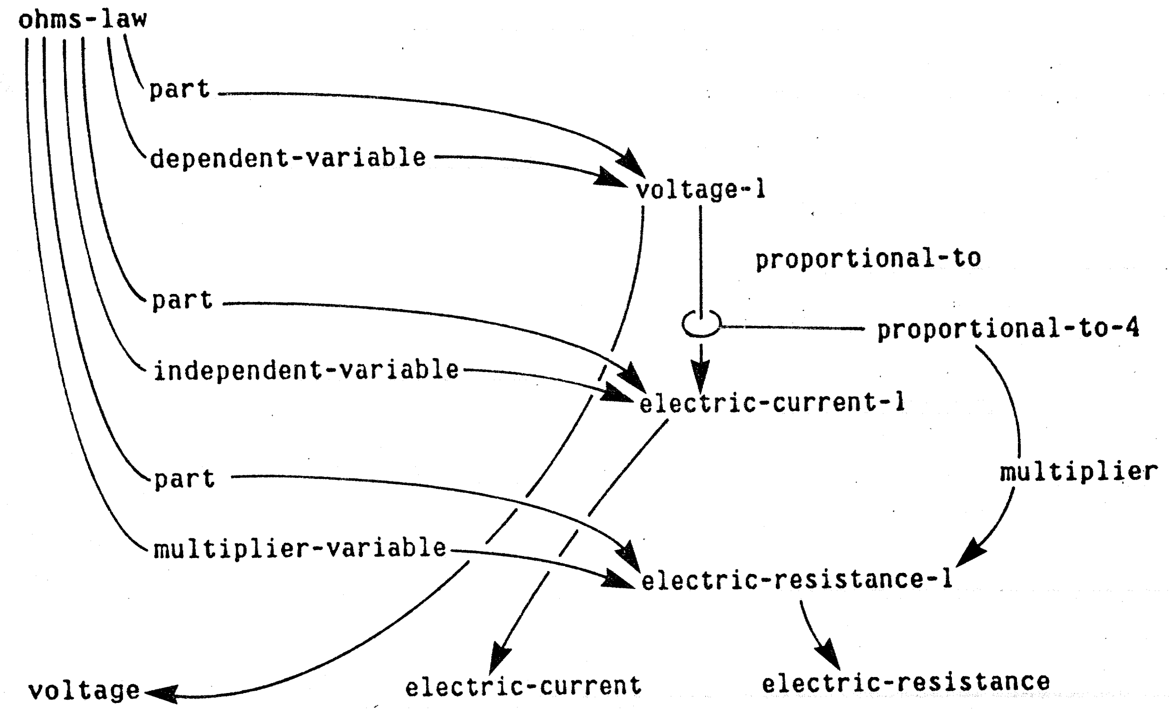


Figure 8: The descriptive part of Ohm's law, as abstracted from two situations analyzed through the water-pipe analogy.

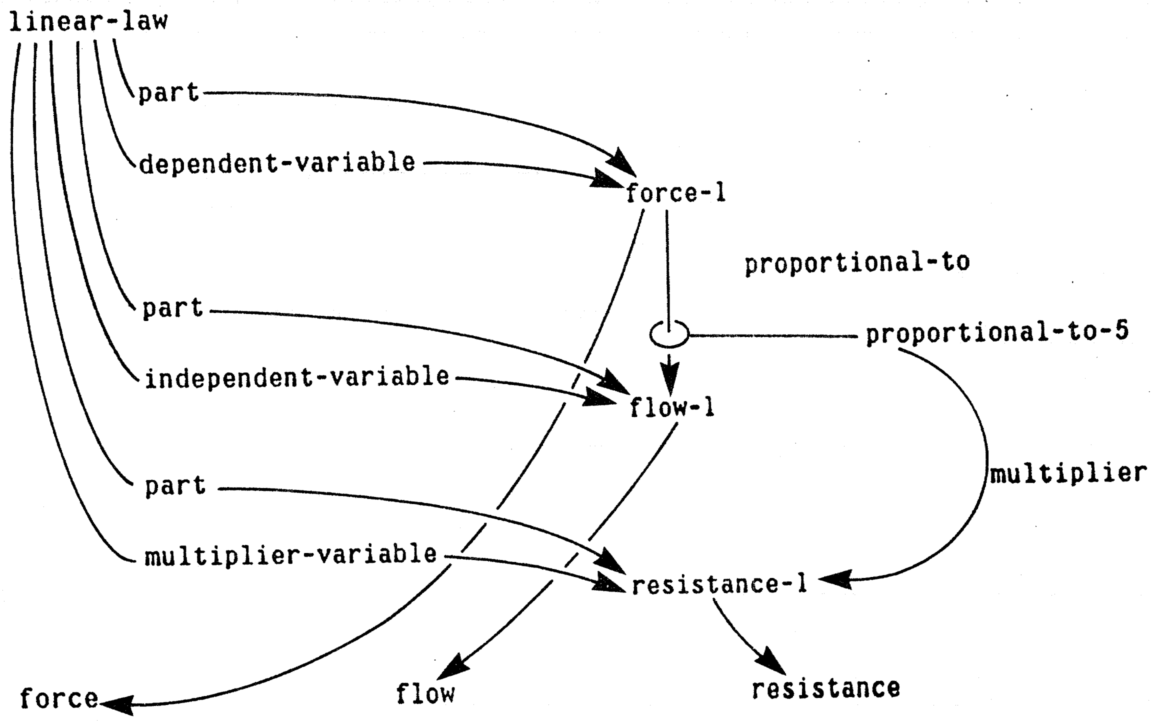


Figure 9: The descriptive part of the general linear law, as abstracted from the water-pipe law and Ohm's law.

- Two specific situations are analyzed using a general situation description in another domain.
- The common parts of the specific situation descriptions determine a general description in the same domain.
- The procedural part of the law in the other domain is brought over without change.

It is possible to push the learning further back, acquiring the procedural part of the law by examples, rather than by copying, but work on this is not yet solid enough to report.

General Laws can be Learned

Specific laws are learned by jointly analyzing two worked-out situations in one domain. General laws are learned by jointly analyzing two specific laws in different domains. The learning procedure is the same.

Figure 9 illustrates the result, given the water-pipe law and Ohm's law. Again the key step is to form a new description out of what is common to both given descriptions. The procedural part, noted to be the same in both specific laws, is adopted without change.

REASONING USING ANALOGY-DRIVEN SITUATION ANALYSIS

How is it possible to know if some relation holds in one situation, given that the situation is analogous to another, well-understood situation? The answer is that the constraint relations in the well-understood situation suggest the right relations to check and the right questions to ask. Here we consider only special cases of constraint involving cause. The key assumption is that the cause structure of a well-understood situation is likely to say something about the possible cause structure in a situation to be analyzed.

The Cause Relations in Situations make Common-sense Reasoning Possible

To be more precise, the following steps are taken in the current implementation when a user asks about some relation:

- First, the relation in question may actually be in the situation being analyzed. If so, no further action is needed.
- Second, in the well-understood situation, the relation in question may be caused by a person. If so, ask if the corresponding person in the situation being analyzed causes the relation.

- Third, in the well-understood situation, the relation in question may be caused by another relation. If so, try to justify that other relation in the situation being analyzed by recursion.

If there are several causes for a relation in the well-understood situation, all must be verified. If any PREVENTED-BY relations are encountered along the way, they are investigated as follows:

- If, in the well-understood situation, a person prevents a relation, ask if the corresponding person in the situation being analyzed prevents the relation.

- If, in the well-understood situation, a relation is prevented by another relation, try to justify that other relation in the situation being analyzed by recursion.

Success in pursuing any of a relation's PREVENTED-BY values means the corresponding relation in the given situation cannot be established.

To see how all this works out, consider the following stripped-down version of *Hamlet*:

```
Ha is a story.

Ghost is a king.  Claudius is a man - marry Gertrude.
Gertrude is a woman.  Hamlet is a man - has-property loyal.
```

As it stands, there is barely enough said to do an unambiguous match against *Macbeth*, but given that a user asks questions, it may still make sense to use the analogy. In particular, the following traces what can be done given the cause connections in *Macbeth*, shown in figure 10, together with a question about whether Claudius dies:

```
(CHECK 'CLAUDIUS 'HAS-PROPERTY 'DEAD IN HA USING MA)

Does GERTRUDE cause the MURDER slot of CLAUDIUS to have
GHOST in it?

> NO (meaning the user does not know)

Does CLAUDIUS DESIRE [ CLAUDIUS A-KIND-OF KING ] ?

> YES (assumed for illustration)

Evidently CLAUDIUS DESIRE [ CLAUDIUS A-KIND-OF KING ] .

Evidently there is sufficient CAUSE for the MURDER slot of
CLAUDIUS to have GHOST in it.
```

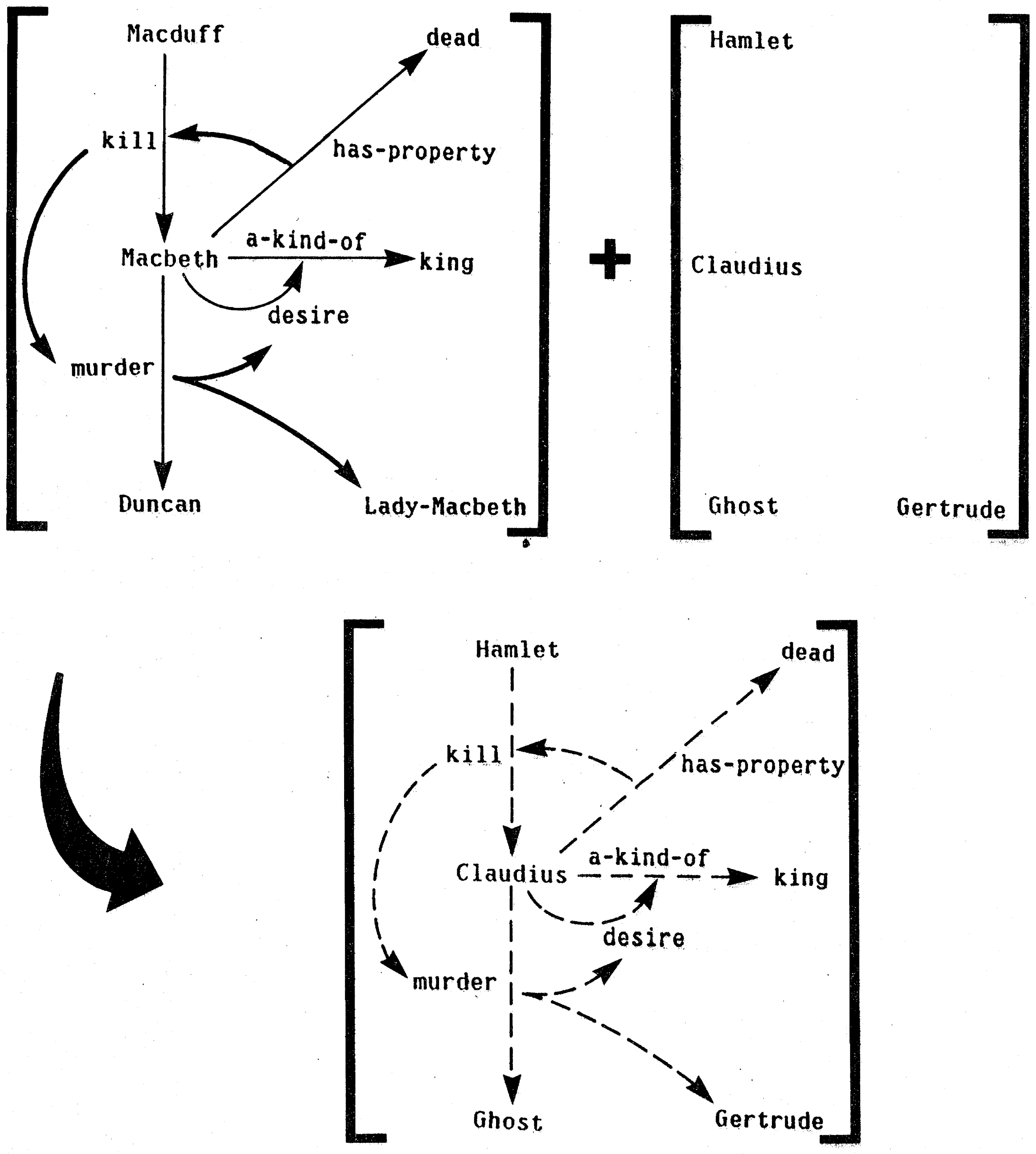


Figure 10: The causal connections in Macbeth. If Macbeth is matched with a situation having no connections, then asking if the Macbeth person dies leads to asking questions determined by the causes. This is done by overlaying the CAUSE connections in Macbeth on those in the new situation, a stripped down version of Hamlet here. The first question is about Gertrude and the cause of a murder. If the answer is unknown, the second question is about the Claudius and whether he desires to be king.

```
Evidently there is sufficient CAUSE for the KILL slot of
HAMLET to have CLAUDIUS in it.

Evidently there is sufficient CAUSE for the HAS-PROPERTY
slot of CLAUDIUS to have DEAD in it.

[ CLAUDIUS HAS-PROPERTY DEAD ] is verified by the precedent
in MA
```

It makes sense. By using analogy, a kind of common-sense reasoning is exhibited. Initial experiments indicate that the same ideas work in law where dealing with legal precedent seems like a combination of situation identification and analogy-driven situation analysis.

FINDING ANALOGIES BY CLASSIFICATION-EXPLOITING HYPOTHESIZING

Before two situations can be matched, it is necessary to find the remembered situation that is most relevant to the situation under analysis. Two mechanisms for such hypothesizing have been implemented: one mechanism uses a situation to guide a search through a network of possibilities related by SIMILAR-TO; and another mechanism uses a situation to probe into an A-KIND-OF tree augmented with indexing information.

The mechanism using SIMILAR-TO relations was devised by Winston [21] and developed by Minsky [10]. A straightforward implementation did not seem particularly illuminating. To be sure, a similarity net was constructed from the ten experimental situations, but ten is not enough to demonstrate anything. With a data base of the size used, only weak illustrations were possible.

Using a Situation to Probe into an A-KIND-OF Tree

The mechanism that uses an A-KIND-OF tree is more important, for it is suggestive of how information retrieval might be done.

To begin, a list is made of everything a situation's parts are a kind of. Each element of this list is checked to see if it has an APPEARS-IN slot. Values in such slots are used to hypothesize situations that are likely to match well.

APPEARS-IN slots are filled as situations are remembered. Everything above a situation's parts in the A-KIND-OF tree has the situation placed in its APPEARS-IN slot.

Suppose, for example, that we start with a *tabula rasa* and supply the following:

Prince is a man. Man is a person. Princess is a woman. Woman is a person. Boy is a man. Girl is a woman.

CI is a story. Charming is a prince. Cinderella is a princess. <Plus other relations involving Charming and Cinderella>

SN is a story. Snow-White is a princess. <Plus other relations involving Snow White>

The effect is to place CI in the APPEARS-IN slot of PRINCE, MAN, PERSON, PRINCESS, and WOMAN, and to place SN in the APPEARS-IN slot of PRINCESS, WOMAN, and PERSON.

When looking for situations to match a given situation, the A-KIND-OF tree standing above each part is searched for APPEARS-IN slots. These slots then vote for the remembered situations they contain. In the actual implementation, the voting is weighted in two ways:

- Each encountered instance of APPEARS-IN casts votes in inverse proportion to the number of values present. This reduces the weight of APPEARS-IN votes coming from frequently occurring classes.

- Each encountered instance of APPEARS-IN casts votes in proportion to the number of slots in the part of the situation associated with it. This increases the weight of APPEARS-IN votes coming from the more important parts of the situation.

Suppose, for example, that the following situation is given:

RJ is a situation. Romeo is a boy. Juliet is a girl. <Plus other facts about Romeo and Juliet>

Suppose Romeo has six slots and Juliet has four. Romeo is connected to MAN, which has only CI in its APPEARS-IN slot, and to PERSON, which has both CI and SN. Thus Romeo's contribution is nine votes for CI and three for SN. Juliet's is connected to WOMAN and PERSON, each of which has both CI and SN in the APPEARS-IN slot. Thus Juliet's contribution is four votes for each. CI beats SN, 13 to 7.

Using relations for indexing and retrieving, incidentally, requires no further machinery, since relation-describing frames can be plot parts just like other things. The following would do this for an instance of KISS used in the CINDERELLA frame, insuring that CI would end up in the APPEARS-IN slot of KISS.

{Cinderella kiss Charming} is a kiss - part-of Ci.

It also may be reasonable to index and retrieve on pairs of situation parts. As it stands, the implementation allows a search for plausible situations that have parts that are a kind of PRINCE and a kind of MARRY. Using pairs for indexing and retrieval would enable a system to look for, say, situations in which a particular prince is married to someone. This has not received much thought yet.

The current method was used on each of the experimental situations. The situation used as a probe was the unambiguous first hypothesis except when the probe was one of the two Ibsen plays. This is not strange. The Ibsen people are just people, men and women, not princes, generals, or other distinguishing things.

It is not Clear if Classification-exploiting Hypothesizing Scales Well

In the end, it must be determined if the A-KIND-OF-based identification method is robust enough to be useful when the number of situations is increased to a practical size.

The previously mentioned work of Rosch *et al.* may be relevant [16]. They argue that people tend to recognize and specify concepts at the so-called basic level first. This might mean that the most useful situation-hypothesizing information would be at that level. Plainly the higher level classifications would not be useful because the situations involving a class like PERSON would be too numerous to provide useful constraint or even to record.

If there are only, say, a thousand basic classes, and if each situation uses, say, four objects and relations in a prominent way, then there could be on the order of 10^{10} situations with different combinations. There would be plenty of room for an expert to know a lot. But of course the space of combinations certainly is filled unevenly by the situations that are useful. A better analysis or some experimentation is needed, therefore, to see if some form of A-KIND-OF-based identification will work in practical situations. Obvious things to think about are context-constrained use of the APPEARS-IN slots and analysis-time discovery of useful classification information.

OTHER QUESTIONS

It is not possible to discuss all that has been done in a palatable-length paper. Among the details left out are explanations of the following:

- The form of the representation when reduced to list structure.
- The form of the English-like interface when reduced to a program.
- The method by which it possible to improve match in difficult circumstances by automatically generating questions to be answered by users or by some deduction system.

- The experimental results exhibited by the matcher when dealing with variously described resistor and water-pipe situations.

- The method by which it possible to learn the procedural part of laws.

- The method by which it possible to learn demons, together with the arguments suggesting that the use of a demon is like the exploitation of a miniature analogy.

- The experimental results of reasoning using legal situations.

- The way the parts of one group of situations can be matched to the parts of another group of situations. (This is required for identifying the analogy between the basic constraints of the electrical world and those in the mechanical world. The resistor, capacitor, and inductance laws constitute the first group, and the damper, spring, and momentum laws, the second.)

All of these points are discussed elsewhere [22].

CONCLUSION: SIMPLE MECHANISMS HAVE PROMISE

This paper is about a set of ideas that enable certain types of learning and reasoning to take place by analogy in domains that satisfy certain restrictions, namely:

- Symbolic sufficiency.

- Description-determined similarity.

- Constraint-determined importance.

- Historical continuity.

The learning and reasoning ideas developed for such domains are these:

- Extensible-relations representation.

- Importance-dominated matching.

- Analogy-driven constraint learning.

- Analogy-driven reasoning.

- Classification-exploiting hypothesizing.

It is now clear that simple situations can be analyzed by attacking them with the analogy process using stored situations ranging from small, demon-sized incidents to *Macbeth*-sized plots. Once analyzed, a situation becomes eligible to be remembered for use in analyzing newer, perhaps bigger situations. Experience makes an analogy-making system smarter, or at least more experienced.

RELATED WORK

In some ways, this work has roots in my previous learning systems. The first of these is known as the ARCH system [20], and the second, FOX [21], both being identified by the typical things involved in the learning.

In addition, many of the ideas in this paper were influenced by other precedents. In particular, Schank [17], Wilks [19], and others stimulated work on the problem of how thinking is determined by stored experience and emphasized the importance of cause relations. Evans [3] broke ground with his work on the geometric analogy problem. Moore and Newell [11] suggested something quite reminiscent of the way laws are learned. Minsky [10] and Goldstein and Roberts [14, 15] worked out key representation ideas. Martin [8] and Rieger [13] demonstrated the utility of hard work on details of vocabulary. And Lenat's success with his mathematical discovery system provoked renewed interest in the entire area of computer learning [7].

Also, since the experiments reported in this paper were done, a better, more natural interface has been designed and implemented by Katz [6]. Similarly, a more efficient matcher has been designed and implemented by Brotsky [1]. Conversion to Katz' interface and Brotsky's matcher is underway.

ACKNOWLEDGMENTS

Early versions of this paper were improved by discussions with Peter Hart, William B. Martin, Saul Amarel, Richard Brown, Jim Stansfield, Boris Katz, Charles Rich, Mark Jeffery, Glen Iba, Margareta Hornell, and Karen Prendergast. Later versions were improved by discussions with Robert Berwick, Michael Brady, and other participants in the Artificial Intelligence Laboratory's Learning Seminar. The drawings are by Karen Prendergast.

REFERENCES

1. Brotsky, Daniel, Efficient Matching of Situations. M.I.T. Artificial Intelligence Laboratory Memorandum in preparation.

2. Carroll, John B., Daves, Peter, and Richmond, Barry. *Word Frequency Book*, Houghton-Mifflin and American Heritage Publishing Companies, New York, 1971.

3. Evans, Thomas G. A heuristic program to solve geometric analogy problems. Ph.D. thesis, in *Semantic Information Processing*, edited by Marvin Minsky, The M.I.T. Press, Cambridge, Massachusetts, 1968.

4. Filmore, C. J. The case for case. In *Universals in Linguistic Theory*, edited by E. Bach and R. Harms, Holt, Rinehart, and Winston, New York, 1968.

5. Givon, Talmy, Cause and Control: on the Semantics of Interpersonal Manipulation. In *Syntax and Semantics*, Volume IV, edited by John Kimball, Academic Press, 1975.

6. Katz, Boris, Parsing Simple English Sentences for a Learning Program, M.I.T. Artificial Intelligence Laboratory Memorandum in preparation.

7. Lenat, Douglas. AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search. Ph.D. thesis, in *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, 1979.

8. Martin, William A. Philosophical foundations for a linguistically oriented semantic network. In preparation.

9. Meldman, J. A preliminary study in computer-aided legal analysis. Ph.D. thesis, M.I.T. Laboratory for Computer Science Technical Report No. MAC-TR-157 (November 1975).

10. Minsky, Marvin. A framework for representing knowledge. In *The Psychology of Computer Vision*, edited by Patrick Henry Winston, McGraw-Hill Book Company, New York, 1975.

11. Moore, J. and Newell, Allen. How can Merlin understand? In *Knowledge and Cognition*, edited by L. Gregg, Lawrence Erlbaum Associates, Potomac, Maryland, 1974.

12. Ogden, C. K. *Basic English: International Second Language*, Harcourt, Brace, and World, New York, 1968.

13. Rieger, Chuck. The commonsense algorithm as a basis for computer

models of human memory, inference, belief, and contextual language comprehension. University of Maryland, College Park, Department of Computer Science Technical Report No. 373 (1975).

14. Roberts, R. Bruce and Goldstein, Ira P. The FRL primer. M.I.T. Artificial Intelligence Laboratory Memo No. 408 (July 1977).

15. Roberts, R. Bruce and Goldstein, Ira P. The FRL manual. M.I.T. Artificial Intelligence Laboratory Memo No. 409 (June 1977).

16. Rosch, Eleanor, Mervis, Carolyn B., Gray, Wayne D., Johnson, David M., and Boyes-Braem, Penny. Basic objects in natural categories. *Cognitive Psychology 8* (1976).

17. Schank, Roger C. *Conceptual Information Processing*, North-Holland Publishing Company, New York, 1975.

18. Tversky, Amos. Features of similarity. *Psychological Review 84*, 4 (July 1977).

19. Wilks, Yorick A. *Grammar, Meaning, and the Machine Analysis of Language*, Routledge and Kegan Paul, London, 1972.

20. Winston, Patrick Henry. Learning Structural Descriptions from Examples. Ph.D. thesis, in *The Psychology of Computer Vision*, edited by Patrick Henry Winston, McGraw-Hill Book Company, New York, 1975.

21. Winston, Patrick Henry. Learning by Creating and Justifying Transfer Frames. *Artificial Intelligence 10*, (1978), 147-172.

22. Winston, Patrick Henry. Learning and Reasoning by Analogy: the Details. Formerly titled Learning by Understanding Analogies. M.I.T. Artificial Intelligence Laboratory Memo No. 520, April 1979.

APPENDIX 1: NOTES

Note 1: Frames can be used to Implement Extensible-relation Representation

A frame can be thought of as a generalized property list. In the local MIT vernacular, the properties of a frame are called slots. Each slot can have a number of subdivisions called facets. One often-used facet is the value facet. Translating a property-list description into a frame description means stuffing property values into value facets, leaving all other facets unused. Here are two sample frames as they appear in the LISP-based implementation:

```
(V (PROPORTIONAL-TO (I)))

(CINDERELLA (KISS (CHARMING)))
```

The frames are CINDERELLA and V. The slots are KISS and PROPORTIONAL-TO. And the values are CHARMING and I. Each value in a frame can be accompanied by comments. Comments are shown by way of additional nesting.

```
(V (PROPORTIONAL-TO (I (COMMENT (PROPORTIONAL-TO-62)))))

(CINDERELLA (KISS (CHARMING (COMMENT (KISS-1)))))
```

Comments assume importance because they provide a means for breaking out of the too-strong orientation of property lists toward binary relations. In the examples, PROPORTIONAL-TO-62 and KISS-1 are frames that further describe frame-slot-value combinations:

```
(PROPORTIONAL-TO-62 (MULTIPLIER (R)))

(KISS-1 (TIME (END)))
```

Thus it is possible to say a lot about a relation. For example, translating the sentence "Prince Charming found Cinderella with her glass shoe," into frames produces the following:

```
(CHARMING (FIND (CINDERELLA (COMMENT (FIND-1)))))

(FIND-1 (INSTRUMENT (SHOE-1)))

(SHOE-1 (A-KIND-OF (SHOE) (PROP))
        (RAW-MATERIAL (GLASS))
        (OWNER (CINDERELLA)))
```

Certainly the agent and the object are emphasized by this way of representing acts just as they are in ordinary active English sentences by position constraint and the lack of case-indicating prepositions. Still, the other participants in an act, such as the instrument, can be easily noted as necessary in the act's comment frame.

The idea of representing relations as commented frame-slot-value combinations was suggested to me by Bruce Roberts. I call it a kind of extensible-slot representation. Several alternatives were rejected in favor of it:

- All slots specifying relations could be replaced by a universal slot with a name like RELATION-SPECIFYING-FRAME. All relations would be described by frames enumerated in this universal slot:

```
(CHARMING (RELATION-SPECIFYING-FRAME (FIND-1)))

(FIND-1 (A-KIND-OF FIND)
        (AGENT (CHARMING))
        (OBJECT (CINDERELLA))
        (INSTRUMENT (SHOE-1)))
```

Such a representation is less perspicuous because it is not clear what Prince Charming is related to or how by just looking at the CHARMING frame. Moreover the representation seems to make it more difficult to think how one group of parts could be matched with another.

- Relation names could be replaced by the names of relation-describing frames:

```
(CHARMING (FIND-1 (CINDERELLA)))

(FIND-1 (A-KIND-OF FIND)
        (AGENT (CHARMING))
        (OBJECT (CINDERELLA))
        (INSTRUMENT (SHOE-1)))
```

This representation is slightly more perspicuous than the universal slot solution since it is possible to see who Prince Charming is related to by looking in the CHARMING frame. Still, it is not possible to see how the Prince is related to Cinderella without going to the FIND-1 frame. Similarly, matching still seems difficult to think about.

- Further description of a relation could be captured by using facets other than the value facet:

```
(CHARMING (FIND (VALUE (CINDERELLA))
                (AGENT (CHARMING))
                (OBJECT (CINDERELLA))
                (INSTRUMENT (SHOE-1))))
```

Unfortunately anything in the other facets would normally apply to all values in the value facet.

In contrast, using extensible slot representation is perspicuous, it does not interfere with matching, and it leaves the facets alone.

Note 2: An Input Program

The following programs define the English-like input interface used in this work. FPUTV+ places a frame-slot-value combination and activates all demons associated with the slot. FGETC! gets the comment associated with a frame-slot-value combination, making one if necessary.

```
(defun major () (let ((subordinate nil)
                      (lst nil))
                     (clause)))

(defun subordinate () (let ((subordinate t))
                           (clause)))

(defun clause ()
 (prog (agent act object case filler)
       getagent
       (word)
       (cond ((and (eq lst '{) subordinate) (go getagent))
             ((eq lst '{) (setq agent (subordinate)))
             ((not (atom lst))
              (print (eval lst)) (terpri)
              (setq count -1) (go getagent))
             ((eq 2nd '^) (setq agent (possessive)))
             ((eq lst 'eof) (return nil))
             (t (setq agent (noungroup lst))))
       getact
       (cond ((memq (infinitive (word)) '(is be))
              (setq act 'ako) (go getclasses))
             ((badwordp getact) (go terminate))
             ((memq lst '(to has)) (go getact))
             ((memq (setq act (infinitive lst))
                    '(cause persuade force prevent dissuade))
              (setq object (subordinate)) (go getobject1))
             (t (go getobject)))
getobject
(cond ((memq (word) '({ that)) (setq object (subordinate)))
      ((memq lst '(} /.)) (go terminate))
      ((eq lst '-) (go getact))
      ((eq lst 'and) (go getobject))
      ((memq lst '([ with)) (go getcase))
      ((memq (setq lst (infinitive lst))
             '(cause persuade force prevent dissuade))
       (setq case lst) (setq filler (subordinate)) (go getfillerl))
      ((memq lst '(because since))
       (setq case 'caused-by)
       (setq filler (subordinate)) (go getfillerl))
      ((badwordp getobject) (go terminate))
      (t (setq object (noungroup lst))))
getobjectl
(fputv+ agent act object)
(go getobject)
getclasses
(cond ((memq (word) '(a an and the)) (go getclasses))
      ((memq lst '(} /.)) (go terminate))
      ((eq lst '-) (go getact))
      ((badwordp getclasses) (go terminate))
      (t (setq object lst)))
(fputv+ agent act object)
(go getclasses)
getcase
(cond ((eq (word) 'has) (go getcase))
      (t (setq case lst)))
getfiller
(cond ((eq (word) '{) (setq filler (subordinate)))
      ((eq lst '-) (go getcase))
      ((eq lst 'and) (go getfiller))
      ((eq lst ']) (go getobject))
      ((memq lst '(} /.)) (go terminate))
      ((badwordp getfiller) (go terminate))
      (t (setq filler (noungroup lst))))
getfillerl
(fputv+ (fgetc! agent act object) case filler)
(go getfiller)
terminate
(return (cond ((and subordinate object)
                (fgetc! agent act object))
               (subordinate (get-comment agent act))
               (t t))))))
```

```
(defun noungroup (x)
       (cond ((eq x 'the) (get-the (word)))
             ((memq x '(a an another)) (get-a (word)))
             (t x)))

(defun infinitive (x)
       (cond ((get x 'infinitive)) (t x)))

(defun possessive ()
       (prog (agent act)
             (setq agent lst)
             getact
             (word)
             (cond ((memq lst '(s ^)) (go getact))
                   (t (return (get-comment agent lst)))
                   ((eq 2nd 'to)
                    (setq act lst) (setq 2nd agent)
                    (return (fgetc! agent act (subordinate)))))))

(defun badwordp fexpr (x)
       (cond ((memq lst '([ ] { } /. -))
              (p '|Unexpected punctuation in state| x
                 '|agent =| agent '|act =| act '|object =| object
                 '|case =| case '|filler =| filler '|lst =| lst)
              (break dosomething))
             (t nil)))

(defun word ()
       (let ((readtable /■readtable))
            (cond ((eq lst '/.) lst)
                  ((eq 2nd 'eof) (setq lst 2nd))
                  (t (setq lst 2nd 2nd (read input 'eof))))
            (cond ((not lst) (word)) (t lst))))

(defun get-a (type)
       (fputv+ type 'instance (genname type)))

(defun get-the (type)
       (get-object type 'instance))

(defun get-comment (agent act)
       (fgetc! agent act (get-object agent act)))
```

```
(defun get-object (frame slot)
       (let ((candidates (fgetv frame slot)))
            (cond ((and (cdr candidates) (not (eq *story* universe)))
                   (setq candidates
                         (mapcan
                          '(lambda (e)
                                   (cond ((fgetvp e 'part-of *story*)
                                          (list e))))
                          candidates))))
            (cond ((null candidates) (setq candidates (fgetv frame slot))))
            (cond ((null candidates)
                   (p '|No object for frame =| frame '|and slot =| slot)))
            (car (last candidates))))
```

Note 3: The Matcher works on Constraining Laws as well as on Plots

This note shows that the matcher works on law-like situations, as well as on plots. The law used for illustration constrains the water pressure and water flow in a pipe. It is described is as follows:

> Water-pressure is a pressure. Pressure is a force. Water-resistance is a resistance. Water-flow is a flow.
>
> PIPE-LAW is a constraint - dependent-variable pressure-pipe-law - independent-variable flow-pipe-law - multiplier-variable resistance-pipe-law. Pressure-pipe-law is a water-pressure. Flow-pipe-law is a water-flow. Resistance-pipe-law is a water-resistance.
>
> Pressure-pipe-law is proportional-to flow-pipe-law [multiplier resistance-pipe-law].

Note that things like *pressure-pipe-law* are used here instead of *a pressure* and *the pressure*. This is simply to make it easier to refer to particular instances of pressures, flows, and multipliers in this explanation.

Matching a specific situation against this pipe law is like matching one plot against another. Consider S1, for example:

> S1 is a situation. Pressure-s1 is a water-pressure. Flow-s1 is a water-flow. Resistance-s1 is a water-resistance.

The result is announced as follows by the matcher:

```
Matching PIPE-LAW and S1
 ......
```

```
Score = 7. versus 10. and 7. -- Best match is 4. better
than next best.
```

The score is entirely dependent on the A-KIND-OF slots. Both pressure-pipe-law and pressure-s1 are a kind of water pressure, pressure, and force. Flow-pipe-law and flow-s1 are a kind of water-flow and flow. And resistance-pipe-law and resistance-s1 are a kind of water-resistance and resistance.

In this first example, no use was made of the fact that water pressure is proportional to water flow. Such relations are critically important, however, since relations between the parts of a situation are often known before their classes are established. Moreover, giving the A-KIND-OF relations seems to be giving away too much.

In the following situation, S2, the parts have no known classes, but it is handled anyway because a relation between the parts involved is known:

S2 is a situation.

Pressure-s2 is proportional-to flow-s2 [multiplier resistance-s2].

The matcher gives the result as follows:

```
Matching PIPE-LAW and S2
 ......

Score = 3. versus 10. and 3. -- Best match is 1. better
than next best.
```

The score now depends entirely on the PROPORTIONAL-TO and MULTIPLIER slots. Pressure-pipe-law and pressure-s2 are proportional to flow-pipe-law and flow-s2. A demon attached to PROPORTIONAL-TO similarly established that flow-pipe-law and flow-s2 are proportional to pressure-pipe-law and pressure-s2. And finally, the comment frames for the PROPORTIONAL-TO relations have resistance-pipe-law and resistance-s2 in their MULTIPLIER slots.

Of course, if both relations and classes are known the results are even more likely to be correct. Since S3 is a situation with both a-kind-of and proportional-to information known, the result is just the sum of the previous results:

```
Matching PIPE-LAW and S3
 ......

Score = 10. versus 10. and 10. -- Best match is 6. better
than next best.
```

Now in fact the match may be done on somewhat less related frame groups. Suppose, for example, that one situation is an electrical one and it is to be matched against the pipe law:

> Voltage is a force. Electric-current is a flow. Electric-resistance is a resistance.
>
> S4 is a situation. Voltage-s4 is a voltage. Current-s4 is a electric-current. Resistance-s4 is a electric-resistance.

The result is:

```
Matching PIPE-LAW and S4
 ......

Score = 3. versus 10. and 6. -- Best match is 2. better
than next best.
```

The score is three for S4 rather than seven, as it was for the parallel S1, because the A-KIND-OF chains leading from the parts of S4 intersect those leading from PIPE-LAW less than those leading from S1. Voltage-s4 is less like pressure-pipe-law than pressure-s1 is, and the same is true for current-s4 and resistance-s4 relative to flow-s1 and resistance-s1.

Note 4: Difficult Situations are Improved by Abstraction

This note shows how matching can be facilitated by using demons to do abstraction. The pipe law and a tenuously related situation are used to illustrate.

Suppose the following is given for PIPE-LAW and a particular situation, P1:

> PIPE-LAW is a constraint. Pressure-pipe-law is a water-pressure. Flow-pipe-law is a water-flow. Resistance-pipe-law is a water-resistance. Pressure-pipe-law is proportional-to flow-pipe-law [multiplier resistance-pipe-law].
>
> P1 is a situation. X-p1 cause y-p1 - determined-by y-p1 [parameter z-p1].

As they stand, these descriptions show no correspondence whatsoever. The frames in P1 have no A-KIND-OF slots at all, and they have no other slots that correspond to slots in the frames of PIPE-LAW.

Still, it seems that something should be done to make the match happen. After all, PRESSURE-PIPE-LAW is known to be a kind of force, and since X-P1 causes something, it also is a kind of force, in a sense. Moreover, PROPORTIONAL-TO is a stronger, more particular, less abstract form of DETERMINED-BY, and there are indications that both RESISTANCE-PIPE-LAW and Z-P1 are both a kind of parameter.

All that is necessary is to reassert the facts with every demon in sight turned on. The demons will then deduce and assert new facts in both the frames of PIPE-LAW and P1. The new facts will create a sort of common ground on which matching can succeed. The particular demons that seem appropriate in this situation and ones like it are defined as follows:

```
(if-added proportional-to (fputv+ frame 'increases-with value))

(if-added increases-with (fputv+ frame 'determined-by value))

(if-added multiplier (fputv+ frame 'parameter value))

(if-added parameter (fputv+ value 'a-kind-of 'parameter))

(if-added cause (fputv+ frame 'a-kind-of 'force))
```

The first three place new relations that are weaker forms of the ones that parent them. In a sense their placement represents a sort of abstraction process. The last two place new items in A-KIND-OF slots. They make simple deductions about what something is a kind of, given what it does.

Now the same descriptions of PIPE-LAW and P1 produce frames that do produce a match:

```
(PRESSURE-PIPE-LAW (A-KIND-OF (WATER-PRESSURE))
                   (PROPORTIONAL-TO (FLOW-PIPE-LAW
                                      (COMMENT (PROPORTIONAL-TO-1))))
                   (INCREASES-WITH (FLOW-PIPE-LAW))
                   (DETERMINED-BY (FLOW-PIPE-LAW)))

(FLOW-PIPE-LAW (A-KIND-OF (WATER-FLOW))
               (PROPORTIONAL-TO (PRESSURE-PIPE-LAW))
               (INCREASES-WITH (PRESSURE-PIPE-LAW))
               (DETERMINED-BY (PRESSURE-PIPE-LAW)))

(RESISTANCE-PIPE-LAW (A-KIND-OF (WATER-RESISTANCE) (PARAMETER)))

(PROPORTIONAL-TO-1 (MULTIPLIER (RESISTANCE-PIPE-LAW))
                   (PARAMETER (RESISTANCE-PIPE-LAW)))

(X-P1 (CAUSE (Y-P1))
      (A-KIND-OF (FORCE))
      (DETERMINED-BY (Y-P1 (COMMENT (DETERMINED-BY-1)))))
```

```
(Y-P1 (DETERMINED-BY (X-P1)))

(Z-P1 (A-KIND-OF (PARAMETER)))

(DETERMINED-BY-1 (PARAMETER (Z-P1)))
```

The actual result produced is as follows:

```
        Matching P1 and PIPE-LAW
         ......

        Score = 4. versus 6. and 16. -- Best match is 1. better
        than next best.
```

Note 5: Match can be Improved by Asking Questions

Suppose particular situation P2 is one in which nothing at all is known about the parts:

P2 is a situation - has part x-p2 y-p2 and z-p2.

What can be done to match P2 against the pipe law, PIPE-LAW? One thing is to ask questions. The implemented program for this starts by asking if the parts of the situation belong to certain of the classes found by tracing out the A-KIND-OF paths originating from the parts of the description to be matched against. An effort is made to suggest things that are as general as possible, while still discriminating. To be precise, the steps in the algorithm are as follows:

Trace out all A-KIND-OF paths starting with the situation's parts. Figure 11 illustrates this. Then note where each A-KIND-OF path first collides with another. Call these places the collision points. In the illustration, these are frames C1, C2, and C3.

Next, pick one of the situation's parts. Follow the A-KIND-OF paths starting from it. Remember any places where these paths run into the paths that start from the situation's parts. Call these places the top points. If none are found by following the A-KIND-OF paths from the situation's parts, use the places where the A-KIND-OF paths from the situation's parts terminate. In the illustration, these top points are frames T1 and T2.

Finally, move down from these top points until the A-KIND-OF tree branches for the last time. This will be at some of the collision points noted before. In the illustration, these are frames C1, C2, and C3 for situation frame X and frames C2 and C3 for Y.

The things to ask about or experiment with are just below the collision points just found. In the illustration, these are frames E1, E2, E3, and E4 for X and frames E2, E3, and E4 for Y.

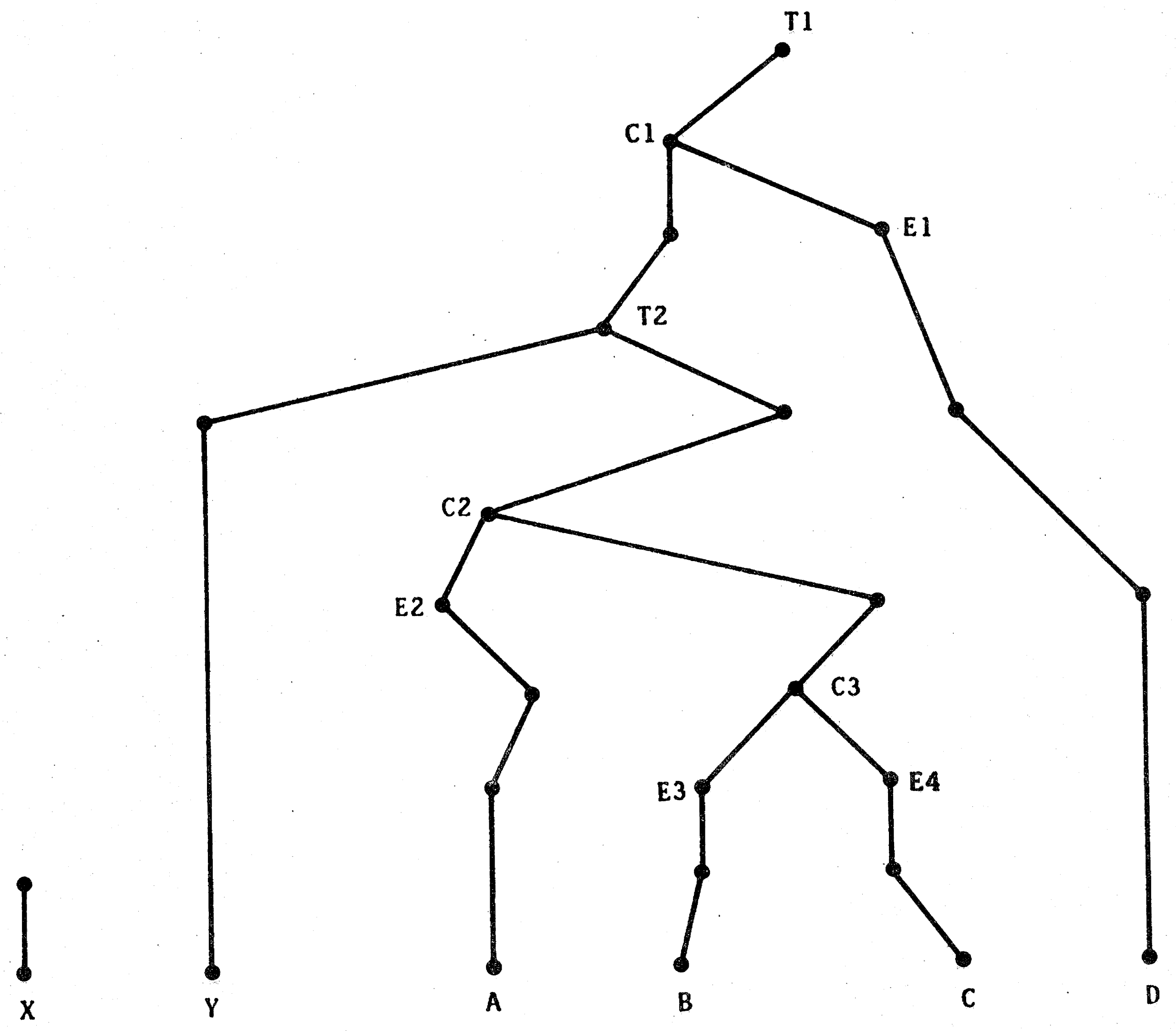


Figure 11: Proposing A-KIND-OF connections so as to improve match. An effort is made to ask the most general question that will help. The assumption is that the most general question is easiest, although it may not be, of course. As shown, questions are asked about what X and Y are. E1, E2, E3, and E4 are proposed for X; E2, E3, and E4 are proposed for Y.

Here is how it works in the simple P2 case:

```
Should I ask about classes?
> YES
Which of the following is X-P2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
```

```
> FORCE
Which of the following is Y-P2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
> FLOW
Which of the following is Z-P2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
> PARAMETER
You made improvements to X-P2 Y-P2 Z-P2
 ......

Score = 3. versus 3. and 16.
-- Best match is  2. better than next best.

((PRESSURE-PIPE-LAW X-P2) (FLOW-PIPE-LAW Y-P2) (RESISTANCE-PIPE-LAW Z-P2))
```

It is easy to think of ways to make this process smarter. For example, if the matcher noted those parts that match well, then the A-KIND-OF-type questions could be limited to those parts that match poorly.

It is also possible that a smarter system would not need to ask the teacher about A-KIND-OF relations. Instead, the smarter system could itself activate procedures or do experiments that answer the classification questions raised. If it is useful to know if X-P2 is a kind of FORCE, the sensible thing to do would be to look first for the sorts of relations forces enter into.

If working with classes does not help, then the implemented system works with relations. The process is simpler: first, find the slots that are used in one situation but not in the other; then, ask if these slots hold anything in any of the parts of the situation.

Working again with P2, the following happens, this time without complete success:

```
Should I ask about classes?
> NO
Then I will ask about relations!
I want to ask if certain slots have
values in any of these:
X-P2 Y-P2 Z-P2
When signaled give a frame and value or nil.
What does PROPORTIONAL-TO hold between?
> (X-P2 Y-P2)
> NIL
What does INCREASES-WITH hold between?
> NIL
What does DETERMINED-BY hold between?
> NIL
What does A-KIND-OF hold between?
> (Z-P2 PARAMETER)
```

```
New instances of these relations were placed:
A-KIND-OF PROPORTIONAL-TO
 ......

Score = 7. versus 7. and 16.
Match is indecisive.
Should I ask about classes?
>
```

Again it is easy to think of more sophisticated ways to decide what to ask about.

Note 6: A Frame-based Production System

The laws involved in the body of this paper have both a description part and a procedure part. This note describes the implemented form of the procedure part, as embedded in frames.

Suppose RULE-PIPE-LAW is a rule specifying that pressure is determined by multiplying resistance times flow if values for them are known. We augment PIPE-LAW by placing RULE-PIPE-LAW in its RULE slot. RULE-PIPE-LAW looks like this:

```
(RULE-PIPE-LAW (IF (IF1) (IF2) (IF3))
               (THEN (THEN1)))
```

The occupants of the IF and THEN slots are ordered lists of frames that are matched against the particular situation at hand. Consider the frame IF1, for example:

```
(IF1 (MATCHES (SITUATION))
     (DEPENDENT-VARIABLE (>X))
     (INDEPENDENT-VARIABLE (>Y))
     (MULTIPLIER-VARIABLE (>C)))
```

The > notation means find the thing that occupies the corresponding slot of the situation frame and shove it into the variable.

After matching the first IF frame, IF1, against the situation and picking up variable values, the rule interpreter proceeds to IF2 and IF3:

```
(IF2 (MATCHES (<C))
     (QUANTITY (>CQ)))

(IF3 (MATCHES (<Y))
     (QUANTITY (>YQ)))
```

The < notation means pull the value out of the variable. Thus the first purpose of the IF frames is to certify that the situation and its related frames have a certain rigid form. The second is to dig certain values out. The values are used by the frames in the THEN slot of the rule.

In the example, the first and only THEN frame is THEN1:

```
(THEN1 (MODIFIES (<X))
       (QUANTITY (= (TIMES <CQ <YQ))))
```

This means that the frame to be modified is the frame that was dug out by the first frame in the IF part of the rule. It is to be modified by placement of a new value in the QUANTITY slot. The = notation means compute the expression that follows.

Note 7: The Production Rules can be Learned

This note shows how to learn the situation-action rules involved in new laws using pairs of analogies or pairs of known laws. There is a simple implemented algorithm based on two assumptions that may or may not be reasonable:

- First, that it is possible to get a situation description that contains only the slots and values that are critical to prediction.

  This may come from a spoon-feeding teacher. Alternatively, it may come from intersecting two versions of a situation that come from a slightly more Socratic teacher.

- Second, that the things that are to end up as predictions are distinguishable.

  For developing constraints, it is assumed that the teacher marks conclusions by placing comment frames that have CONCLUSION in their HAS-PROPERTY slots. For learning rules from situations it is assumed that the things that happen at the end of a situation are as good as conclusions and that they also should be predicted. They are assumed to be marked by comment frames that have END in their TIME slots.

To begin, the algorithm makes a list of the situation and its parts. Potentially each element of the list can lead to an IF frame, a THEN frame, or both. If one of these frames has a slot-value combination that is not marked END or CONCLUSION, then an IF frame is formed and that slot-value combination ends up in it. Similarly, if a frame does have a slot-value combination marked with END or CONCLUSION, a THEN frame is formed.

The first time a situation part is encountered, it is made into a variable with a > symbol. Each subsequent time one is encountered, it is made into a variable with a < symbol.

Note 8: Legal Situations Illustrate the need for Attention to Details of Cause

Several legal situations were recorded to get a feel for what is involved in reasoning by analogy in law. In particular, two items of legal doctrine and two precedent-setting cases were adapted from the minilegal domain used by Jeffery Meldman in his thesis [9]. Here are approximations to the actual input in ordinary English:

Assault: There is an assault if B apprehends contact from A and A intends the apprehension.

Battery: There is a battery if A contacts B and A intends the contact.

Adams versus Zent: Adams intentionally knocked off Zent's glasses. It is held that there is a battery because the facts indicate intentional contact, as required by legal doctrine.

Smith versus Wesson: Smith pointed a rifle at Wesson to frighten him. The rifle was not loaded, and it was therefore harmless. It is held that an assault has taken place even though the rifle was not loaded because Wesson did not know it and had reason to be apprehensive.

Having recorded these, it is possible to deal, by doctrine and precedent, with the following:

Guilty versus Innocent: Guilty wanted to kick Innocent and did.

Villain versus Victim: Villain pointed a pistol at Victim in order to frighten him. The pistol was harmless toy. Villain furthermore knocked off Victim's hat.

Now let us see what the reasoning system thinks about whether these are cases of assault and battery:

```
(CHECK 'GUILTY-VS-INNOCENT 'A-KIND-OF 'ASSAULT
     IN GUILTY-VS-INNOCENT USING ASSAULT)

Does the APPREHEND slot of INNOCENT have [ GUILTY CONTACT
INNOCENT ] in it?

> YES

Does the INTEND slot of GUILTY have [ INNOCENT APPREHEND [
GUILTY CONTACT INNOCENT ] ] in it?
```

```
> YES

Evidently there is sufficient CAUSE for the A-KIND-OF slot
of GUILTY-VS-INNOCENT to have ASSAULT in it.

[ GUILTY-VS-INNOCENT A-KIND-OF ASSAULT ] in GUILTY-VS-
INNOCENT is verified by the precedent in ASSAULT1
```

The situation, as described, left holes that had to be filled by asking. To establish battery, however, only one question need be asked since KICK establishes CONTACT by way of a demon:

```
(CHECK 'GUILTY-VS-INNOCENT 'A-KIND-OF 'BATTERY
       IN GUILTY-VS-INNOCENT USING BATTERY)

It is known that the CONTACT slot of GUILTY has INNOCENT in
it.  Does the INTEND slot of GUILTY have [ GUILTY CONTACT
INNOCENT ] in it?

> YES

Evidently there is sufficient CAUSE for the A-KIND-OF slot
of GUILTY-VS-INNOCENT to have BATTERY in it.

[ GUILTY-VS-INNOCENT A-KIND-OF BATTERY ] in GUILTY-VS-
INNOCENT is verified by the precedent in BATTERY1
```

Now consider the harder case. Anticipating failure with direct match against assault and battery, we try to deal instead with the questions that would have been asked, starting with contact:

```
(CHECK 'VILLAIN 'CONTACT 'VICTIM
       IN VILLAIN-VS-VICTIM USING SMITH-VS-WESSON)

It is known that the KNOCK-OFF slot of VILLAIN has HAT-VV
in it.  I see HAT-VV is a kind of CLOTHING

Evidently there is sufficient CAUSE for the CONTACT slot of
VILLAIN to have VICTIM in it.

[ VILLAIN CONTACT VICTIM ] in VILLAIN-VS-VICTIM is verified
by the precedent in ADAMS-VS-ZENT
```

Next we deal with intention:

```
(CHECK 'VILLAIN 'INTEND 'CONTACT-VV
       IN VILLAIN-VS-VICTIM USING SMITH-VS-WESSON)
```

Does the INTEND slot of VILLAIN have [VILLAIN KNOCK-OFF HAT-VV] in it?

> YES

Evidently there is sufficient CAUSE for the INTEND slot of VILLAIN to have [VILLAIN CONTACT VICTIM] in it.

[VILLAIN INTEND [VILLAIN CONTACT VICTIM]] in VILLAIN-VS-VICTIM is verified by the precedent in ADAMS-VS-ZENT

Next we deal with apprehension:

```
(CHECK 'VICTIM 'APPREHEND 'CONTACT-VV
       IN VILLAIN-VS-VICTIM USING ADAMS-VS-ZENT)
```

Does the NOT-KNOW slot of VICTIM have [PISTOL HAS-PROPERTY HARMLESS] in it?

> YES (this, perhaps, is the most interesting question)

Evidently the KNOW slot of VICTIM is PREVENTED FROM having [PISTOL HAS-PROPERTY HARMLESS] in it. It is known that the FRIGHTEN slot of VILLAIN has VICTIM in it.

Evidently there is sufficient CAUSE for the APPREHEND slot of VICTIM to have [VILLAIN CONTACT VICTIM] in it.

[VICTIM APPREHEND [VILLAIN CONTACT VICTIM]] in VILLAIN-VS-VICTIM is verified by the precedent in SMITH-VS-WESSON

Finally, we see if the apprehension was intended:

```
(CHECK 'VILLAIN 'INTEND 'APPREHEND-VV
       IN VILLAIN-VS-VICTIM USING ADAMS-VS-ZENT)
```

Does the INTEND slot of VILLAIN have [VILLAIN FRIGHTEN VICTIM] in it?

> YES

Evidently there is sufficient CAUSE for the INTEND slot of VILLAIN to have [VICTIM APPREHEND [VILLAIN CONTACT VICTIM]] in it.

[VILLAIN INTEND [VICTIM APPREHEND [VILLAIN CONTACT

VICTIM]]] in VILLAIN- VS-VICTIM is verified by the precedent in SMITH-VS-WESSON

Having worked on the facts to make new, more abstract facts by precedent, it is possible to return to the original questions:

```
(CHECK 'VILLAIN-VS-VICTIM 'A-KIND-OF 'ASSAULT
       IN VILLAIN-VS-VICTIM USING ASSAULT)
```

It is known that the APPREHEND slot of VICTIM has CONTACT-VV in it. It is known that the INTEND slot of VILLAIN has APPREHEND-VV in it.

Evidently there is sufficient CAUSE for the A-KIND-OF slot of VILLAIN-VS-VICTIM to have ASSAULT in it.

[VILLAIN-VS-VICTIM A-KIND-OF ASSAULT] in VILLAIN-VS-VICTIM is verified by the precedent in ASSAULT1

```
(CHECK 'VILLAIN-VS-VICTIM 'A-KIND-OF 'BATTERY
       IN VILLAIN-VS-VICTIM USING BATTERY)
```

It is known that the CONTACT slot of VILLAIN has VICTIM in it. It is known that the INTEND slot of VILLAIN has CONTACT-VV in it.

Evidently there is sufficient CAUSE for the A-KIND-OF slot of VILLAIN-VS-VICTIM to have BATTERY in it.

[VILLAIN-VS-VICTIM A-KIND-OF BATTERY] in VILLAIN-VS-VICTIM is verified by the precedent in BATTERY1

The law case of Villain versus Victim illustrates several particularly suggestive ideas. First that A-KIND-OF relations can be the result of a successful match. Second, that a situation may be analyzed by way of analogy with several already known situations. And third, that the result is a situation description that itself may become a useful part of the general knowledge store after analysis augments it with cause links.

Note 9: Matching one Situation agains a Short List of Possibilities

Let us look at a matching scenario in which a short description is to be matched against *Macbeth, Hamlet, Othello, Julius Caesar*, and *The Taming of the Shrew*. The scenario begins with a statement that two persons are involved. We will see identification become sharper, as illustrated by figure 12, as more is specified.

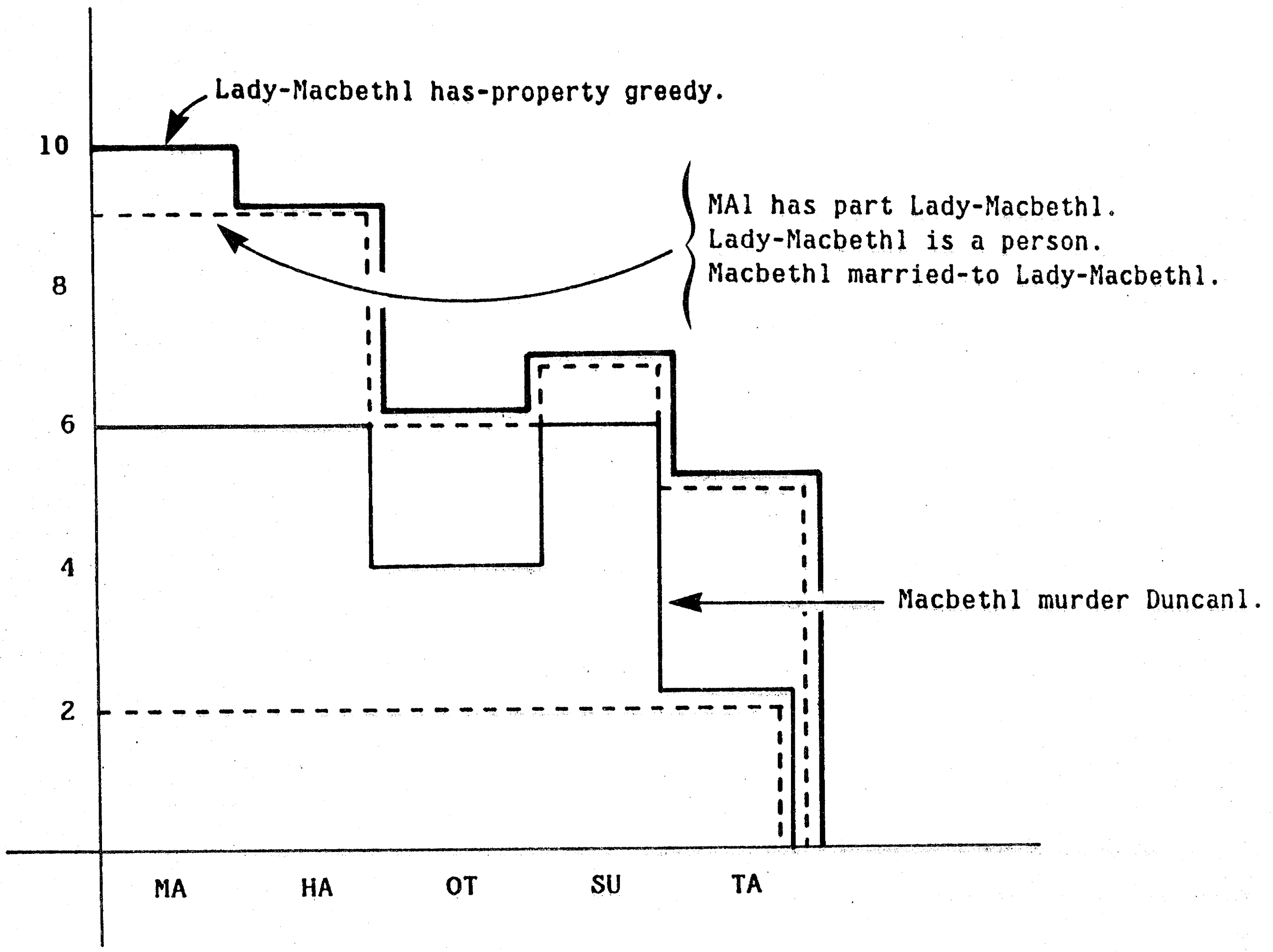


Figure 12: Identification of a situation at hand against several possibilities sharpens as more is known. The curve rises as more facts are given.

MA1 is a story.

Macbeth1 is a person. Duncan1 is a person.

The identification algorithm yields indifference since all the possibilities have two persons:

```
The matches are:
2.        100. %  MA
2.        100. %  HA
2.        100. %  OT
2.        100. %  JU
2.        100. %  TA
```

Now we add that one person murders another:

```
        Macbeth1 murder Duncan1.
```

This improves the best score by four because the murder assertion is augmented by demons that add that MACBETH1 is evil, that DUNCAN1 is dead, and that MACBETH1 kills DUNCAN1. Now some differentiation is evident:

```
The matches are:
6.        100. %  MA
6.        100. %  HA
6.        100. %  JU
4.        66. %   OT
2.        33. %   TA
```

Next a third person, married to one of the existing ones, is added:

```
        MA1 has part Lady-Macbeth1.  Lady-Macbeth1 is a person.
        Macbeth1 marry Lady-Macbeth1.
```

```
The matches are:
9.        100. %  MA
9.        100. %  HA
7.        77. %   JU
6.        66. %   OT
5.        55. %   TA
```

Finally, it is stated that the new person is greedy:

```
        Lady-Macbeth1 has-property greedy.
```

```
The matches are:
10.       100. %  MA
9.        90. %   HA
7.        70. %   JU
6.        60. %   OT
5.        50. %   TA
```

Note 10: Matching a Situation Group against a Situation Group

Sometimes it is necessary to adapt an incremental approach to matching if the number of things to be matched would be too big otherwise. Finding the analogy between the basic constraints of the electrical world and those of the mechanical world is such a situation. It would not be practical to throw all of the parts of the laws in each group into two bags and to then match those bags. The number of possibilities doing things that way would be, in fact, a staggering 11! = 33,916,800.

The basic step in an alternative, incremental approach is to match the first unmatched thing in the first group with all the things in the second. The most similar thing is then removed from the second group and the process is repeated until one of the groups is exhausted.

Importantly, the corresponding items found in the best matches so far form part of the link list for new matches. In general, this biases the new matches toward compatibility with the old ones. In this particular example, it biases the matching toward a solution in which all voltages and all currents are identified with nothing but forces or nothing but velocities. It is also necessary to have RELATED-TO relations between all instances of voltages, currents, forces, and velocities. These relations are placed by an if-added demon on A-KIND-OF.

In the results that follow, the first match is indecisive. In fact, there are two equally good analogies between these electrical and mechanical laws as described to the system. By chance, the standard one was selected by the system at the point of indecision. The standard one would be forced by simply noting that voltage and mechanical force are both a kind of more abstract force. Much of the output is suppressed as it is extraneous.

```
Matching RESISTOR-SITUATION and DAMPER-SITUATION
Matching RESISTOR-SITUATION and SPRING-SITUATION
Matching RESISTOR-SITUATION and MOVING-MASS-SITUATION

I match RESISTOR-SITUATION against DAMPER-SITUATION

I have an initial link list:
RESISTANCE B
CONDUCTANCE 1-OVER-B
PROPORTIONAL-TO-R-2 PROPORTIONAL-TO-D-2
PROPORTIONAL-TO-R-1 PROPORTIONAL-TO-D-1
RESISTOR-VOLTAGE FORCE-ON-DAMPER
RESISTOR-CURRENT DAMPER-VELOCITY

Matching CAPACITOR-SITUATION and SPRING-SITUATION
Matching CAPACITOR-SITUATION and MOVING-MASS-SITUATION

I match CAPACITOR-SITUATION against SPRING-SITUATION
```

```
I have an initial link list:
RESISTANCE B
CONDUCTANCE 1-OVER-B
CAPACITANCE 1-OVER-K
CAPACITOR-CURRENT SPRING-VELOCITY
RESISTOR-VOLTAGE FORCE-ON-DAMPER
RESISTOR-CURRENT DAMPER-VELOCITY
PROPORTIONAL-TO-C PROPORTIONAL-TO-S3
CAPACITOR-VOLTAGE FORCE-ON-SPRING
PROPORTIONAL-TO-R-2 PROPORTIONAL-TO-D-2
PROPORTIONAL-TO-R-1 PROPORTIONAL-TO-D-1

Matching INDUCTOR-SITUATION and MOVING-MASS-SITUATION

I match INDUCTOR-SITUATION against MOVING-MASS-SITUATION

((9. RESISTANCE B)
 (9. CONDUCTANCE 1-OVER-B)
 (8. CAPACITANCE 1-OVER-K)
 (3. INDUCTOR-VOLTAGE FORCE-ON-MASS)
 (3. CAPACITOR-CURRENT SPRING-VELOCITY)
 (3. RESISTOR-VOLTAGE FORCE-ON-DAMPER)
 (3. RESISTOR-CURRENT DAMPER-VELOCITY)
 (2. INDUCTOR-CURRENT MASS-VELOCITY)
 (2. CAPACITOR-VOLTAGE FORCE-ON-SPRING)
 (1. PROPORTIONAL-TO-C PROPORTIONAL-TO-S3)
 (1. PROPORTIONAL-TO-R-2 PROPORTIONAL-TO-D-2)
 (1. PROPORTIONAL-TO-R-1 PROPORTIONAL-TO-D-1)
 (0. PROPORTIONAL-TO-L PROPORTIONAL-TO-M)
 (0. INDUCTANCE MASS))
```

APPENDIX 2: MISCELLANEOUS PLOTS (COMMENT -*-text-*-

This is the general story file -- slavish attention to consistent usage was avoided on the ground the programs should be robust enough to deal with some variety and some error -- much of the initial stuff sets up demons and establishes the A-KIND-OF tree -- the first story is MA (short for Macbeth)

Note that HQ = HAS-PROPERTY and AKO = A-KIND-OF

END OF COMMENT)

```
(setq *prematch* '(man person prop))

(setq stories '(ma ha ot ju ta))

(ferase) ;initialize

(setq *insert-part* nil) ;do not insert PART relation automatically

(fread xcause) ;read cause demons

Murder has if-added
 (fputv+ frame 'hq 'evil)
 (fputv+ (fgetc!  frame 'murder value)
         'imply
         (fgetc!  frame 'kill value)).

Kill has if-added
 (fputv+ (fgetc!  frame 'kill value)
         'imply
         (fgetc!  value 'hq 'dead)).

Hq has if-added
 (cond ((memq value '(cruel))
        (fputv+ frame 'hq 'evil))
       ((memq value '(loyal honest))
        (fputv+ frame 'hq 'good))).
```

Person has instance man and woman. Man has instance father son boy gentleman and bastard. Woman has instance mother daughter girl lady hag and bitch. Shrew is a bitch. Soldier has instance general colonel. Ruler has instance Emporer Empress King Queen Noble. Man has instance Emporer King Prince Noble. Woman has instance Empress Queen Princess.

```
(setfv *insert-part* t) ;insert PART relation automatically
```

MA IS A STORY.

{Macbeth ako noble} before {Macbeth ako king}. Macbeth marry Lady-macbeth. Lady-macbeth is a woman - hq greedy ambitious. Duncan is a king. Macduff is a noble - hq loyal angry. Wierd-sisters is a hag group - hq old ugly wierd - number 3.

Wierd-sisters predict {Macbeth murder Duncan}. Macbeth desire {Macbeth ako king} [cause {Macbeth murder Duncan}]. Lady-macbeth persuade {Macbeth murder Duncan}. Macbeth murder Duncan [coagent Lady-macbeth - instrument knife]. Lady-macbeth kill Lady-macbeth. Macbeth murder Duncan [cause {Macduff kill Macbeth}].

HA IS A STORY.

Hamlet is a prince - parent Ghost Gertrude. Ghost is a king - sibling Claudius. Claudius is a king. Gertrude is a queen. Laertes is a man.

Ghost marry Gertrude [before {Claudius marry Gertrude} {Ghost hq dead}]. {Claudius desire {Claudius ako king}} cause {Claudius murder Ghost}. {Claudius murder} cause {Hamlet kill Claudius} {Ghost order {Hamlet kill}}. {Ghost hq dead} cause {Hamlet hq unhappy}. Claudius persuade {Laertes kill Hamlet}. Laertes kill Hamlet [instrument sword - cause {Hamlet kill Laertes [instrument sword]}. Gertrude hq unhappy - kill Gertrude [instrument poisen]. Hamlet kill Claudius [instrument sword].

JU IS A STORY.

Caesar is a general emporer - hq ambitious foolish. Brutus is a man - hq honest unhappy - love Rome. Antony is a man - hq loyal. Cassius is a man - hq thin.

Cassius hate Caesar - persuade {Brutus murder Caesar [instrument knife]} - murder Caesar [instrument knife]. Antony love Caesar - persuade {people attack Brutus} {people attack Cassius}. Brutus attack Antony. Cassius attack Antony. Brutus kill Brutus. Cassius kill Cassius.

OT IS A STORY.

Othello is a Moor general and man - hq weak foolish. Desdamona is a woman - hq honest. Iago is a man - hq cruel ambitious. Cassio is a man - hq loyal weak alcoholic. Othello love Desdamona - marry Desdamona. Desdamona love Othello.

Iago hate Cassio [cause {Iago persuade {Othello believe {Desdamona love

Cassio}}}]. Desdamona love Cassio [hq lie]. {Othello believe} cause {Othello jealous Cassio}. {Othello jealous} cause {Othello hate Cassio} {Othello help Iago} {Othello kill Desdamona} {Othello kill Othello} {Cassio hq dead}.

TA IS A STORY.

Petruchio is a man - hq strong smart. Katharina is a shrew - hq rich. Lucentio is a student man. Bianca is a woman - sibling Katharina.

Petruchio marry Katharina - forbid {Katherina eat *}. {Petruchio love Katharina} cause {Petruchio forbid}. {Petruchio forbid} cause {Katharina love Petruchio}. Katharina hq masochist [cause {Katharina love Petruchio}]. Petruchio dominate Katharina [caused-by {Katharina love Petruchio} - hq strong].

Bianca love Lucentio - marry Lucentio. Lucentio love Bianca. Lucentio dominate Bianca [hq weak].

eof ;stop here for most work

DH IS A STORY.

Nora is a woman - hq sensitive proud - friend-of Christina - marry Torvald. Friend-of has inverse friend-of. Torvald is a man - hq crude weak. Krogstad is a man - hq cruel. Christina is a woman - hq good. Letter-dh is a letter prop.

Torvald love Nora - hate Krogstad - help Christina [caused-by {Torvald hurt Krogstad}]. Krogstad prevent {Torvald help Cristina} [act {Krogstad threaten Nora [act {Krogstead give letter-dh [destination Torvald]}]}]. Christina prevent {Torvald give letter-dh} [act {Christina marry Krogstad} - hq failed]. Torvald attack Nora [caused-by {Krogstad give letter-dh}]. Nora leave Torvald [caused-by {Torvald attack}].

HG IS A STORY.

Hedda is a woman - hq proud beautiful. George is a man - hq dull. Lovberg is a man - hq smart. Elvsted is a woman - hq dull ugly. Pistol-hg is a pistol prop. Manuscript-hg is a manuscript prop.

Hedda marry George - hate George. George want {George ako professor} - jealous-of Lovberg. Lovberg love Hedda - write manuscript-hg - lose manuscript-hg. Elvsted help Lovberg [act {Lovberg write}] - control Lovberg. Hedda hate Elvsted [caused-by {Elvsted control Lovberg} {Hedda want {Hedda control Lovberg}}]. Hedda persuade {Lovberg destroy manuscript-hg} [caused-by {Hedda control Lovberg}]. Lovberg kill Lovberg [caused-by Hedda - hq failed - instrument pistol-hg]. Hedda kill Hedda [instrument pistol-hg].

PY IS A STORY.

Eliza is a girl - hq beautiful proud stubborn sensitive. Higgens is a professor man - hq smart crude - friend-of Pickering. Pickering is a gentleman colonel - hq wise old proper - friend-of Higgens. Freddy is a man - hq poor young foolish.

Higgens teach Eliza [subject English] - help Eliza [method teach]. Higgens hurt Eliza [caused-by {Higgens hq crude} - cause {Eliza leave Higgens} {Eliza marry Freddy}] - love Eliza [hq strange].

Pickering help Eliza. Eliza is a lady [caused-by Higgens]. {Higgens cause} method teach. Eliza love Higgens [caused-by {Higgens help}]. Eliza marry Freddy [cause {Higgens hq unhappy} {Eliza hq unhappy}].

CI IS A STORY.

Cindy is a servant girl - hq beautiful sweet unhappy. Charming is a prince - hq beautiful. Godmother is a fairy and woman - hq wise old and kind. Stepmother is a bitch - hq cruel. Glass-slipper is a prop and shoe.

Step-mother hates Cindy. Godmother help Cindy [act {Godmother give dress} {Godmother give coach}]. Cindy attend ball. Cindy meet Charming - love Charming. Charming love Cindy - lose Cindy. Charming find Cindy [instrument glass-slipper]. Cindy marry Charming - hq happy.

AD IS A STORY.

God is a spirit - hq good strong. Spirit is a person. Devil is a spirit - hq bad strong cruel. Adam is a man - hq happy. Eve is a woman - hq happy foolish. Evil-apple is a apple prop.

God forbid {Adam eat evil-apple} [hq failed]. Devil persuade {Eve persuade {Adam eat}}}. God move Adam [source Paradise - destination Outer-darkness]. God move Eve [source Paradise - destination Outer-darkness]. God punish Adam [caused-by {Adam eat} - act {God move Adam} {God move Eve}].

eof