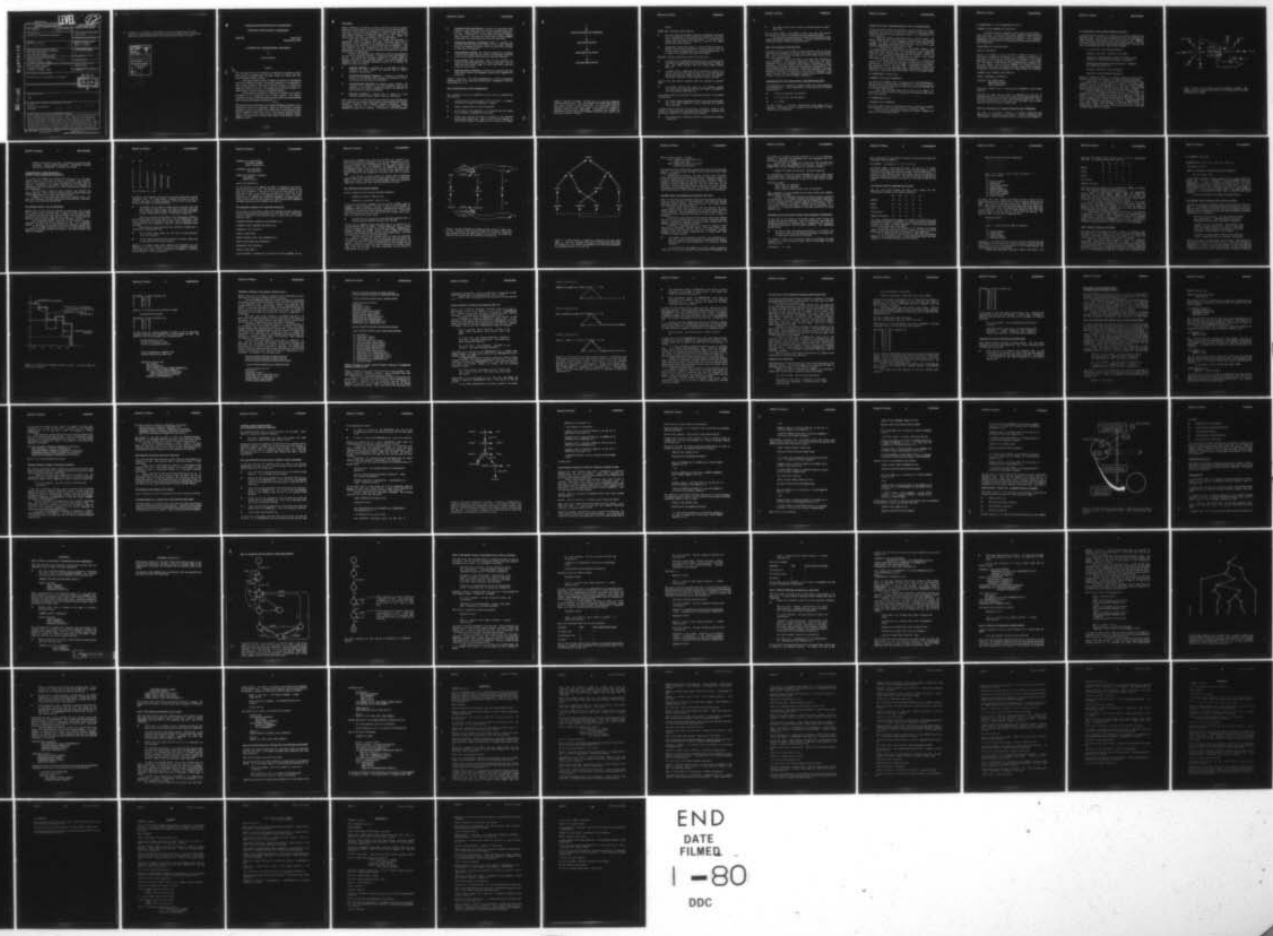
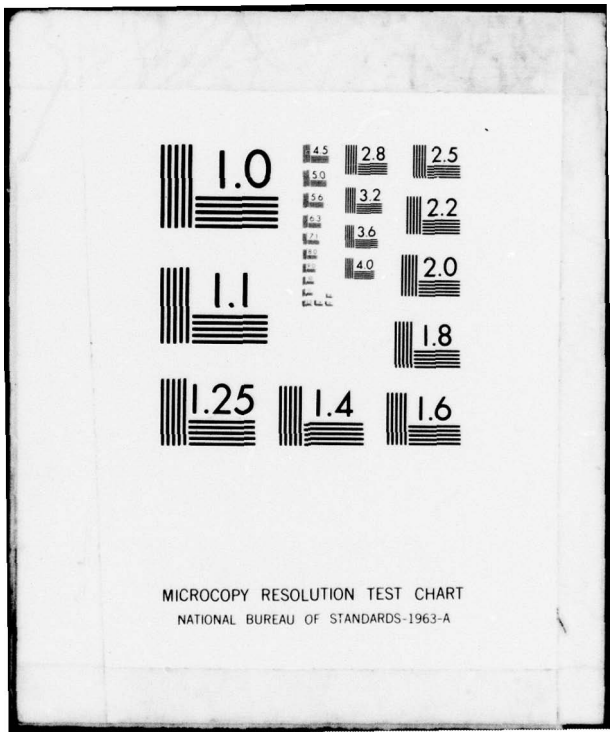
AD-A078 123 MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTE--ETC F/G 6/4
LEARNING BY UNDERSTANDING ANALOGIES.(U)
JUN 79 P H WINSTON N00014-75-C-0643
UNCLASSIFIED AI-M-520 NL

1 OF 1
AD A078123



END
DATE FILMED
1-80
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data En

LEVEL II

12

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
A. I. Memo 520

2. GOVT ACCESSION NO.
AI-M-520

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Learning by Understanding Analogies

5. TYPE OF REPORT & PERIOD COVERED
memorandum sept,

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Patrick H. Winston

8. CONTRACT OR GRANT NUMBER(s)
N00014-75-C-0643,
N00014-77-C-0389

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, Virginia 22209

12. REPORT DATE
June 79

13. NUMBER OF PAGES
91

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Office of Naval Research
Information Systems
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

DDC
RECEIVED
DEC 13 1979
D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Learning
Artificial Intelligence
Analogies

79 12 11 030

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We use analogy when we say something is a Cinderella story and when we learn about resistors by thinking about water pipes. Experts use analogy when they learn 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 deal with an unfamiliar situation. A teacher may supply the analogy or may not. The analogy may be between situations in a single domain or between situations in very different domains. Frames represent the situations. Relations are

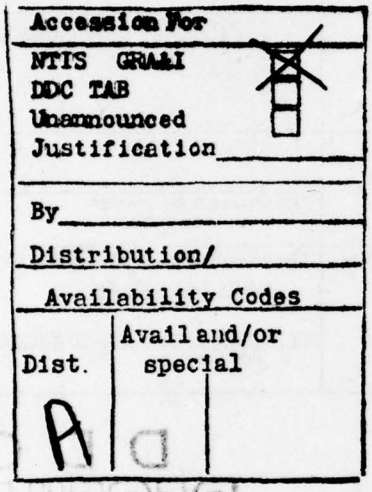
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

UNCLASSIFIED 407 483 mt
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A078123

DDC FILE COPY

20. expressed in a way that is reminiscent of the case-grammar view of simple sentences. The essential computations tie pairs of frame-described situations together and make knowledge about one become knowledge about the other.



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

AIM 520

April 1979
Revised June 1979

LEARNING BY UNDERSTANDING ANALOGIES

by

Patrick H. Winston

Abstract

We use analogy when we say something is a Cinderella story and when we learn about resistors by thinking about water pipes. Experts use analogy when they learn 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 deal with an unfamiliar situation. A teacher may supply the analogy or may not. The analogy may be between situations in a single domain or between situations in very different domains.

Frames represent the situations. Relations are expressed in a way that is reminiscent of the case-grammar view of simple sentences. The essential computations tie pairs of frame-described situations together and make knowledge about one become knowledge about the other.

This paper describes research 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 of thinking is by analogy. We face a situation, we recall a similar situation, we match them up, we draw conclusions, we test the conclusions. We do all of this when we say something is a Cinderella story and when we learn about resistors by thinking about water pipes. Experts do it when they analyze situations in Economics, Medicine, and Law. Indeed, the experts often learn their specialties by the case study method.

This paper presents a theory of analogy and describes an implemented system that embodies the theory. The paper begins with a description of the competence to be understood and a discussion of the representation that seems best suited to dealing with the competence. Then the competence and the representation are used to specify certain computations. And finally, specific, implemented algorithms are presented that perform the computations.

Briefly, the competence to be understood is that of using analogies to deal with an unfamiliar situation. A teacher may supply the analogy or may not. The analogy may be between situations in a single domain or between situations in very different domains. The theory explains the competence when the individual situations involved in an analogy are subject to certain principles:

- **Symbolic sufficiency.** A situation can be described by using a repertoire of types, features, and relations that is finite, although it may be large.

- **Description-determinined similarity.** A situation is similar to another if the important types, features, and relations in their descriptions can be placed in correspondence.

- **Cause-determined importance.** The important types, features, and 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 causal relationships.

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

Given these points, it would seem that any system for doing analogy must use a powerful representation scheme together with mechanisms that propose potential analogies, match allegedly analogous situations, suggest conclusions, and test to see if conclusions are supportable. The implemented system has all of these. In particular it has the following:

- **Commented-frames representation.** Situations are represented using frame-slot-value triples that stress the most important parts of a relation. Comments attached to the triples facilitate elaboration. The idea is reminiscent of the way agents and objects dominate simple sentences. The agents and objects appear without flagging prepositions, while other things do not get the same status.

- **Classification-exploiting hypothesizing.** Memory is searched for situations that are likely to be similar to a new, given situation because the remembered situations involve the same sorts of things at some level of abstraction.

- **Cause-dominated matching.** The similarity between two situations is measured by finding the best possible correspondence according to the causal framework exhibited by the situations themselves.

- **Frame-oriented rules.** Conclusions about a given situation are reached by using knowledge found in a similar situation. Sets of frame-slot-value triples trigger the rules, which then create new frame-slot-value triples.

- **Experience-driven verification.** A conclusion is tested by using the causal chains found in the remembered situation that suggested the conclusion.

Figure 1 illustrates. The actual implementation is based on primitives resembling those in the frame-oriented language FRL, implemented in LISP by Bruce Roberts [12, 13].

THE COMPETENCE TO BE UNDERSTOOD

More concretely, the sort of competences I have in mind are suggested by the following:

- A story outline is given in terms of 40 or 50 facts. It appears reminiscent of several of Shakespeare's tragedies.

- Analysis suggests that it is most like Macbeth.

- On the basis of the similarity, it is predicted that the person that corresponds to Macbeth will end up dead.

- Working more, questions are asked to determine if the predicted outcome really makes sense. Does the person that corresponds to Lady Macbeth persuade the Macbeth person to murder the Duncan

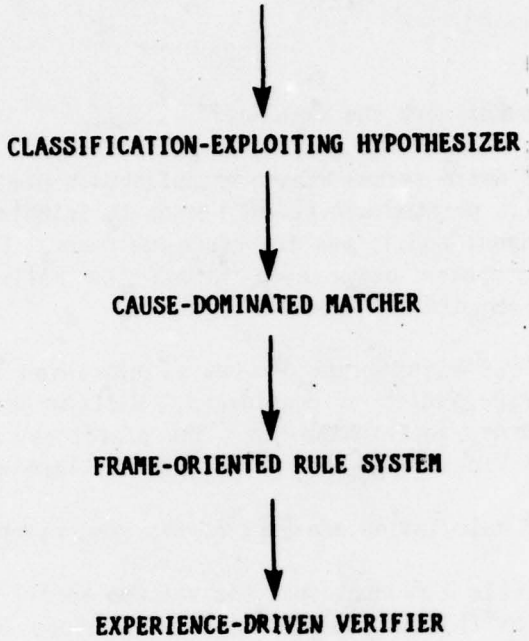


Figure 1: The keys to analogy. The first job is to find some remembered situation that may apply. Next, the hypotheses are checked using a matcher that relies heavily on what causes what. The best match leads to conclusions by way of rules that trigger on the existence of situation-defining frame-slot-value combinations. Conclusions are checked by determining if causes found in the remembered situation can be found in the new one or otherwise confirmed.

person?

Simple legal situations work the same way:

- 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. It is held that an assault has taken place even though the rifle was not loaded because Wesson did not know it.

- 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. It is asked if Victim knew that the pistol was harmless.

Analogies involving calculation are part of the same story:

- 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.

- A teacher tells a student the exports and imports of wheat in a certain year affect the supply in that year in the same way the faucet and drain rates affect the amount of water in a bathtub. The student calculates the wheat supply at the end of the year.

Practice with some specific instances enables the invention of specific laws:

- 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.

Once several forms of the same sort of constraint are known, it is possible to generalize:

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

I propose that the ability to do this kind of learning is related to the ability to understand the relationship between specific instances and applicable constraints as illustrated by these examples:

- The voltage across a particular resistor is calculated by reference to Ohm's law.

- The amount of wheat after a period is calculated using the import-export law.

All of these examples, from Macbeth to Ohm's law, have been handled successfully in a series of experiments using an implemented system.

The experiments seemed successful, subjectively, because the system reached justifiable, human-like conclusions. A person could reach the same conclusions without seeming particularly eccentric.

There are Precedents in Previous Work

From time to time it may be useful to make comparisons with my previous learning systems inasmuch as the roots of this one go back to the earlier ones. Let us call the first one ARCH [18] and the second FOX [19] after the typical things involved.

In addition, many of the ideas in this paper were influenced by other precedents. In particular, Schank [15], Wilks [17], and others stimulated work on the problem of how thinking is determined by stored experience. Evans [3] and Brown [1] broke ground on certain special forms of the analogy problem. Minsky [8], and Goldstein and Roberts [12, 13] worked out key ideas on representation. Martin [6] and Rieger [11] brought attention to the need for hard work on the details of vocabulary. And Lenat's success with his mathematical discovery system provoked renewed interest in the entire area of computer learning [5].

REPRESENTING SITUATIONS USING COMMENTED-FRAMES

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

- It makes the important facts explicit.
- It is clear that it can be computed.
- It is simple.

In this section, a particular representation using frames will be explained. Alternatives will be dismissed, and the importance of cause relations will be emphasized.

Frames can be used to Describe Situations by way of Commented Slots

In describing both stories and constraints, binary relations are clearly important. There must be a clean way of saying that Cinderella kisses Prince Charming and that some particular voltage, V, is proportional to a certain current, I.

Using the familiar property-list idea is nearly enough. The symbols CINDERELLA and V would have lists of property-value pairs attached to them. The property list of CINDERELLA would have KISS combined with CHARMING, the symbol representing Prince Charming. The one for V would include a PROPORTIONAL-TO property with an I value.

A property-list representation is not enough, however. The relationship between Cinderella and Prince Charming is really something about which more could be said. The relationship between voltage and current is really a ternary relationship involving resistance. There is no room in the property-list representation to express these things comfortably. There is room in the frames representation.

For my purpose here, 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:

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

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

The frames are CINDERELLA and V. The slots are KISS and PROPORTIONAL-TO. And the values in the VALUE facet are CHARMING and I. Putting these down in the form of nested list structure is a habit that derives from working with a LISP-based implementation.

Since only the VALUE facet will be used for the moment, the notation for a frame often will be abbreviated by omitting the facet part, as in the following:

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

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

The other facets are used to hold default values or demon-like procedures to be executed as values are placed in a slot, removed, or sought.

Each value in a frame can be accompanied by comments. Comments are shown by way of additional nesting.

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

(CINDERELLA (KISS (CHARMING (SEE (KISS-CI)))))
```

In standard practice, comments are often used to record where a value came from. In this work on analogy, comments assume additional 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-CI are frames that further describe frame-slot-value combinations:

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

(KISS-CI (TIME (END)))
```

Thus it is possible to say a lot about a relation or an act. In particular, in describing a simple act, it is possible to specify all of the objects involved in the action using a vocabulary of slot names something like the vocabulary of cases typically used in a case grammar [4]. For example, in the sentence "Prince Charming found Cinderella with her glass slipper," FIND is the act, CHARMING is the agent of the act, Cinderella is the object, and Cinderella's glass slipper is the instrument. Consequently translating the sentence into frames produces the following:

```
(CHARMING (FIND (CINDERELLA (SEE FIND-CI))))

(FIND-CI (INSTRUMENT (SLIPPER-CI)))

(SLIPPER-CI (AKO (SHOE) (PROP))
            (RAW-MATERIAL (GLASS))
            (OWNER (CINDERELLA)))
```

Where AKO = A-Kind-Of and CI = suffix used in *Cinderella* to avoid naming conflicts.

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.

There are Alternatives to Commented Frame-slot-value Combinations

The idea of representing relations as commented frame-slot-value combinations was suggested to me by Roberts. Several alternatives were then rejected. They are described in note 1. All notes are in Appendix 1.

It is Convenient to have an Input Language and Demons

In working with many examples, it is good to have an easily readable way of preparing frames. Note 2 gives the transition network that describes the input translator currently used. The following version of the CINDERELLA frames illustrates the input form that the translator accepts. Note the use of AKO, meaning A-Kind-Of, and HQ, meaning Has-Quality. These are the only slot abbreviations used in this and other examples.

> CI is ako story - part Charming Cinderella.
>
> Charming is ako prince. Cinderella is ako woman.
>
> Charming has job entertaining - hq brave and strong - loves Cinderella. Cinderella has job cleaning - hq beautiful.
>
> Charming persuade Cinderella [see persuade-ci]. Cinderella kiss Charming [see kiss-ci].
>
> Persuade-ci act kiss-ci. Kiss-ci time end.
>
> Where AKO = A-Kind-Of and HQ = Has-Quality.

With more complicated stories, it is usually necessary to start with a diagram, such as the one in figure 2, in order to keep things straight.

Additionally, it is good to have obvious inferences made on input, reducing the need for tedious attention to details. For example, when Cinderella marries Prince Charming, it is clear that Prince Charming marries Cinderella and that each is married to the other. Similarly when one person kills another, it is clear that the killed person is dead.

Using if-added demons is the way to do this. If-added demons are procedures that are automatically invoked when a slot is filled. In Roberts' FRL, these procedures are stored in the IF-ADDED facet of the slot they are relevant to and they are inherited through AKO chains. In my own private implementation, I prefer to put them in a frame describing the slot. Thus I specify if-added demons, for example, for MARRY and KILL. In these demons, FRAME, SLOT, and VALUE are variables whose values are automatically assigned by the basic value-insertion function for use by any if-added demons that are triggered:

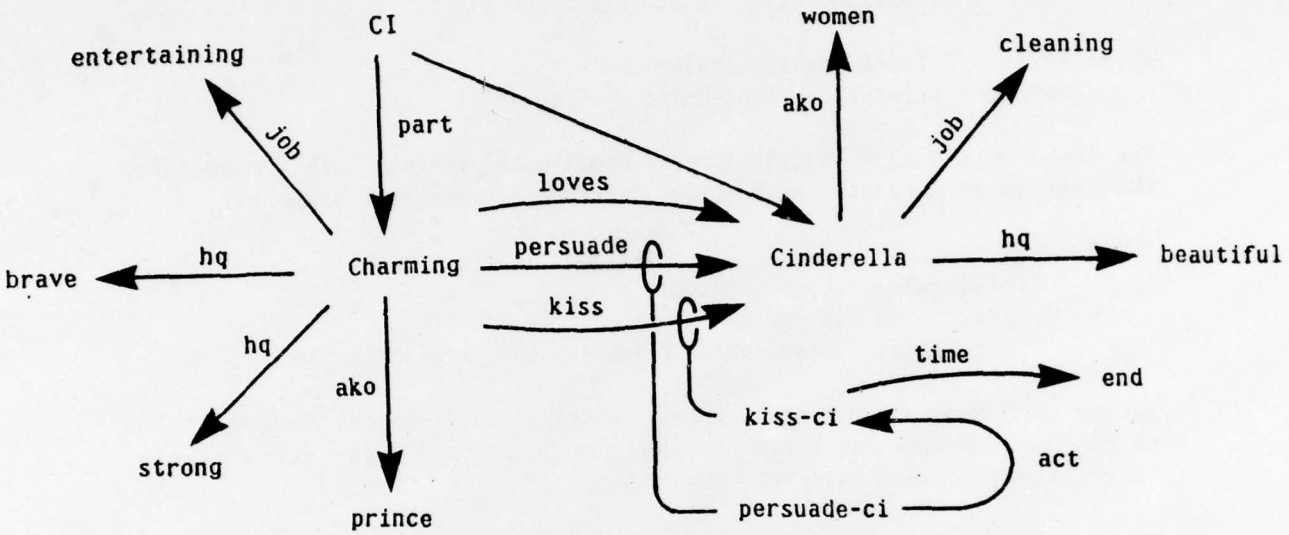


Figure 2: Doing a story usually starts with preparing a diagram. Each comment frame is connected to the arrow of the relation that is further described by the comment.

```
MARRY has if-added (FPFSV FRAME 'MARRIED-TO VALUE).

KILL has if-added
        (FPFSVCV VALUE 'HQ 'DEAD
                'BECAUSE (LIST FRAME 'KILL VALUE)).

Where FPFSV = F-Put-Frame-Slot-Value and
      FPFSVC = F-Put-Frame-Slot-Value-Comment-Value
```

The demon on the slot INVERSE is more complicated perhaps. It arranges for the inverse of a relationship to be asserted if the relationship is:

```
INVERSE has if-added
        (fpfsv value 'inverse frame)
        (fpfsv frame 'if-added
               (list 'fpfsv 'value (list 'quote value) 'frame)).
```

In the end, demons prove so important to the analogy process that there has to be some concern about whether they can be learned. In fact they can. Later, the following will be argued:

- Using a demon is just 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 relationships at the level of demons.

Capturing Story Plots Requires Attention to Cause

Using commented frames, a dozen story plots were set down. The purpose was to discover if the representation is adequate as a way of arranging symbols and to work out a sample vocabulary of symbols.

The stories include four of Shakespeare's tragedies, two plays by Ibsen, and some random things, all selected by thumbing through an encyclopedia of plots.

For the most part, things seemed to work out, leading to the results in the Appendix 2 and to the following observations:

- 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 [10] and in the first thousand or two most frequent English words [2].

- Time is tricky. In the examples time is represented in two ways. First, time is represented explicitly by using scene frames as the value in TIME slots. Second, time is represented implicitly by way of CAUSE slots.

- Cause is tricky too.

To handle cause, the following conventions were honored, with seeming success:

- The CAUSE and PREVENT slots can occur in any frame.

- Acts or relations are caused or prevented. People are not. Consequently a comment frame is the only thing that can be in a CAUSE or PREVENT slot. The inverses of CAUSE and PREVENT are CAUSED-BY and PREVENTED-BY. The REASON, MOTIVE, and DESIRE slots are closely related to CAUSED-BY.

  Here is an illustrative fragment from Appendix 2:

  Katharina hq masochist [see hq-ta] - love Petruchio [see love-ta]. Hq-ta cause love-ta.

- People can be ordered to do something or forbidden to do something. Thus the ORDER and FORBID slots have people in them. Comment frames describing specific instances of ordering and forbidding can specify the acts involved by way of an ACT slot.

- A comment frame describing a specific instance of a cause may specify a method by way of a METHOD slot. Popular occupants of the METHOD slot are comment frames referring to specific instances of ORDER or FORBID slots.

  The following, from *Hamlet*, illustrates: Hamlet kill Claudius [see kill-ha]. Ghost cause kill-ha [see cause-ha]. Cause-ha method order-ha. Ghost order Hamlet [see order-ha]. Order-ha act kill-ha.

- People can be persuaded to do something or that something is true. Thus PERSUADE slots have people in them. Their comments can have an ACT slot, specifying something to do, or a RELATION slot, specifying something to believe. Their comments also can have a METHOD slot. The METHOD slot describing a cause may specify an instance of a persuade relation. The SUGGEST slot behaves much like PERSUADE.

  Consider this:

Macbeth kill Duncan [see kill-ma]. Lady-Macbeth cause kill-ma [see cause-ma]. Cause-ma method persuade-ma. Lady-Macbeth persuade Macbeth [see persuade-ma]. Persuade-ma act kill-ma.

DETERMINING CORRESPONDENCE USING CAUSE-DOMINATED MATCHING

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 for which desperate need already had been established.

For the moment, assume all types, features, and relations are equally important. Later, it will be shown how cause relations can be used to distinguish important relations from incidental ones.

Examples will deal with both stories and scientific laws. For some, the system must do a kind of abstraction before matching is possible. For others, the system must ask questions.

The Simplest Matcher Tries *all Possibilities*

Suppose there are two stories or situations or constraints, each of which has a group of frames in the PART slot. In general, if there are N1 frames in one group and N2 in the other, then the number of ways the frames can be paired up is N1!/(N1-N2)!, given that N1 is equal to or greater than N2. Often N1 equals N2, and the number of distinct pairings is just N1!.

The implemented matcher tries each of the possible pairings, calculates how good each is, and announces the best. It is therefore similar to the matcher used by Evans in his work on geometric analogy [3].

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:

N1	N2				
	1	2	3	4	5
1	1				
2	2	2			
3	3	6	6		
4	4	12	24	24	
5	5	20	60	120	120
6	6	30	120	TB	TB
7	7	42	TB	TB	TB
8	8	56	TB	TB	TB
9	9	72	TB	TB	TB
10	10	90	TB	TB	TB

TB ==> Too Big, i.e. > 150

The matcher, when compiled, handles 100 or so pairing possibilities without excessive strain. For larger numbers, something else must be done to constrain the number of pairings considered. We will return to this later. For now, think about the following question:

- The matcher will be unable to handle groups with more than some small number of frames in them. Thus there is a natural constraint on the analogies that the system can understand in that the number of parts involved must be small. Is this true of people too?

Each way of pairing off the frames in two frame groups produces a set of paired frames that we will refer to as a list of linked pairs. Each list of linked pairs is evaluated in two steps: first a similarity score is calculated for each linked pair in the list; then the similarity scores are added.

The similarity between two frames that constitute a linked pair is calculated by inspecting the slots:

- If two linked frames contain the same value in some particular slot, score one point.

- If two linked frames contain the two parts of another linked pair in some particular slot, score one point.

Suppose, for example, that Prince Charming and Cinderella are the characters in one story 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))
          (HQ (BRAVE) (STRONG))
          (LOVES (CINDERELLA)))

(CINDERELLA (JOB (CLEANING))
            (HQ (BEAUTIFUL)))

(ROMEO (JOB (FIGHTING) (CLEANING))
       (HQ (STRONG))
       (LOVES (JULIET)))

(JULIET (HQ (BEAUTIFUL)))
```

The winning match pairs CHARMING with ROMEO and CINDERELLA with JULIET, giving a match score of three. The similarity of CHARMING and ROMEO is two under this arrangement -- one point is for having the same value in the HQ slot and the other is for having the two halves of a linked pair in the LOVES slot. The similarity of CINDERELLA and JULIET is one. The other possible match, CHARMING with JULIET and CINDERELLA with ROMEO gives a score of one because the only similarity is that found between CINDERELLA and ROMEO because both have CLEANING in the JOB slot.

Corresponding Comments are Treated like Linked Pairs

For the sake of illustration, suppose Prince Charming persuades Cinderella to kiss him and Romeo persuades Juliet to do the same to him. These facts would be indicated by the following frames, in which only the relevant slots are shown:

```
(CHARMING (PERSUADE (CINDERELLA (SEE (PERSUADE-CI)))))

(CINDERELLA (KISS (CHARMING (SEE (KISS-CI)))))

(PERSUADE-CI (ACT (KISS-CI)))

(KISS-CI (TIME (END)))

(ROMEO (PERSUADE (JULIET (SEE (PERSUADE-RJ)))))

(JULIET (KISS (ROMEO (SEE (KISS-RJ)))))

(PERSUADE-RJ (ACT (KISS-RJ)))

(KISS-RJ (TIME (END)))
```

Pairing CHARMING and ROMEO gives one new point from the PERSUADE slot as

does pairing CINDERELLA and JULIET from the KISS slot. However it is evident that pairing the characters in this way puts PERSUADE-CI and KISS-CI in correspondence with PERSUADE-RJ and KISS-RJ. There should be two additional points since the PERSUADE-CI and PERSUADE-RJ frames have corresponding frames in the ACT slot and since KISS-CI and KISS-RJ have corresponding frames in the TIME slot. The total score should be seven. Figure 3 illustrates the combinations that lead to this score in graphic form.

This scoring is insured because the 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.

The AKO Slot Needs Special Handling

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.

Curiously, having *PRINCE in CHARMING's AKO slot and BOY in ROMEO's lends no* strength to their similarity, nor does it help to have PRINCESS in CINDERELLA's AKO slot and GIRL in JULIET's. Something must be done so that the matcher recognizes the implied relationships that we see. It must know that a prince and a boy are the same sex as are a princess and a girl. Here is a solution that seems to work:

- All AKO slots are treated as if they contained everything that is found by tracing through the AKO hierarchy from them.

The AKO slot of CHARMING contains only PRINCE, but it is treated as if it contained PRINCE, MAN, and PERSON. Similarly ROMEO's AKO slot is treated as if it contained not only BOY, but also MAN and PERSON. CINDERELLA is a PRINCESS, a WOMAN, and a PERSON. JULIET is a GIRL, a WOMAN, and a PERSON. Figure 4 shows the AKO hierarchy that gives these augmentations. They lead to a match score of eleven, four more than before.

Some may feel it lacks aesthetic purity to treat one particular slot specially. Consequently I note in passing that there is another way to achieve the same effect without putting knowledge about AKO slots in the matcher. The idea is to put the following demon in the if-added slot of AKO so that AKO slot expansion is done as AKO values are recorded, rather than at the time of match. The demon, regrettably, will be obscure to those who do not know LISP:

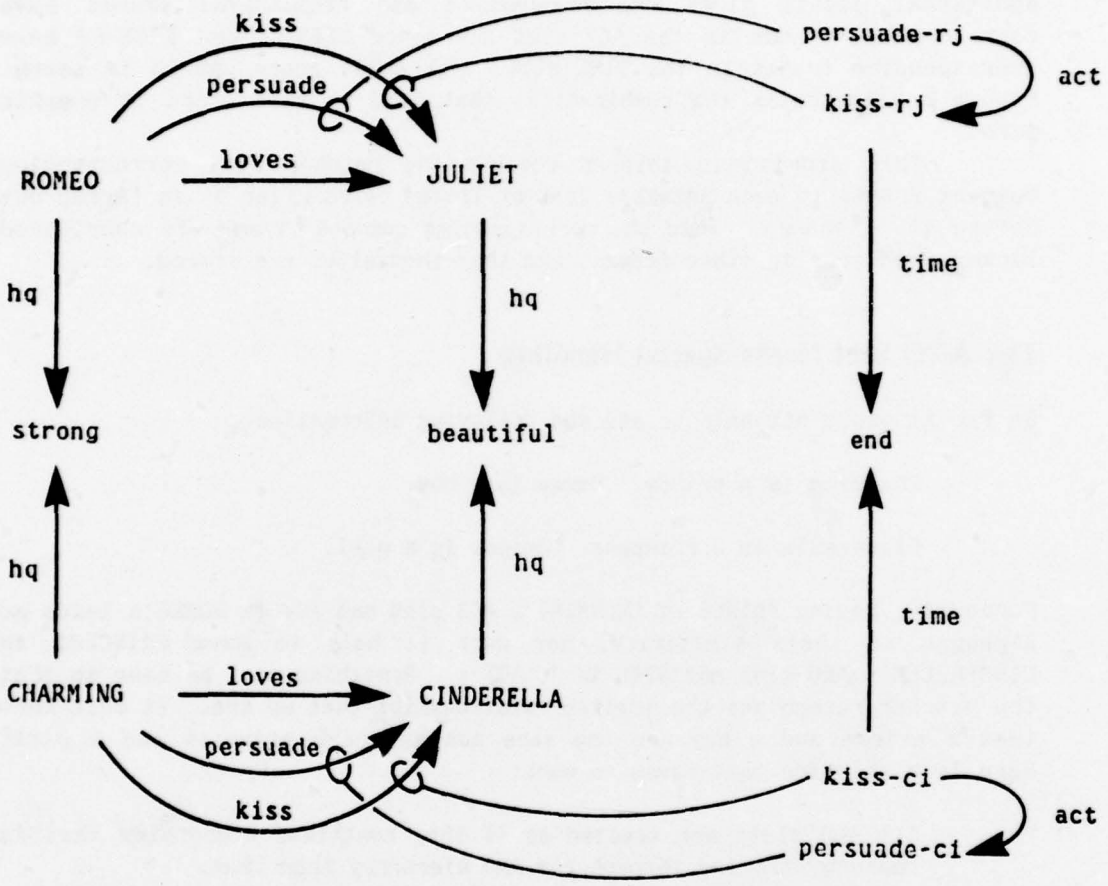


Figure 3: Matching *Cinderella* with *Romeo and Juliet* produces a match score of seven. Having the same qualities accounts for two points; having the same relations between the people accounts for three more; and having the comment frames on the link list accounts for the final two.

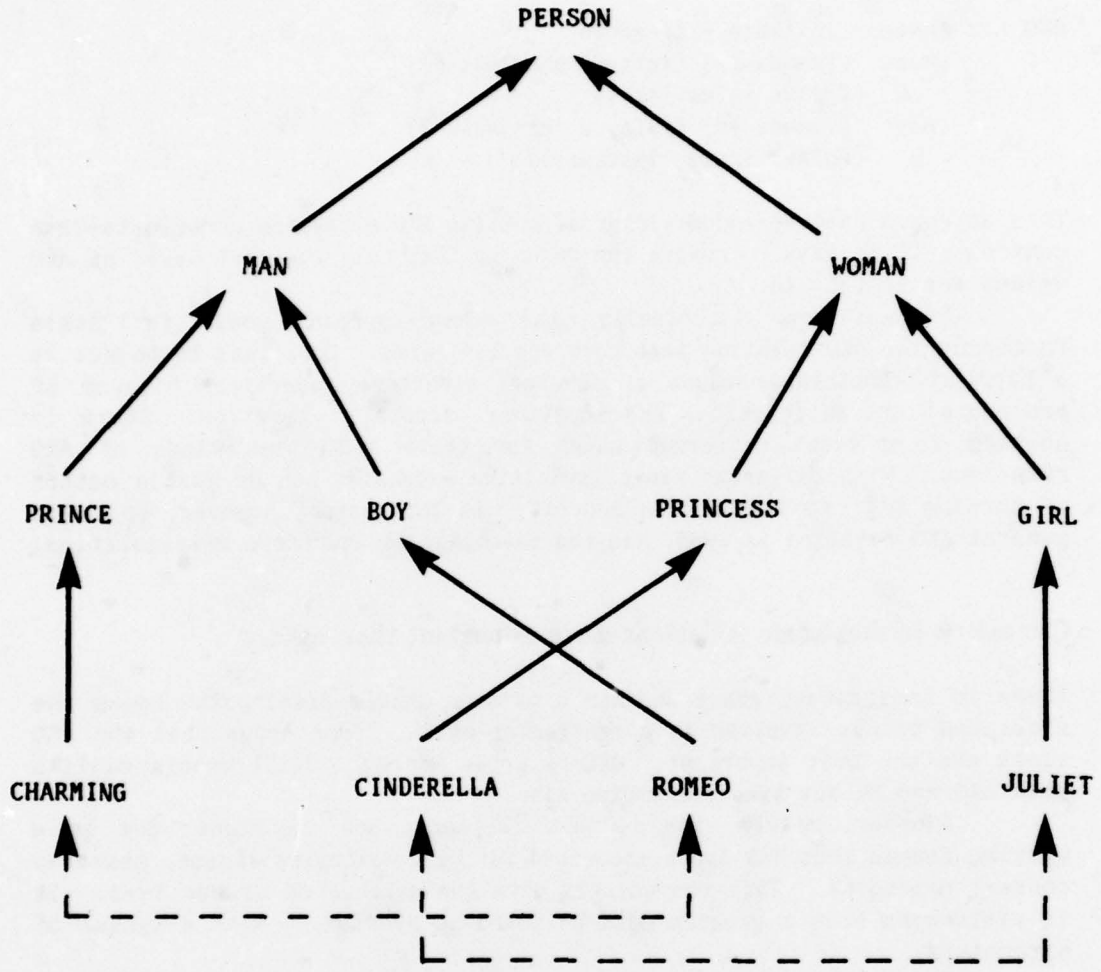


Figure 4: The AKO hierarchy is exploited in matching if the parts have types. The AKO slots contribute four points to the scoring of the match with ROMEO paired with CHARMING and CINDERELLA with JULIET even though each has something different in its AKO slot.

```
AKO has inverse instance - if-added
        (mapc '(lambda (e) (fpfsv frame 'ako e))
              (fgfsvs value 'ako))
        (mapc '(lambda (e) (fpfsv e 'ako value))
              (fgfsvs frame 'instance)).
```

This approach has the added virtue of putting AKO expansion under data-base control. It is easy to revise the demon so that only the next level of AKO values are brought in.

Perhaps more importantly, the demon approach would facilitate factoring the AKO relation into more precise forms. Sometimes an object is a kind of something because of physical structure, sometimes because of procedural characteristics, and sometimes because of location. There is nothing to prevent different names for these and other kinds of AKO relations. With different names, selective expansion can be just a matter of turning the right demons on and off. In this paper, however, only the general AKO relation is used, and the advantage is therefore a speculation.

Causality Makes some Relations more Important than others

There is some debate about whether a matcher should distinguish among the slots and values involved in a particular match. Some argue that the AKO slots are the most important. Others argue for HQ. Still others dislike both AKO and HQ but like everything else.

Another, milder view is that all slots are 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 discovered through the analogy process. Sometimes the important relations are AKOs, sometimes HQs, and sometimes things that are normally incidental.

Thankfully, importance tends to be taught by teachers, either explicitly or implicitly. Explicit teaching is done when teacher says, perhaps without justification, that some fact is important. Implicit teaching is done when a teacher includes some fact in 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 and by noting whether it is involved in causing something.

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 HQ slot. The HQ slot of a comment frame can have IMPORTANT placed in it directly as follows:

```
Macbeth kill Duncan [see kill-ma]. Kill-ma hq important.
```

Or, alternatively, the HQ 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:

```
Cause has if-added
        (fpfsv frame 'hq 'important)
        (fpfsv value 'hq 'important)
        (fpfsv (fgfsvcr frame slot value 'see) 'hq 'important).
```

Other candidates for such treatment are the cause-related slots PREVENT, MOTIVE, REASON, and DESIRE.

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 seems to 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 type, 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)! * W1!/(W1-W2) = 3! * 4! = 6 * 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 types for the matcher. There may be some way of doing this by inspecting the AKO hierarchy in the vicinity of the frames involved in the match.

The Matcher finds Corresponding parts in Stories

Here are some results showing the match scores between the four Shakespearean tragedies and one comedy in Appendix 2:

	MA	HA	OT	JU	TA
MAcbeth	61	23	14	17	10
HAmlet	23	66	14	19	9
OThello	14	14	57	17	13
JUlius Caesar	17	19	17	51	7
TAming of The Shrew	10	9	13	7	46

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 56. Evidently the average number of facts known about each story is therefore 56. Some of the facts are derived by demons and others are implied by the AKO connections.

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

```
Matching MA and HA Trying 120. permutations.
 .......... ..........
 .......... ..........
 .......... ..........
 .......... ..........
 .......... ..........
 .......... ..........

Score = 23. versus 61. and 66. Match is decisive -- 1.
better than next best.

(23.
 (9. MACBETH CLAUDIUS)
 (5. DUNCAN GHOST)
 (4. LADY-MACBETH GERTRUDE)
 (3. MACDUFF HAMLET)
 (1. MURDER-MA MURDER-HA)
 (1. WEIRD-SISTERS LAERTES)
 (0. KILL-MA KILL-CLAUDIUS)
 (0. AKO-MA-2 AKO-HA))
```

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. Their wives are Lady Macbeth and Gertrude. Macduff and Hamlet kill them. Other matches are weak and contribute only weakly to the similarity.

On the other hand, The *Taming of the Shrew* and *Julius Caesar* show little similarity. In fact, there is nothing beyond the fact that there are four people to pair up and two of the four have the same sex. The score of seven is at the level of background noise.

```
Matching TA and JU
 .......... ..........
 ....

Score = 7. versus 50. and 51. Match is indecisive.

(7.
 (2. PETRUCHIO BRUTUS)
 (2. BIANCA ANTONY)
 (2. LUCENTIO CAESAR)
 (1. KATHARINA CASSIUS))
```

Incidentally, the general shape of the table, as well as the best and the worst matches, are the same if zero points are given for values in the AKO slot or even for values in both AKO and HQ. For the stronger matches, the relationships between parts are enough.

Perhaps more importantly, 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	OT	JU	TA
MAcbeth	14	12	5	9	0
HAmlet	12	27	6	11	0
OThello	5	9	16	7	1
JUlius Caesar	9	11	7	24	1
TAming of The Shrew	0	0	1	1	11

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

The table was produced by the following steps: first, importance-marking demons on CAUSE, PREVENT, MOTIVE, REASON, and DESIRE were activated; second, existing demons on MURDER and KILL not only infer new facts, they also note that the triggering facts cause the inferred facts; and third, only relations marked as important are counted.

In this context, it probably would be sensible to add further conditioning to the demon mechanism such that if-added demons would fire only if their antecedents are important. As it stands, the demons on MURDER and KILL make instances of MURDER, KILL, and certain HQs important by automatically placing them in CAUSE chains.

Unfortunately, conditioning demons on importance is not practical at the moment. The reason is that if-added demons, as now implemented, are triggered when a relation is added. Consequently demons have no knowledge of what will be said about a relation after it is in place. In particular, demons cannot predict whether IMPORTANT will end up in the HQ slot of a relation's comment frame at the time the relation is first established.

Other Similarity Measures are Possible

The similarity measure between situations is scored as a byproduct of the matching process. The similarity is just the total number of corresponding slot-value combinations exhibited when the frames in two groups are optimally paired. Thus the similarity is a measure of overlap.

Many other authors have considered the question of similarity measurement, although not between groups of frames. In particular, Tversky considers situations in which two objects defined by feature sets are to be compared [16]. He argues persuasively that similarity should be determined not just 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:

$$SIMILARITY(A,B) = \theta\, f(A \cap B) - \alpha\, f(A - B) - \beta\, f(B - A)$$

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 that feature sets. $A \cap B$ is analogous to the corresponding slot-value combinations; $A - B$ and $B - A$ would be analogous to the slot value combinations 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 0. If the analogs to $A - B$ and $B - A$ were used with unequal α and β, the measure would be unsymmetric -- one situation may be more similar a second that the second is to it.

Later we will return to the subject of matching stories.

The Matcher works on Physical Laws as well as on Stories

Consider the relationship between the water pressure and water flow in a pipe. It is possible to talk about the kind of thing each is, as well as about how each is related to the other and to the resistance of the pipe. The facts can be represented using the same symbol-arrangement conventions together with a new vocabulary:

> PIPE-LAW is ako constraint - has part pressure-pi flow-pi and resistance-pi - dependent-variable pressure-pi - independent-variable flow-pi - multiplier resistance-pi.
>
> Pressure-pi is ako water-pressure. Water-pressure is ako pressure. Pressure is ako force. Flow-pi is ako water-flow. Water-flow is ako flow. Resistance-pi is ako water-resistance. Water-resistance is ako resistance.
>
> Pressure-pi is proportional-to flow-pi [see proportional-to-pi]. Proportional-to-pi has multiplier resistance-pi.

Matching a specific situation against this constraint is like matching one story against another. Note 3 gives details, showing that matching is successful if a situation just has AKO relations, or just has relations between parts, or both. The note also shows that a specific situation and a general law need not be so closely related, as when an electrical

situation involving a resistor is matched against the water law involving a pipe.

Difficult Situations are Improved by Abstraction

There may be clear clues about how two situations correspond, but no direct evidence. Suppose, for example, that A is proportional to B in one situation while X is determined by Y in the other. Or suppose that A is known to be a kind of force in one situation and X is known to cause something in the other. Matching should be possible because being proportional to something is a way of being determined by something and forces are things that cause.

Note 4 gives the details of how this can be done by using demons to make simple one-step deductions when matching seems too difficult. No changes are made to the matcher itself. The result is the following conclusion:

- Demons make simple abstraction possible. Two situations can match well after abstraction even if they do not match well as they stand.

Match can be Improved by Asking Questions

If abstraction is not sufficient to produce a good match, asking some simple questions may help. Note 5 shows how to generate such questions by using the AKO connections of the parts involved in a situation or by using the relations between parts.

IDENTIFYING ANALOGIES AND CLASSIFICATION-EXPLOITING HYPOTHESIZING

Several identification and hypothesizing mechanisms have been implemented, including the following: matching one situation against a list of possibilities; matching a situation group against a situation group; using a situation to guide a search through a network of possibilities related by SIMILAR-TO; and using a situation to probe into an annotated AKO tree.

Of these, the most important is the mechanism that uses an annotated AKO tree, for it is suggestive of how information retrieval might be done. Hasty readers are advised to skip to the subsection describing it.

Matching one Situation against a List of Possibilities

First, a situation can be matched against a list of given situations and the results are ordered by the matching score. Dull. This identification method is not much use unless there is a limiting context or some method for producing a small number of good hypotheses.

Given a small context or set of hypotheses, the best match can be identified, however. Let us look at a scenario in which Shakespeare's *Macbeth, Hamlet, Othello, Julius Caesar*, and *The Taming of the Shrew* are to be considered. We will see identification become sharper, as illustrated by figure 5, as more is specified. The scenario begins with a statement that two persons are involved:

```
XA is ako story - part Macbeth-xa Duncan-xa.

Macbeth-xa is ako person.  Duncan-xa is ako person.
```

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

```
The matches, in order of quality, are:
2.       100. %  MA
2.       100. %  HA
2.       100. %  OT
2.       100. %  JU
2.       100. %  TA
```

Now we add that one person murders another:

```
Macbeth-xa murder Duncan-xa.
```

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

```
The matches, in order of quality, 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:

```
XA has part Lady-Macbeth-xa.  Lady-Macbeth-xa is ako
person.  Macbeth-xa married-to Lady-Macbeth-xa.
```

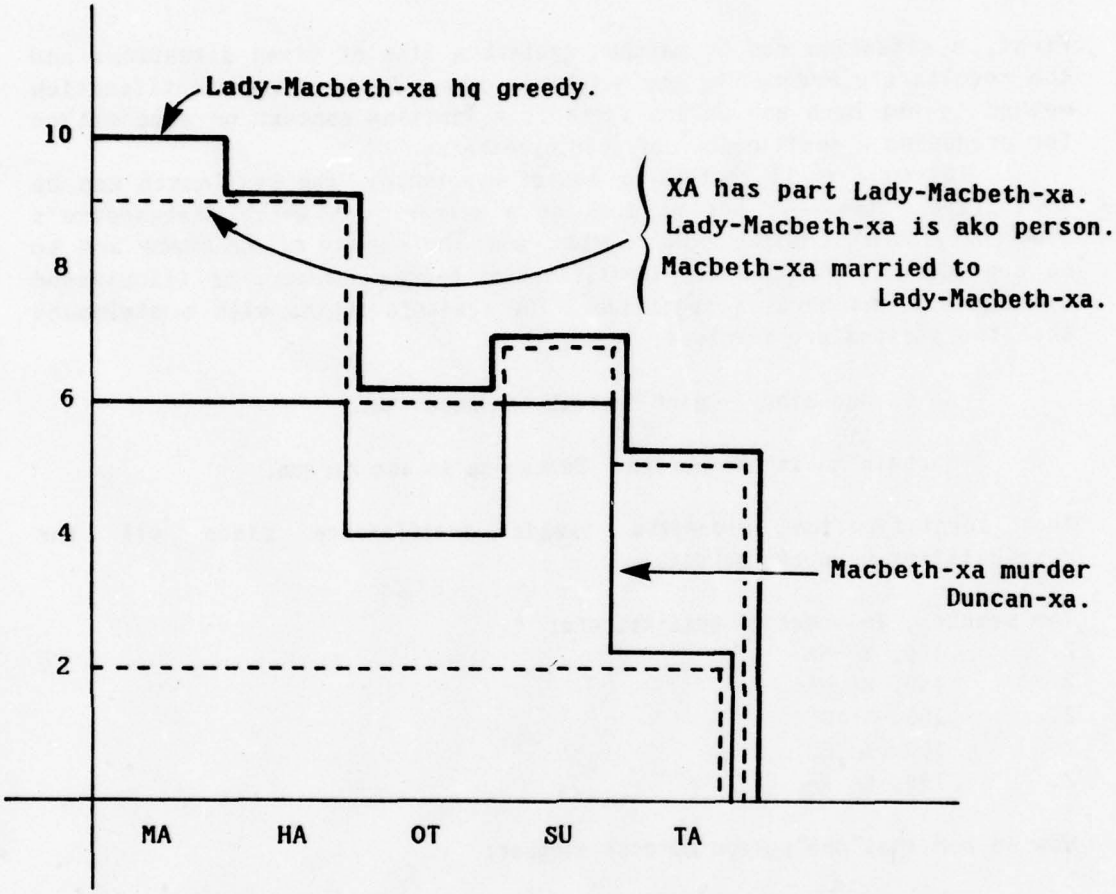


Figure 5: Identification sharpens as more is known. The curve rises as more facts are given.

```
The matches, in order of quality, 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-Macbeth-xa hq greedy.

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

At this point the scenario continues by making a sort of knee-jerk, reflexive prediction that MACBETH-XA will end up unhappy and dead. The details of just how this happens will be explained later.

```
        Considering MA HA OT JU TA
        The most like XA seems to be MA
        I will try a prediction based on it.
         .
         .
         .
        Evidently MACBETH-XA has UNHAPPY in HQ
        Evidently MACBETH-XA has DEAD in HQ
         .
         .
         .
        (MACBETH-XA (PART-OF (XA))
              (AKO (PERSON))
              (MURDER (DUNCAN-XA))
              (KILL (DUNCAN-XA (BECAUSE ((MURDER ASSERTED)))))
              (HQ (EVIL (BECAUSE ((PERSON MURDERS))))
                  (UNHAPPY (INSERTED-BY-RULE (RULE-MA)))
                  (DEAD (INSERTED-BY-RULE (RULE-MA))))
              (MARRIED-TO (LADY-MACBETH-XA)))
```

Matching a Situation Group against a Situation Group

Second, the parts of one group of situations can be identified one after the other with the parts of another group of situations.

This was implemented primarily to handle the problem of 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. 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 this 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 AKO.

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. See Appendix 3 for the actual input.

```
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))
```

Using a Situation to Guide a Search through a Network of Possibilities Related by SIMILAR-TO

Third, a situation can be used to drive a best-first search through a user-prepared network of SIMILAR-TO relations. Search terminates when some user-supplied number of alternatives has been examined.

This is a straightforward implementation of ideas from Winston [19] and Minsky [8]. Regrettably, not much was learned from the implementation. To be sure, a similarity net was constructed from the stories in Appendix 2, but there are not enough of them and they are not similar enough to demonstrate anything. With a data base of the size used, only

illustrations are possible. Yet as it stands there is enough data to make a PDP-10 uncomfortable. The LISP machine will help, of course.

Figure 6 shows the similarity net and the results of some searches in it.

Using a Situation to Probe into an Annotated AKO Tree

Forth, 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 a CONSTRAINED-BY slot. Values in such slots are used to hypothesize likely matching situations. This method has an information-retrieval flavor.

Information is placed in CONSTRAINED-BY slots whenever an AKO relation is placed in a frame that already has a PART-OF slot. In fact, everything in the PART-OF slot is placed in the CONSTRAINED-BY slot of the frames in the AKO slot and all of the ones above in the AKO tree. Suppose, for example, that we start with a *tabula rasa* and supply the following:

> Prince is ako man. Man is ako person. Princess is ako woman. Woman is ako person. Boy is ako man. Girl is ako woman.
>
> CI is ako story - part Charming Cinderella. Charming is ako prince. Cinderella is ako princess. <Plus other facts about Charming and Cinderella>
>
> SN is ako story - part Snow-White. Snow-White is ako princess. <Plus other facts about Snow White>

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

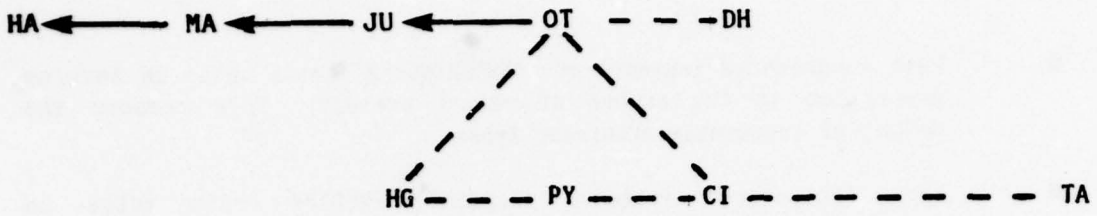
The CONSTRAINED-BY slots are used when looking for stories to match a given situation. The AKO tree standing above each part of the given situation is searched for CONSTRAINED-BY slots. These slots then vote for the stories they contain. Suppose, for example, that the following situation is given:

> RJ is ako situation - part Romeo and Juliet. Romeo is ako boy. Juliet is ako girl. <Plus other facts about Romeo and Juliet>

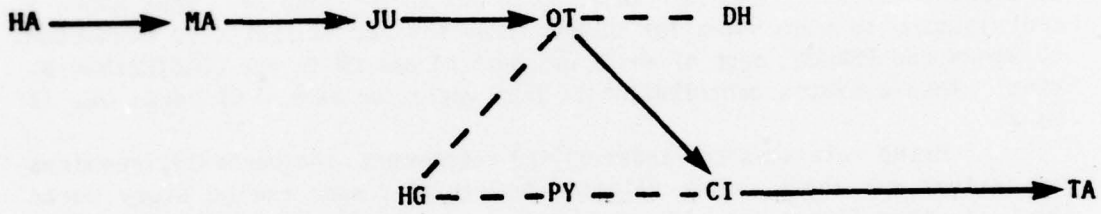
Since Romeo is a boy and Juliet is a girl, BOY, GIRL, MAN, WOMAN, and PERSON are checked for CONSTRAINED-BY values. Three vote for CI and two for SN.

In the actual implementation, the voting is weighted in two ways:

Find HA, starting at OT.



Find TA, starting at HA.



Find HG, starting at TA.

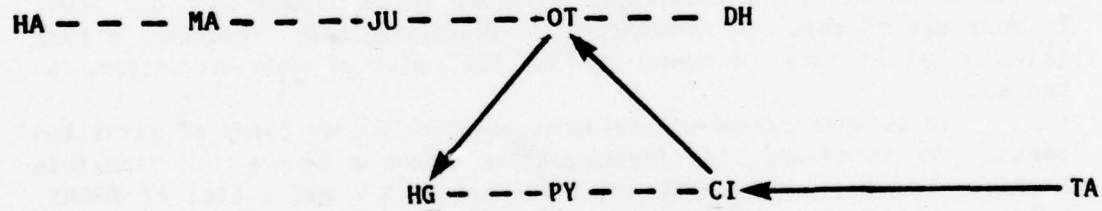


Figure 6: A similarity net and some searches through it. Connections were placed between situations that showed a match score of 30% or more of score of either matched against itself. The structure of the net depended delicately on the choice of this threshold. The key is as follows: HA = *Hamlet*, MA = *Macbeth*, JU = *Julius Caesar*, OT = *Othello*, DH = *A Doll's House*, HG* = *Hedda Gabler*, PY = *Pygmalion*, CI = *Cinderella*, and TA = *Taming of the Shrew*. Note that situations attached directly to situations on path were considered too.

- Each encountered instance of CONSTRAINED-BY casts votes in inverse proportion to the number of values present. This reduces the weight of frequently occurring types.

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

As told, the story RJ has two parts, Romeo and Juliet. Romeo has six slots and Juliet has four. Romeo is connected to MAN, which has only CI in its CONSTRAINED-BY 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 CONSTRAINED-BY 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 story parts just as other frames are. The following would do this for the frame KISS-CI, which appears as a comment in the KISS slot of the CINDERELLA frame:

> Kiss is ako slot. Kiss-ci is ako kiss. Kiss-ci is part-of ci.

CI would end up in the CONSTRAINED-BY slots of the frames KISS and SLOT. To make use of this new information in identifying some situation, a kiss relation would have to appear in the PART slot of that situation, of course.

It is also reasonable to index and retrieve on pairs of situation parts. As it stands, the implementation allows a search for plausible stories 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, stories in which a frame that is a kind of PRINCE has a MARRY slot. I have not thought much about how to do this or even if it should be done.

Whether relations or objects should dominate indexing and retrieval seems to depend on circumstance. Off hand, since there are fewer relation types than object types, it seems likely that the slot hierarchy would become gorged sooner, therefore suggesting that objects are better unless there is some good limiting context.

Appendix 4 gives the results obtained by using the method on each of the stories in Appendix 2. Only the two Ibsen plays gave results in which the story used as a probe was other than the unambiguous first choice. This is not strange. The Ibsen people are just people, men and women, not princes, generals or other distinguishing things.

Similarly, the XA version of Macbeth, used before, gives indifferent results, since it too has no useful information in the AKO slots of the people.

It is not Clear if the Classification-exploiting Method Scales Well

In the end, the gorging question must be looked at to determine if the AKO-based identification method is robust enough to be useful when the number of stories increases to a practical size.

The work of Rosch et al is probably relevant [14]. They argue that the world of human experience is such that there is a so-called basic level of class abstraction in the AKO 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.

Importantly, people tend to recognize and specify concepts at the basic level first, according to their results. This might mean that the most useful situation-hypothesizing information would be at that level since people specify things only at that level and since the higher level evidence would be far more ambiguous.

If there are only, say, a thousand basic types, 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 unevenly filled by the situations that are useful. A better analysis or some experimentation is needed, therefore, to see if the AKO-based identification method will work in practical situations.

Now let us look at the subject of abstraction as it impacts on the matching that follows hypothesis generation.

Demons can do Abstraction

The matching part of identification often requires some abstraction of the relations involved. The fact that demons can do abstraction was made in the discussion of matching. The point deserves emphasis.

Suppose two abstract situations are proposed, one involving a tragic event, and the other, a person in conflict with himself:

> TE is ako situation - part evil-person good-person.
>
> Evil-person is ako person. Good-person is ako person. Evil-person hurt Good-person. Evil-person hq evil. Good-person hq good.

IS is ako situation - part Grubbla.

Grubbla is ako person. Grubbla has-conflict-with Grubbla.

As they stand, neither has anything at all in common with the stories in Appendix 2, 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.

Abstraction demons make good matches happen anyway. Existing demons on HQ and MURDER arrange for people to have GOOD and EVIL qualities. The following ones add machinery for generating values in HURT and HAS-CONFLICT-WITH slots.

Kill has if-added (fpfsv frame 'hurt value).
Hurt has if-added (fpfsv frame 'has-conflict-with value).

Given these, we can look for matches in the stories of Appendix 2, with the following results, first for the tragic event situation:

The matches, in order of quality, are:

5.	83. %	MA
5.	83. %	HA
5.	83. %	OT
5.	83. %	JU
4.	66. %	DH
4.	66. %	PY
4.	66. %	AD
3.	50. %	CI
2.	33. %	TA
2.	33. %	HG

The act of murder made Macbeth an evil person and established that he hurt Duncan. The story lacks perfect match with the description of 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 Desdamona, but to kill does not imply a person is evil as murder does. Desdamona, 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. %  HA
2.      100. %  OT
2.      100. %  JU
2.      100. %  HG
1.      50. %   MA
1.      50. %   TA
1.      50. %   DH
1.      50. %   PY
1.      50. %   CI
1.      50. %   AD
```

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 incidents, at least as described:

> IR is ako situation - part unfortunate-person desired-act actual-act.
>
> Unfortunate-person is ako person. Unfortunate-person want desired-act - attempt desired-act [see attempt-ir]. Attempt cause actual-act. Desired-act opposite actual-act.

Expansion goes toward Abstraction and toward Detail

Some demons expanded situations by adding details. Just now others expanded situations by doing abstraction. Thus I take the following stand on canonical representation:

- There should be no obsession with canonical form. If two situations do not match well, but should, then the right approach is to expand them until they show signs of similarity. This may be in the direction of very basic primitives or very abstract generalities.

REACHING CONCLUSIONS USING A FRAME-ORIENTED RULE SYSTEM

Having identified a situation with a story or constraint, the next thing is to use it. One approach would simply look for conclusions in the story or constraint and map them into the situation at hand.

Another approach leads to the development of frame-based situation-action rules. There are two arguments for this approach: one is that such rules constitute a more explicit representation of what is to happen in situations involving algebraic knowledge; and the other is that if-added demons can be viewed as simple special cases of such rules. It is not clear that this second approach is better. It was followed.

Inasmuch as there is considerable detail involved in explaining just how this works, some readers may wish to jump to the next section to see how verification of conclusions is done, particularly in legal cases.

Story and Constraint Frames Provide Access to Situation-action Rules

To reach a conclusion, two things are necessary: first it must be possible to identify the things that are constrained and the role that each thing has with respect to the situation and second, it must be possible to do deductions, given that the situation's parts and their roles are known.

To express rules, a rule representation must provide mechanisms for finding things in the slots of the frame describing some particular situation. Then it must further provide mechanisms for placing new things in other slots. To meet these requirements, situation-action rules can be represented as frames consisting of two slots, the SITUATION slot and the ACTION slot. The best way to explain how the rule interpreter uses the information in these slots is to work through an example.

Suppose that there is some particular combination of water pressure, water flow, and pipe resistance. Further suppose that these are assembled together to form X1, a frame that resembles the PIPE-LAW frame:

> PIPE-LAW is ako constraint - has part pressure-pi flow-pi and resistance-pi - dependent-variable pressure-pi - independent-variable flow-pi - multiplier resistance-pi.
>
> X1 is ako situation - has part pressure-X1 flow-X1 and resistance-X1 - dependent-variable pressure-X1 - independent-variable flow-X1 - multiplier resistance-X1.

Now there is to be a rule specifying that pressure is determined by multiplying resistance times flow if values for them are known. Let us call the rule for this RULE-PI. We augment PIPE-LAW by placing RULE-PI in its RULE slot:

> Pipe-law has rule rule-pi.

RULE-PI looks like this:

```
(RULE-PI (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, X1. Consider the frame IF1, for example:

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

The > notation means find the thing that occupies the corresponding slot of the situation frame and shove it into the variable. Thus matching the first IF frame against X1 yields PRESSURE-X1 as the value of X; FLOW-X1 as the value of Y; and RESISTANCE-X1 as the value of C.

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

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

The < notation means pull the value out of the variable. Thus this IF frame is to be compared with RESISTANCE-X1. If RESISTANCE-X1 has something in the QUANTITY slot, it becomes the value of CQ. Similarly if FLOW-X1 has something in its QUANTITY slot, it becomes the value of YQ by way of IF frame IF3:

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

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 PRESSURE-X1, the frame that was dug out of X1 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.

All this taken together means that the rule interpreter can use situation-action rule RULE-PI together with the situation described by X1, PRESSURE-X1, RESISTANCE-X1, and FLOW-X1 to place new information in PRESSURE-X1's QUANTITY slot. Thus RULE-PI and PIPE-LAW together describe a constraint. Figure 7 shows how applying a rule is a bit like working with a template.

It is convenient to have a simple way to translate simply written situation-action rules into frames of the sort shown in the example. Note 2 shows a transition network describing a translator that does the job. The translator accepts RULE-PI in the following form:

```
RULE rule-pi IF situation has dependent-variable >x -
multiplier >c - independent-variable >y. <c quantity >cq.
<y quantity >yq. THEN <x quantity = (times <cq <yq). END
```

To summarize, constraints are represented as constraint frames that provide access to situation-action rules and that specify what a situation frame must look like to trigger the rules. For the moment, one question is left dangling: how is a frame describing the situation made to look like the constraint frame?

Copying is Initiated once Matching Provides Links

For RULE-PI to be applied to a particular situation, say X1, the frame for X1 must be made to look like PIPE-LAW so that there can be hope that the situation part of RULE-PI will match it. Suppose, for example, that X1 is a situation only known to involve three things constrained by the pipe law:

```
(X1 (AKO (SITUATION))
    (PART (PRESSURE-X1) (RESISTANCE-X1) (FLOW-X1)))
```

Before RULE-PI has a chance, situation X1 must be transformed into this:

```
(X1 (AKO (SITUATION))
    (PART (PRESSURE-X1) (RESISTANCE-X1) (FLOW-X1))
    (DEPENDENT-VARIABLE (PRESSURE-X1))
    (INDEPENDENT-VARIABLE (FLOW-X1))
    (MULTIPLIER (RESISTANCE-X1)))
```

This is done by a two-step procedure: first, the parts in X1 are matched against the parts in PIPE-LAW; and second, most of the structure of PIPE-LAW is copied into X1 with substitutions determined by the matching step of the procedure. Let us look at both steps in more detail.

First, PRESSURE-X1, FLOW-X1, and RESISTANCE-X1 of X1 are matched against PRESSURE-PI, FLOW-PI, and RESISTANCE-PI of PIPE-LAW. The match is straightforward given suitable information about what the parts of X1 are

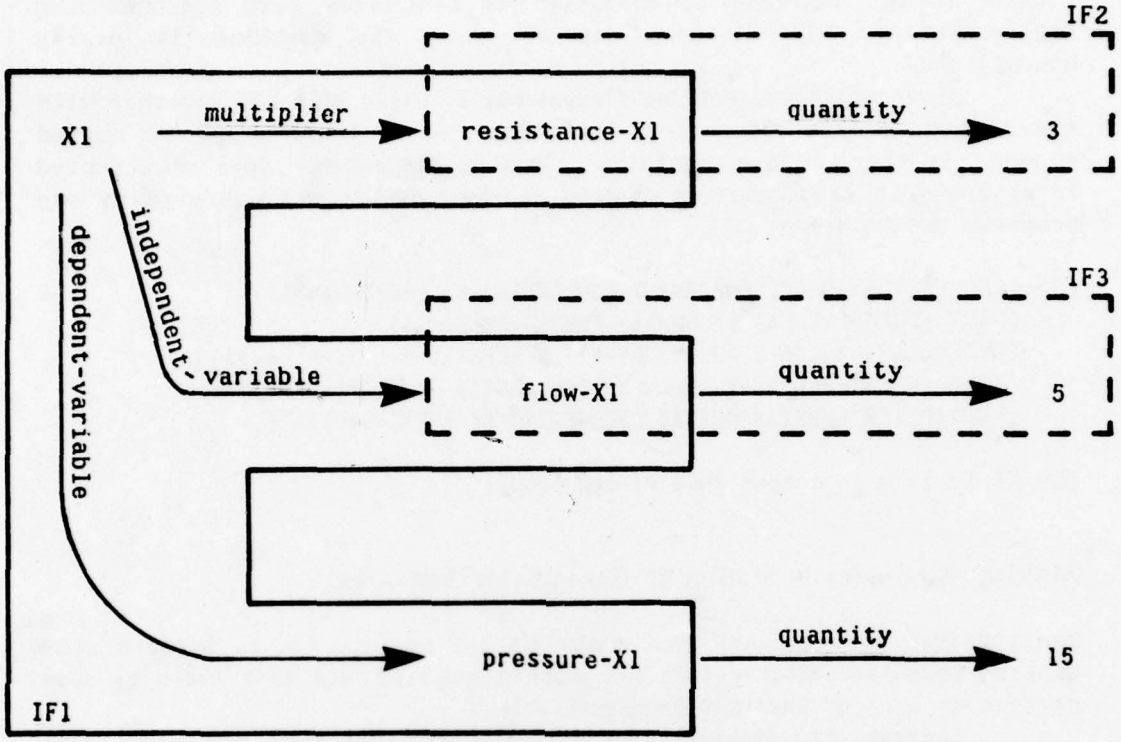


Figure 7: Situation-action rules are invoked when a sequence of template-like tests fits over a situation description. Here IF1 is the first test, and it fits X1. In addition, it identifies where IF2 and IF3 are to be applied. Success calls for altering PRESSURE-X1 by establishing its value.

or how they are related or both. Recall, for example, the water pipe situations S1, S2, and S3 discussed in connection with the matching implementation. If X1 were any of them, the matching is easily accomplished.

After matching, copying fleshes out X1 using PIPE-LAW together with the list of paired frames from the match. Slots in PIPE-LAW are copied into X1 if they contain something paired by the match. Note that copied information is commented for clarity -- the comments are not used by any programs at the moment.

```
(X1 (AKO (SITUATION) (CONSTRAINT (SUGGESTED-BY (PIPE-LAW))))
    (PART (PRESSURE-X1) (FLOW-X1) (RESISTANCE-X1))
    (DEPENDENT-VARIABLE (PRESSURE-X1 (SUGGESTED-BY (PIPE-LAW))))
    (INDEPENDENT-VARIABLE (FLOW-X1 (SUGGESTED-BY (PIPE-LAW))))
    (MULTIPLIER (RESISTANCE-X1 (SUGGESTED-BY (PIPE-LAW)))))
```

Now X1 is in a form that RULE-PI can handle.

Making Analogies is Similar to Recognizing Instances

Now consider what happens when a student has no idea how to perform some desired computation other than the teacher-supplied fact that there is some particular analogy that may be exploitable.

Suppose, for example, that a student does not know Ohm's law. The student does know, from the teacher, that there is a voltage, resistance, and current together with a suggestion to think in terms of the pipe law:

> X2 is ako situation - has part voltage-x2 current-x2 and resistance-x2.

If this were a water-pipe situation, specific variables would be matched against the general variables in PIPE-LAW as before. But now, since the teacher has claimed an analogy between a specific electrical situation and the general water law, the match must be between the electrical things, VOLTAGE-X2, CURRENT-X2, and RESISTANCE-X2, and the water things, PRESSURE-PI, FLOW-PI, and RESISTANCE-PI.

Still, the match is straightforward, given suitable information about what the parts of X2 are or how they are related or both. In particular, there is no problem if the parts of X2 correspond to those in any one of the electrical situations, S4, S5, and S6, discussed in connection with the matching implementation.

The match binds VOLTAGE-X2 to PRESSURE-PI, CURRENT-X2 to FLOW-PI, and RESISTANCE-X2 to RESISTANCE-PI. This enables the structure found in the PIPE-LAW frame to be copied into the X2 frame with match-determined substitutions:

```
(X2 (AKO (SITUATION) (CONSTRAINT (SUGGESTED-BY (PIPE-LAW))))
    (PART (VOLTAGE-X2) (CURRENT-X2) (RESISTANCE-X2))
    (DEPENDENT-VARIABLE (VOLTAGE-X2 (SUGGESTED-BY (PIPE-LAW))))
    (INDEPENDENT-VARIABLE (CURRENT-X2 (SUGGESTED-BY (PIPE-LAW))))
    (MULTIPLIER (RESISTANCE-X2 (SUGGESTED-BY (PIPE-LAW)))))
```

The purpose of all this has been to create the DEPENDENT-VARIABLE, MULTIPLIER, and INDEPENDENT-VARIABLE slots and to place the proper things in them. Now the situation part of RULE-PI can apply as before with X1.

Note that the matching operation is robust. Saying a lot more about the particular, incidental properties of the electrical variables typically would not matter at all, since typically there would be no corresponding incidental properties for the abstractions of PIPE-LAW.

Both Specific and General Laws can be Discovered

As we solve problems using analogies, we gain experience that enables us to formulate new laws. These new laws make it easier and quicker to reach conclusions.

Recall that a constraining law consists of a descriptive frame together with one or more situation-action rules that trigger if the descriptive frame is successfully matched to a particular situation description.

Note 6 shows how to learn the descriptive-frame part of a law. It first demonstrates that the descriptive frame for Ohm's law can be discovered after two electrical situations are enriched by way of the water-pipe analogy. It then demonstrates that the descriptive frame for the general force-and-flow law can be discovered by combining Ohm's law and the water pipe law. Both demonstrations use exactly the same program.

The Situation-Action Rules can be Learned

Note 7 shows how to learn the situation-action rules involved in new laws.

If-needed Demons are a Special Case of the Situation-Action Rules

If-needed demons are important since they enable the simple abstractions that make possible the more interesting matches. Thankfully, the if-needed demons can be learned using the same methods that apply to other analogies. If-needed demons are just smaller. Note 8 supplies details and examples.

TESTING CONCLUSIONS USING AN EXPERIENCE-DRIVEN VERIFIER

The situation-actions rules are a bit too eager to use the match. There really is no need to be quite so sophomoric:

- The cause relationships in a known story suggest the right relations to check and the right questions to ask.

Assuming that the cause relations in the known story are reasonable and are reasonably complete, it makes sense to use them to check the validity of making conclusions about the situation being analyzed. If a particular fact in a known story is justified by various other facts, then the corresponding fact in the situation being analyzed may have a similar justification.

The Cause Relations in Stories make it Possible to Check Conclusions

To be more precise, the following steps are taken in the current implementation to check a fact in a given situation, given the cause and cause-related slots in a known story:

- First, the fact in question may actually be in the situation being analyzed. If so, no further action is needed.
- Second, the fact may correspond to one in the known story that has a person in its DESIRED-BY slot. If so, ask if the corresponding person in the situation has sufficient desire to cause the fact to come to be.
- Third, the fact may correspond to one in the known story that has a person in its CAUSED-BY slot. If so, ask if the corresponding person in the situation causes the fact in the situation being analyzed.
- Fourth, the fact may correspond to one in the known story that has another fact in its CAUSED-BY slot. If so, try to check that other fact in the situation by recursion.
- Fifth, the fact may correspond to one in the known story that has another fact in its REASON slot. If so, try to check by recursion.
- Sixth, repeat with the MOTIVE slot.

If the slots in the known story have more than one value, all must be verified. If any PREVENTED-BY slots are encountered along the way, they

are investigated as follows:

- If there is a person in the PREVENTED-BY slot, ask if the corresponding person in the situation prevents the fact from coming to be.

- If there is a fact in the PREVENTED-BY slot, check it by recursion.

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

To make use of all the cause-related rules, it is necessary to go into a careful mode that generates a comment for every slot-value combination that does not already have one (with three exceptions). Three slot are then automatically filled: FRAME, SLOT, and VALUE. The reason is to have a way of getting to the frame, the slot, and the value for every relation.

Furthermore, it is necessary to add some demons that tie things together with cause relations. See Appendix 5 for the details.

To see how all this works out, consider the following version of a Macbeth story:

```
XB is ako story - part Macbeth-xb Duncan-xb Lady-Macbeth-xb
Macduff-xb.

Man has instance Macbeth-xb Duncan-xb Macduff-xb.  Woman
has instance Lady-Macbeth-xb.

Macbeth-xb married-to Lady-Macbeth-xb.  Lady-Macbeth-xb hq
greedy.  Macduff-xb hq loyal.
```

As it stands, there is barely enough said to do an unambiguous match to Macbeth, but given that a teacher insists, it makes sense to question the validity of the rule asserting that the tragic figure should die.

Figure 8 shows the cause connections in Macbeth. The following shows what the verifier does with them:

```
(CHECK 'MACBETH-XB 'HQ 'DEAD 'XB 'MA)

Matching XB and MA
 ......

Does LADY-MACBETH-XB cause the MURDER slot of MACBETH-XB to
have DUNCAN-XB in it?

> NO (meaning the user does not know)

Does  MACBETH-XB  sufficiently  desire  the  AKO  slot  of
```

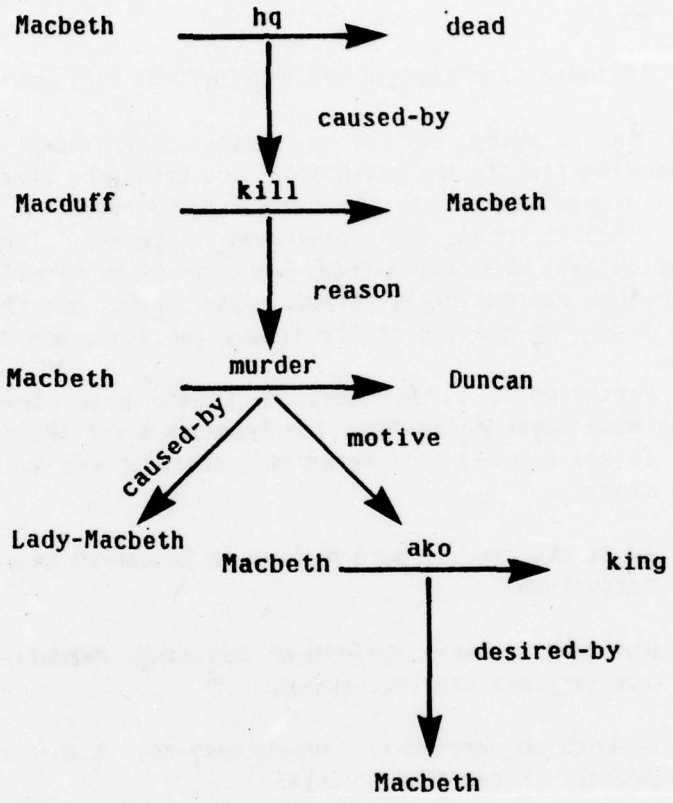


Figure 8: The cause connections in Macbeth. If Macbeth is matched with a situation having no connections, then asking if the Macbeth person dies leads to asking questions determined by the causes, reasons, motives, and desires. The first question is about the Lady Macbeth person and the cause of a murder. If the answer is unknown, the second question is about the Macbeth person and whether he desires to be king.

MACBETH-XB to have KING in it?

> YES (assumed for illustration)

Evidently there is sufficient DESIRE for the AKO slot of MACBETH-XB to have KING in it.

Evidently there is sufficient MOTIVE for the MURDER slot of MACBETH-XB to have DUNCAN-XB in it.

Evidently there is sufficient REASON for the KILL slot of MACDUFF-XB to have MACBETH-XB in it.

Evidently there is sufficient CAUSE for the HQ slot of MACBETH-XB to have DEAD in it.

[MACBETH-XB HQ DEAD] in XB is verified by the precedent in MA

It makes sense.

Legal Situations Illustrate the Need for Attention to Details of Cause

Dealing with legal precedent seems like a combination of situation identification and cause verification. Since legal precedents are carefully stated stories, they should be susceptible to the identification and verification methods already given.

Several legal situations were recorded to get a feel for what is involved, particularly with respect to the verification issue. 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 [7]. The actual input form for these is given in Appendix 6. Here are approximations 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 system thinks about whether these are cases of assault and battery. The following scenarios are invoked:

```
(CHECK GI 'AKO 'ASSAULT GI AS)

Matching GUILTY-VS-INNOCENT and ASSAULT1
 ..

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 AKO slot of
GUILTY-VS-INNOCENT to have ASSAULT in it.

[ GUILTY-VS-INNOCENT AKO 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 GI 'AKO 'BATTERY GI BA)

Matching GUILTY-VS-INNOCENT and BATTERY1
 ..

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 AKO slot of
GUILTY-VS-INNOCENT to have BATTERY in it.

[ GUILTY-VS-INNOCENT AKO 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 VV SW)

Matching VILLAIN-VS-VICTIM and ADAMS-VS-ZENT
 ....

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 VV SW)

Matching VILLAIN-VS-VICTIM and ADAMS-VS-ZENT
 ....

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 VV AZ)

Matching VILLAIN-VS-VICTIM and SMITH-VS-WESSON
 ....

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

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

Evidently the KNOW slot of VICTIM is PREVENTED FROM having
[ PISTOL HQ 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 VV AZ)

Matching VILLAIN-VS-VICTIM and SMITH-VS-WESSON
 ....

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 VV 'AKO 'ASSAULT VV AS)

Matching VILLAIN-VS-VICTIM and ASSAULT1
 ....
```

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 AKO slot of VILLAIN-VS-VICTIM to have ASSAULT in it.

[VILLAIN-VS-VICTIM AKO ASSAULT] in VILLAIN-VS-VICTIM is verified by the precedent in ASSAULT1

(CHECK VV 'AKO 'BATTERY VV BA)

Matching VILLAIN-VS-VICTIM and BATTERY1

....

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 AKO slot of VILLAIN-VS-VICTIM to have BATTERY in it.

[VILLAIN-VS-VICTIM AKO 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 AKO relations can be the result of a successful match. Second, that a situation may be analyzed by way of analogy with several already known stories. 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. See figure 9.

CONCLUSION: SIMPLE MECHANISMS ENABLE LEARNING

This paper is about a set of ideas that enable learning to take place by analogy in domains that adhere to certain principles, namely:

- Symbolic sufficiency.
- Description-determined similarity.
- Cause-determined importance.
- Historical continuity.

The most important of the ideas that make learning possible in such domains

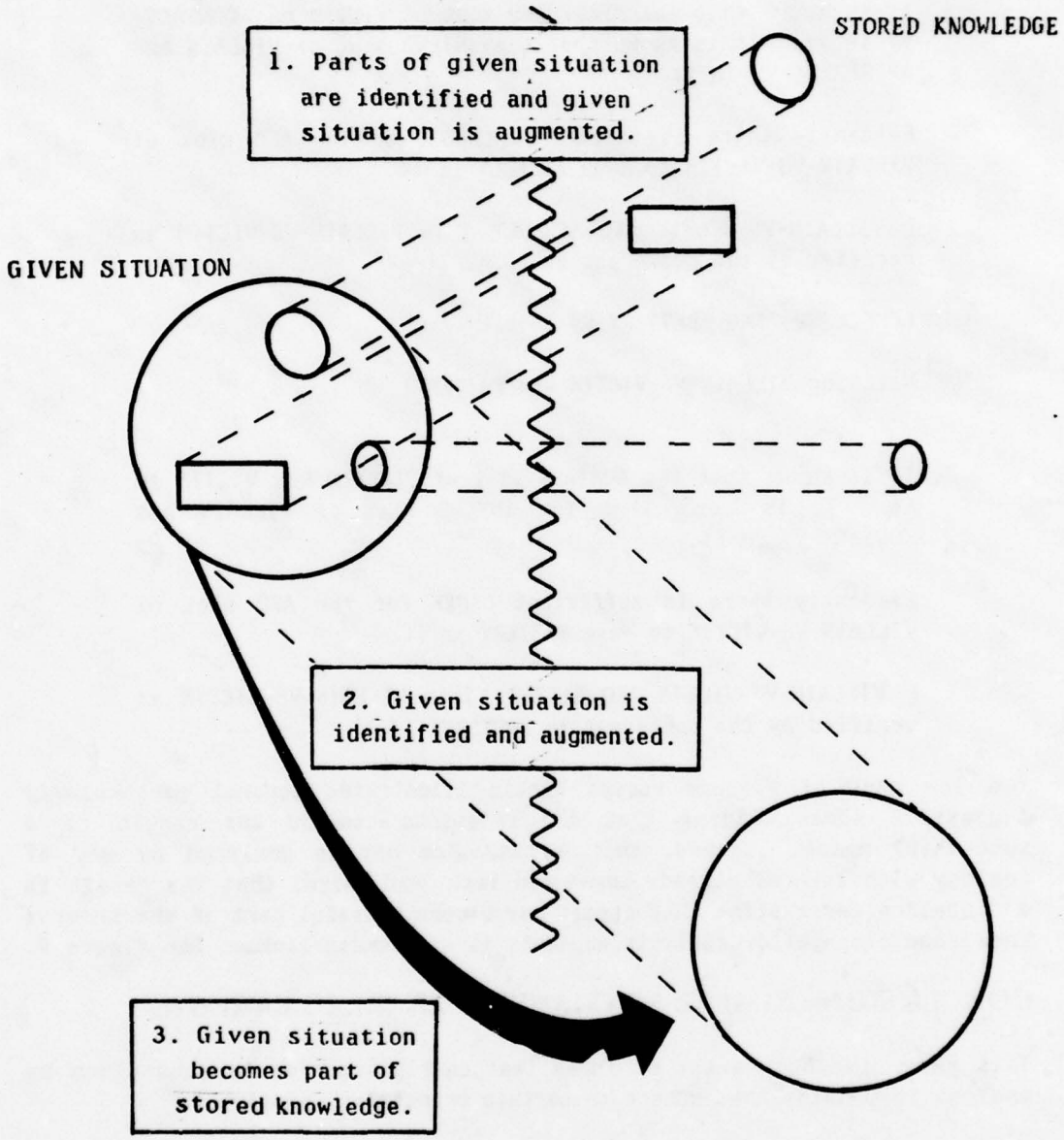


Figure 9: The system builds on its own results. Identification and correct reaction to part of a description enables further identification and reaction.

are these:

- Commented-frames representation.
- Classification-exploiting hypothesizing.
- Cause-dominated matching.
- Frame-oriented, rule-based deduction.
- Experience-driven verification.

It is now clear that simple situations can be analyzed by attacking them with the analogy process using stored things that lie all along the spectrum from if-added demons to complex cases. Once analyzed, a situation becomes eligible itself for use in analyzing newer, perhaps bigger situations. Experience makes an analogy-making system smarter, or at least more experienced.

ACKNOWLEDGMENTS

This paper was 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. The drawings are by Karen Prendergast.

REFERENCES

1. Richard Henry Brown, "Use of Analogy to Achieve New Expertise," Master's thesis, M.I.T. Artificial Intelligence Laboratory Technical Report No. 403, April 1977.

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

3. Thomas G. Evans, "A Heuristic Program to Solve Geometric Analogy Problems," PhD thesis, in *Semantic Information Processing*, edited by Marvin Minsky, The M.I.T. Press, Cambridge, Massachusetts, 1968.

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

5. Douglas Lenat, "AM: An Artificial Intelligence Approach to Discovery in

Mathematics as Heuristic Search," PhD thesis, to be published in Randall Davis and Douglas Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, 1979.

6. William Martin, "Philosophical Foundations for a Linguistically Oriented Semantic Network," in preparation.

7. J. Meldman, A preliminary Study in Computer-Aided Legal Analysis, PhD thesis, M.I.T. Laboratory for Computer Science Technical Report No. MAC-TR-157, November, 1975.

8. Marvin Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, edited by Patrick Henry Winston, McGraw-Hill Book Company, New York, 1975.

9. J. Moore and Allen Newell, "How can Merlin Understand?" in *Knowledge and Cognition*, edited by L. Gregg, Lawrence Erlbaum Associates, Potomac, Maryland, 1974.

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

11. Chuck Rieger, "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.

12. R. Bruce Roberts and Ira P. Goldstein, "The FRL Primer," M.I.T. Artificial Intelligence Laboratory Memo No. 408, July 1977.

13. R. Bruce Roberts and Ira P. Goldstein, "The FRL Manual," M.I.T. Artificial Intelligence Laboratory Memo No. 409, June 1977.

14. Eleanor Rosch, Carolyn B. Mervis, Wayne D. Gray, David M. Johnson, and Penny Boyes-Braem, "Basic Objects in Natural Categories," *Cognitive Psychology*, 8, 1976.

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

16. Amos Tversky, "Features of Similarity," *Psychological Review*, volume 84, number 4, July, 1977.

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

18. Patrick Henry Winston, "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.

19. Patrick Henry Winston, "Learning by Creating and Justifying Transfer Frames," *Artificial Intelligence*, Vol. 10, 1978, pp. 147-172.

APPENDIX 1

Note 1: There are Alternatives to Commented Slot-value Combinations

This note describes three alternative representations and why they were rejected in favor of comment slot-value combinations.

- 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-CI)))

(FIND-CI (AKO FIND)
         (AGENT (CHARMING))
         (OBJECT (CINDERELLA))
         (INSTRUMENT (SLIPPER-CI)))
```

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 people could be matched with another. Using this alternative strategy, correspondence has to be established between relation frames as well between the people frames.

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

```
(CHARMING (FIND-CI (CINDERELLA)))

(FIND-CI (AKO FIND)
         (AGENT (CHARMING))
         (OBJECT (CINDERELLA))
         (INSTRUMENT (SLIPPER-CI)))
```

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-CI frame. Similarly, matching still seems to require extra work.

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

```
(CHARMING (FIND (VALUE (CINDERELLA))
                (AGENT (CHARMING))
                (OBJECT (CINDERELLA))
```

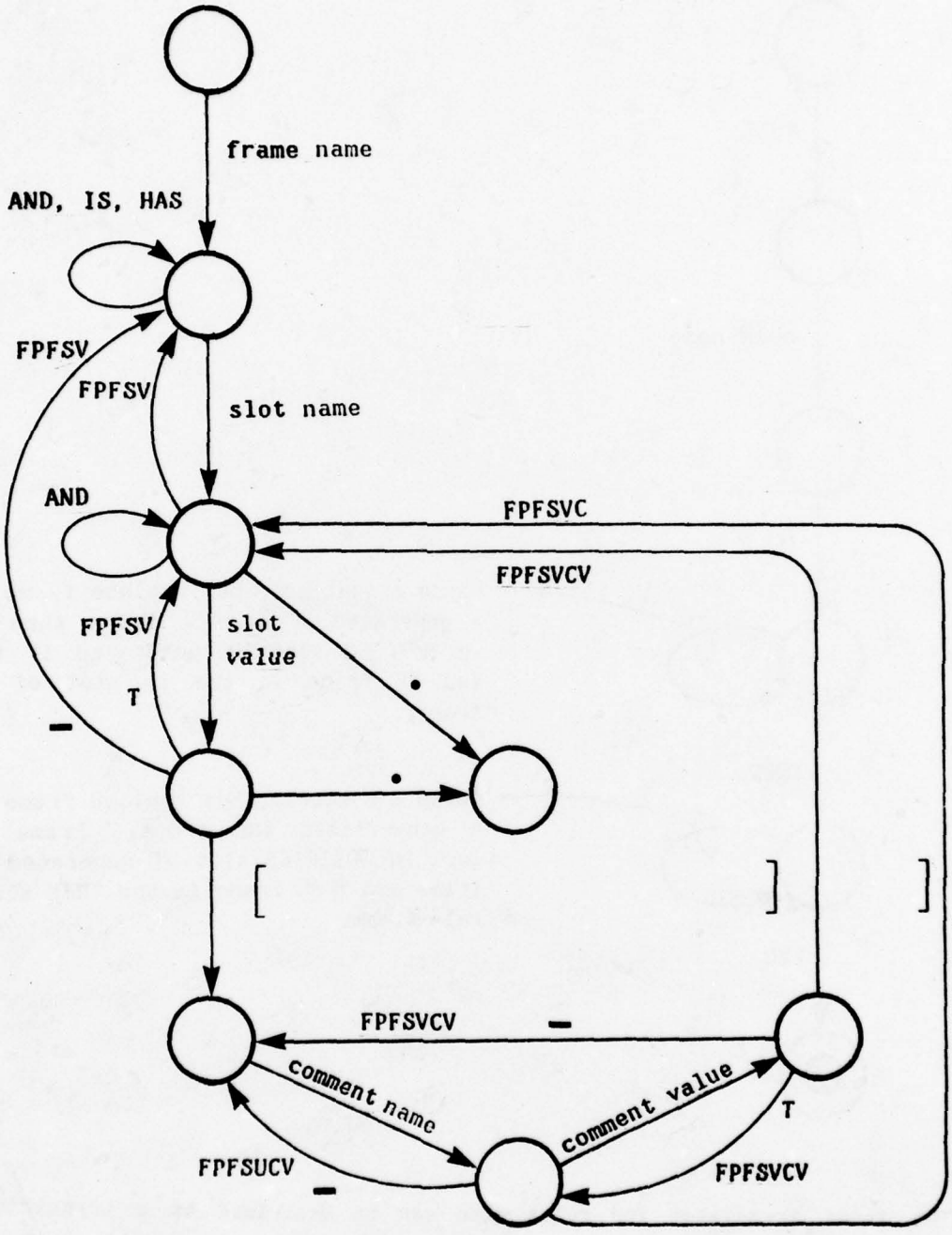
PRECEDING PAGE NOT FILMED
BLANK

(INSTRUMENT (SLIPPER-CI))))

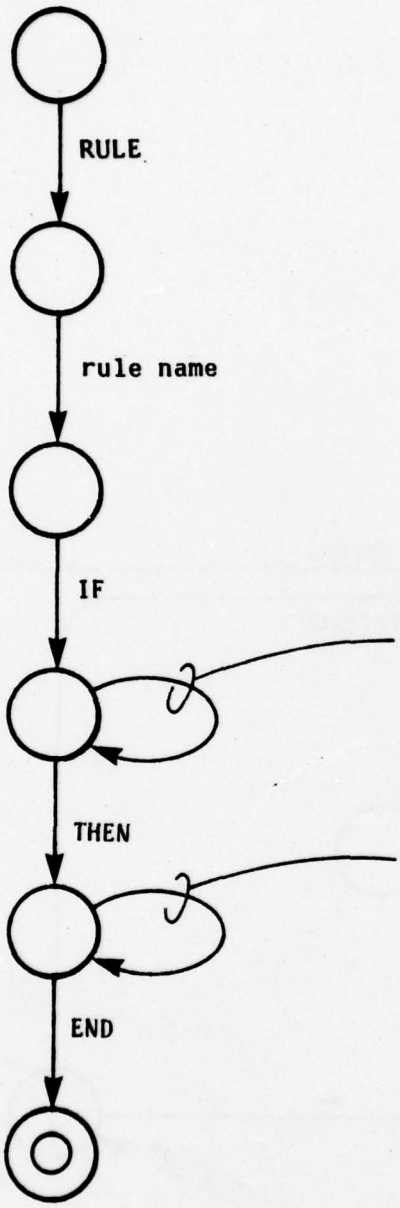
Unfortunately anything in the other facets would normally apply to all values in the value facet. Moreover, it does not feel quite right to mix supplementary description with the demons that are normally found in the facets.

In contrast, using commented slots is perspicuous, does not interfere with matching, and leaves the facets alone.

Note 2: Transition Networks Define an Input Representation



The input translator can be described as a transition network. Most of the labels on the arcs specify how the atoms in the input stream become associated with associated with arguments to value and comment inserting functions. Arcs with the labels ., [,], and - determine how syntax guides the association. The atoms AND, IS, and HAS are ignored. FPFSV means place a value using the current associations. FPFSVCV means place a value with a comment, again with the current associations.



Parse a sentence, but replace frame with a generated IF frame. Frame then goes in *MATCHES slot of generated IF frame* and IF frame in the IF slot of rule frame.

Parse a sentence, but replace frame with a generalized THE frame. Frame then goes in MODIFIES slot of generated THEN frame and THEN frame in the THEN slot of rule frame.

The input translator for rules also can be described as a transition network.

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

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

> PIPE-LAW is ako constraint - has part pressure-pi flow-pi and resistance-pi - dependent-variable pressure-pi - independent-variable flow-pi - multiplier resistance-pi.
>
> Pressure-pi is ako water-pressure. Water-pressure is ako pressure. Pressure is ako force. Flow-pi is ako water-flow. Water-flow is ako flow. Resistance-pi is ako water-resistance. Water-resistance is ako resistance.
>
> Pressure-pi is proportional-to flow-pi [see proportional-to-pi]. Proportional-to-pi has multiplier resistance-pi.

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

> S1 is ako situation - has part pressure-s1 flow-s1 and resistance-s1.
>
> Pressure-s1 is ako water-pressure. Flow-s1 is ako water-flow. Resistance-s1 is ako water-resistance.

The result is announced as follows by the matcher:

> Matching PI and S1
> .....
>
> Score = 7. versus 10. and 7. Match is decisive -- 4. better than next best.

The score is entirely dependent on the AKO slots. Both pressure-pi and pressure-s1 are a kind of water pressure, pressure, and force. Flow-pi and flow-s1 are a kind of water-flow and flow. And resistance-pi 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 types are established. Moreover, giving the AKO relations seems to be giving away too much.

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

S2 is ako situation - has part pressure-s2 flow-s2 and resistance-s2.

Pressure-s2 is proportional-to flow-s2 [see proportional-to-s2].

Proportional-to-s2 has multiplier resistance-s2.

The matcher gives the result as follows:

Matching PI and S2
.....

Score = 3. versus 10. and 3. Match is decisive -- 1. better than next best.

The score now depends entirely on the PROPORTIONAL-TO and MULTIPLIER slots. Pressure-pi and pressure-s2 are proportional to flow-pi and flow-s2. A demon attached to PROPORTIONAL-TO similarly established that flow-pi and flow-s2 are proportional to pressure-pi and pressure-s2. And finally, the comment frames PROPORTIONAL-TO-PI and PROPORTIONAL-TO-S2 have resistance-pi and resistance-s2 in their MULTIPLIER slots.

Of course, if both relations and types are known, as in the following situation, 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 PI and S3
.....

Score = 10. versus 10. and 10. Match is decisive -- 6. better than next best.

Here are the matching results for all three situations:

SITUATION	SCORE	BETTER THAN NEXT BEST MATCH
S1 (types only)	7	4
S2 (relations only)	3	1
S3 (both)	10	6

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:

S4 is ako situation - has part voltage-s4 current-s4 and resistance-s4.

Voltage-s4 is ako voltage. Voltage is ako force. Current-s4 is ako electric-current. Electric-current is ako flow. Resistance-s4 is ako electric-resistance. Electric-resistance is ako resistance.

The result is:

Matching PI and S4
.....

Score = 3. versus 10. and 6. Match is decisive -- 2. better than next best.

The score is three for S4 rather than seven, as it was for the parallel S1, because the AKO 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-pi than pressure-s1 is, and the same is true for current-s4 and resistance-s4 relative to flow-s1 and resistance-s1.

Situations S5 and S6 parallel S2 and S3:

S5 is ako situation - has part voltage-s5 current-s5 and resistance-s5.

Voltage-s5 is proportional-to current-s5 [see proportional-to-s5]. Proportional-to-s5 has multiplier resistance-s5.

Matching PI and S5
.....

Score = 3. versus 10. and 3. Match is decisive -- 1. better than next best.

S6 is ako situation - has part voltage-s6 current-s6 and resistance-s6.

Voltage-s6 is ako voltage. Current-s6 is ako electric-current. Resistance-s6 is ako electric-resistance. Voltage-s6 is proportional-to current-s6 [see proportional-to-s6]. Proportional-to-s6 has multiplier resistance-s6.

Matching PI and S6
.....

Score = 6. versus 10. and 9. Match is decisive -- 3. better than next best.

Here is the summary:

SITUATION	SCORE	BETTER THAN NEXT BEST MATCH
S4 (types only)	3	2
S5 (relations only)	3	1
S6 (both)	6	3

On the whole, the correspondence is not as solid or unambiguous, but the correct correspondence is still found.

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 PI and a particular situation, D1:

> AKO has inverse instance. Proportional-to has inverse proportional-to. Increases-with has inverse increases-with. Determined-by has inverse determined-by.
>
> PI is ako constraint - has part pressure-pi flow-pi and resistance-pi.
>
> Pressure-pi is ako water-pressure. Water-pressure is ako pressure. Pressure is ako force. Flow-pi is ako water-flow. Water-flow is ako flow. Resistance-pi is ako water-resistance. Water-resistance is ako resistance. Pressure-pi is proportional-to flow-pi [see proportional-to-pi]. Proportional-to-pi has multiplier resistance-pi.
>
> D1 is ako situation - has part x-d1 y-d1 and z-d1.
>
> X-d1 cause y-d1 - determined-by y-d1 [see determined-by-d1]. Determined-by-d1 has parameter z-d1.

As they stand, these descriptions lead to the following frames, with PI and D1 showing no correspondence whatsoever. The frames in D1 have no AKO

slots at all, and they have no other slots that correspond to slots in the frames of PI.

```
(PRESSURE-PI (AKO (WATER-PRESSURE))
             (PROPORTIONAL-TO (FLOW-PI (SEE (PROPORTIONAL-TO-PI)))))
(FLOW-PI (AKO (WATER-FLOW)) (PROPORTIONAL-TO (PRESSURE-PI)))
(RESISTANCE-PI (AKO (WATER-RESISTANCE)))
(PROPORTIONAL-TO-PI (MULTIPLIER (RESISTANCE-PI)))

(X-D1 (CAUSE (Y-D1)) (DETERMINED-BY (Y-D1 (SEE (DETERMINED-BY-D1)))))
(Y-D1 (DETERMINED-BY (X-D1)))
(Z-D1)
(DETERMINED-BY-D1 (PARAMETER (Z-D1)))
```

Still, it seems that something should be done to make the match happen. After all, PRESSURE-PI is known to be a kind of force, and since X-D1 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-PI and Z-D1 are both a kind of parameter.

At first, it might seem that awkward or complicated solutions are required. It might seem that the matcher would have to be smarter, perhaps made so by making it look into the AKO slots of the slots themselves.

Actually, there is an alternative that seems more natural and certainly is easy to effect. 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 PI and D1. 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 as follows:

```
Proportional-to has if-added (fpfsv frame 'increases-with
value).

Increases-with has if-added (fpfsv frame 'determined-by
value).

Multiplier has if-added (fpfsv frame 'parameter value).

Parameter has if-added (fpfsv value 'ako 'parameter).

Cause has if-added (fpfsv frame 'ako '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 AKO slots. They make simple deductions about what something is a kind of, given what it does.

- Demons make simple abstraction possible. Two situations can match well after abstraction even if they do not match well as they stand.

Now the same descriptions of PI and D1 produce frames that do produce a match:

```
(PRESSURE-PI (AKO (WATER-PRESSURE))
             (PROPORTIONAL-TO (FLOW-PI (SEE (PROPORTIONAL-TO-PI))))
             (INCREASES-WITH (FLOW-PI))
             (DETERMINED-BY (FLOW-PI)))
(FLOW-PI (AKO (WATER-FLOW))
         (PROPORTIONAL-TO (PRESSURE-PI))
         (INCREASES-WITH (PRESSURE-PI))
         (DETERMINED-BY (PRESSURE-PI)))
(RESISTANCE-PI (AKO (WATER-RESISTANCE) (PARAMETER)))
(PROPORTIONAL-TO-PI (MULTIPLIER (RESISTANCE-PI))
                    (PARAMETER (RESISTANCE-PI)))

(X-D1 (CAUSE (Y-D1))
      (AKO (FORCE))
      (DETERMINED-BY (Y-D1 (SEE (DETERMINED-BY-D1)))))
(Y-D1 (DETERMINED-BY (X-D1)))
(Z-D1 (AKO (PARAMETER)))
(DETERMINED-BY-D1 (PARAMETER (Z-D1)))
```

The actual result produced is as follows:

```
Matching D1 and PI
 ......

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

Note 5: Match can be Improved by Asking Questions

Suppose situation D2 is one in which nothing at all is known about the parts:

```
D2 is ako situation - has part x-d2 y-d2 and z-d2.
```

What can be done to match D2 against the pipe law, PI? 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 AKO 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 AKO paths starting with the constraint's parts. The figure illustrates. Then note where each AKO 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 AKO paths starting from it. Remember any places where these paths run into the paths that start from the constraint's parts. Call these places the top points. If none are found by following the AKO paths from the situation's parts, use the places where the AKO paths from the constraint's parts terminate. In the illustration, these top points are frames T1 and T2.

Finally, move down from these top points until the AKO 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.

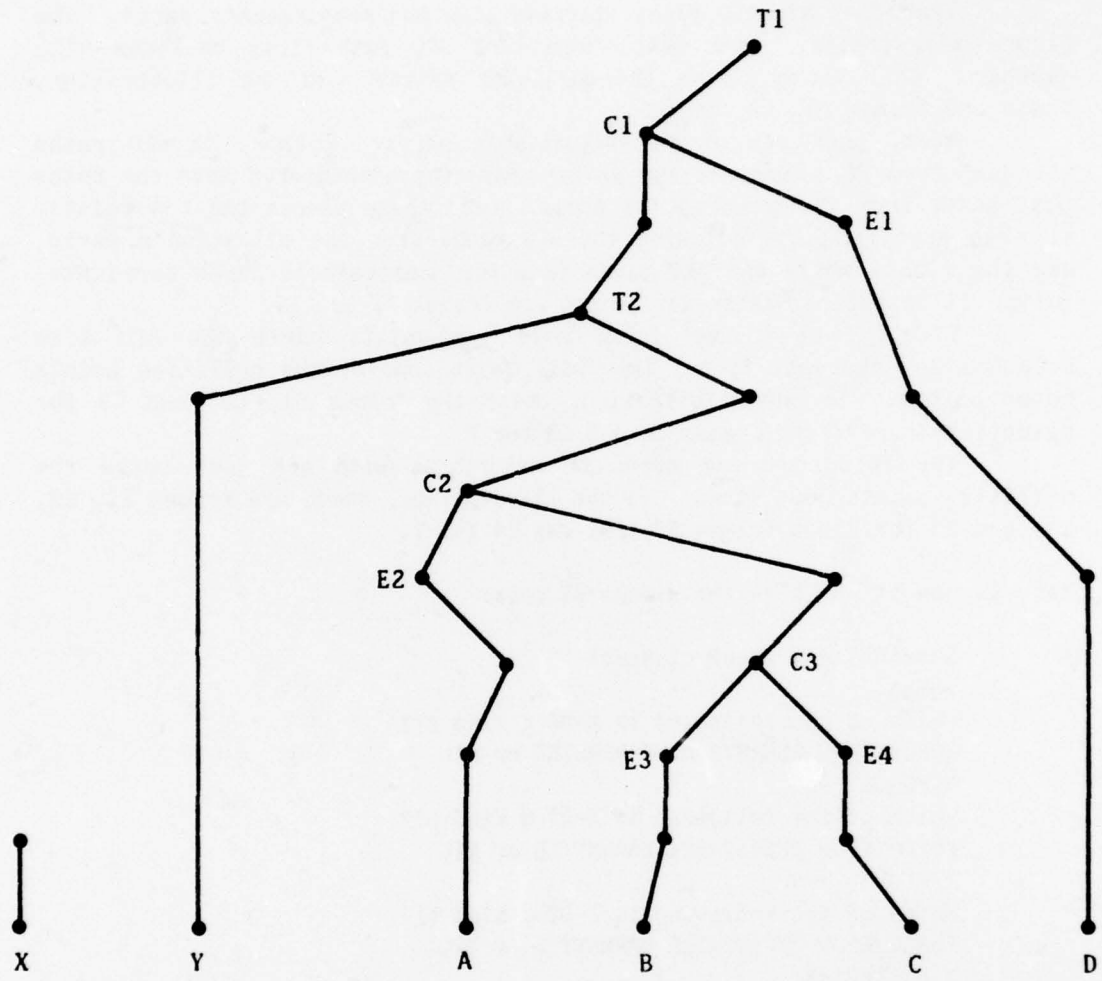
Here is how it works in the simple D2 case:

```
Should I ask about classes?
> YES
Which of the following is X-D2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
> FORCE
Which of the following is Y-D2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
> FLOW
Which of the following is Z-D2 a kind of?
FORCE FLOW RESISTANCE PARAMETER or NIL
> PARAMETER
You made improvements to X-D2 Y-D2 Z-D2
 ......

Score = 3. versus 3. and 16.
Match is decisive -- 2. better than next best.
((PRESSURE-PI X-D2) (FLOW-PI Y-D2) (RESISTANCE-PI Z-D2))
```

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 AKO-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 AKO relations. Instead, the smarter system could itself activate procedures or do experiments that answer the classification



Proposing AKO 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.

questions raised. If it is useful to know if X-D2 is a kind of FORCE, the sensible thing to do would be to look first for the sorts of relationships forces enter into. This makes the decision about what something is into an identification, matching, conclusion, and verification problem, thus recursing into the whole system.

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 the constraint but not in the situation. Then, ask if these slots hold anything in any of the parts of the situation.

Working again with D2, 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-D2 Y-D2 Z-D2
When signaled give a frame and value or nil.
What does PROPORTIONAL-TO hold between?
> (X-D2 Y-D2)
> NIL
What does INCREASES-WITH hold between?
> NIL
What does DETERMINED-BY hold between?
> NIL
What does AKO hold between?
> (Z-D2 PARAMETER)
New instances of these relations were placed:
AKO 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. And again, a smarter system would recurse into itself, not just ask.

Note 6: Both Specific and General Laws can be Discovered

This note shows how to learn the desciptive-frame part of new laws using pairs of analogies or pairs of known laws. To begin, suppose a second

electrical problem is solved using the same water-oriented analogy that was used for X2. Then there will be two similar situation frames, X2 and a new one, say Y2:

```
(X2 (AKO (SITUATION) (CONSTRAINT (SUGGESTED-BY (PIPE-LAW))))
    (PART (VOLTAGE-X2) (CURRENT-X2) (RESISTANCE-X2))
    (DEPENDENT-VARIABLE (VOLTAGE-X2 (SUGGESTED-BY (PIPE-LAW))))
    (INDEPENDENT-VARIABLE (CURRENT-X2 (SUGGESTED-BY (PIPE-LAW))))
    (MULTIPLIER (RESISTANCE-X2 (SUGGESTED-BY (PIPE-LAW)))))

(Y2 (AKO (SITUATION) (CONSTRAINT (SUGGESTED-BY (PIPE-LAW))))
    (PART (VOLTAGE-Y2) (CURRENT-Y2) (RESISTANCE-Y2))
    (DEPENDENT-VARIABLE (VOLTAGE-Y2 (SUGGESTED-BY (PIPE-LAW))))
    (INDEPENDENT-VARIABLE (CURRENT-Y2 (SUGGESTED-BY (PIPE-LAW))))
    (MULTIPLIER (RESISTANCE-Y2 (SUGGESTED-BY (PIPE-LAW)))))
```

Of course, these two frames and their parts are supersimilar and easy to match up. Once they are matched, it is possible to perform a merging operation that creates a constellation of new frames, one for each for the linked pairs produced by the match. Here is the result, given two things: first that X2 and Y2 are given a CONSTRAINED-BY slot with PIPE-LAW in it; and that both have parts described like those in S4:

```
I just made a new law named LAW1
The parts look like this:
(LAW1 (AKO (PIPE-LAW))
      (PART (FRAME1) (FRAME2) (FRAME3))
      (DEPENDENT-VARIABLE (FRAME1))
      (INDEPENDENT-VARIABLE (FRAME2))
      (MULTIPLIER (FRAME3)))
(FRAME1 (PART-OF (LAW1)) (AKO (VOLTAGE)))
(FRAME2 (PART-OF (LAW1)) (AKO (ELECTRIC-CURRENT)))
(FRAME3 (PART-OF (LAW1)) (AKO (ELECTRIC-RESISTANCE)))
```

Note that the descriptions of LAW1 and its parts are much like what Ohm's law would be like, except that LAW1 is a kind of PIPE-LAW, rather than CONSTRAINT. The appearance of PIPE-LAW insures access to RULE-PI, thus making the operational part of the new constraint functionally equivalent to Ohm's law. Thus the merge operation has invented Ohm's Law by using two examples in which electric problems were solved using water knowledge!

The merge involves these details:

- The matcher matches two groups, each of which consists of a situation frame and its parts. This produces a set of linked frames.

- All the slot-value combinations that are common to two linked

frames are copied into the new frame that represents them. If the two parts of a linked pair are found in the same slot of another linked pair, the relationship is preserved in the copy.

- The AKO slot is treated specially. The AKO slots of two linked frames start off trees of AKO-related frames. The AKO slot of the new frame that represents the linked frames is filled with those frames found where the two trees intersect.

- Some bookkeeping is done. SITUATION is removed from the AKO slot of the frame representing the new law. The values in the CONSTRAINED-BY slots of the situations are moved to the AKO slot of the new law. The CONSTRAINED-BY slot in the new law, if any, is then removed entirely.

Having done all this, electric problems can now be solved by using LAW1 more easily than by using PIPE-LAW. This is because the more electrically familiar things in LAW1 give the matcher easier going. Recall that the parts of X2 matched those in PIPE-LAW with a score of three. They match the parts in LAW1 with a score of six.

This development has shown how to make a new, specific law in one domain by repeated analogy to an old law in another domain. It is also possible to generalize two similar laws from two distinct domains. Suppose, for example, that a teacher suggests that there is something useful to be had by merging the newly acquired law, LAW1, with the PIPE-LAW from which it was derived:

```
(PIPE-LAW (AKO (CONSTRAINT))
          (PART (PRESSURE-PI) (FLOW-PI) (RESISTANCE-PI))
          (DEPENDENT-VARIABLE (PRESSURE-PI))
          (INDEPENDENT-VARIABLE (FLOW-PI))
          (MULTIPLIER (RESISTANCE-PI)))

(LAW1 (AKO (Y2) (X2))
      (PART (FRAME1) (FRAME2) (FRAME3))
      (DEPENDENT-VARIABLE (FRAME1))
      (INDEPENDENT-VARIABLE (FRAME2))
      (MULTIPLIER (FRAME3)))
```

After copying the old laws into the AKO slot of the new one and eliminating redundancy in that slot, the new law and its parts look like this:

```
I just made a new law named LAW2
The parts look like this:
(LAW2 (AKO (LAW1))
      (PART (FRAME4) (FRAME5) (FRAME6))
      (DEPENDENT-VARIABLE (FRAME4))
```

```
        (INDEPENDENT-VARIABLE (FRAME5))
        (MULTIPLIER (FRAME6)))
(FRAME4 (PART-OF (LAW2)) (AKO (FORCE)))
(FRAME5 (PART-OF (LAW2)) (AKO (FLOW)))
(FRAME6 (PART-OF (LAW2)) (AKO (RESISTANCE)))
```

This new law, LAW2, gives access to the same rule, RULE-PI, as before. The things involved, however, are more general than the things in PIPE-LAW or LAW1.

Note 7: The Situation-Action 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. Algorithms for doing this intersecting have been described just above.

- 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 HQ slots. For learning rules from stories it is assumed that the things that happen at the end of a story 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.

The following scenario illustrates these points and some other,

smaller details. The intent is to develop a rule suitable for attachment to the Othello story. It begins with a statement that is assumed given by the teacher or computed by intersecting situations named by a teacher:

```
SAMPLE is ako story - part Othello Desdamona - tragic-
figure Othello.

Othello married-to Desdamona - kill Desdamona Othello [see
curtain].

Curtain time end.
```

The results are as follows, as reported by the algorithm:

```
IF frames are:
(IF45 (MATCHES (SITUATION))
      (AKO (STORY))
      (TRAGIC-FIGURE (>OTHELLO)))
(IF46 (MATCHES (<OTHELLO))
      (MARRIED-TO (>DESDAMONA))
      (KILL (<DESDAMONA)))

THEN are:
(THEN32 (MODIFIES (<OTHELLO)) (KILL (<OTHELLO)))

Rule is:
(RULE24 (IF (IF45) (IF46)) (THEN (THEN32)))
```

Note 8: If-needed Demons are a Special Case of the Situation-Action Rules

Consider the optimistic demon that says a man loves a woman if he marries her and vice versa. The ordinary if-needed demon expressing this idea looks like this:

```
Marry has if-needed (fpfsv frame 'love value).
```

Let us see how this too can be learned as a special case of the general method. First the teacher supplies a situation and draws attention to it:

```
LOVE is ako fragment - part lo-1 lo-2 marry-lo - partl lo-1
- part2 lo-2.

Person instance lo-1 lo-2. Lo-1 marry lo-2 [see marry-lo].
Lo-1 love lo-2 [see love-lo]. Love-lo hq conclusion.
```

Applying the method for converting the situation into a rule leads to the

following result:

```
IF frames are:
(IF7 (MATCHES (SITUATION))
     (AKO (FRAGMENT))
     (PART1 (>LO-1))
     (PART2 (>LO-2)))
(IF8 (MATCHES (<LO-1)) (AKO (PERSON)) (MARRY (<LO-2)))
(IF9 (MATCHES (<LO-2)) (AKO (PERSON)))

THEN frames are:
(THEN8 (MODIFIES (<LO-1)) (LOVE (<LO-2)))

Rule is:
(RULE3 (IF (IF7) (IF8) (IF9)) (THEN (THEN8)))
```

Having learned this, the following situation illustrates its use:

```
SI is ako situation - part si-1 si-2 marry-si.

Person instance si-1 si-2.  Si-1 marry si-2 [see marry-si].
```

Now we are ready to work RULE3:

```
(PREDICT 'SI 'LOVE)
 ..

Score = 3. versus 3. and 5.
Match is decisive -- 1. better than next best.
The situation has been expanded by LOVE
The expanded story looks like this:
(SI (AKO (SITUATION) (FRAGMENT (SUGGESTED-BY (LOVE))))
    (PART (SI-1) (SI-2))
    (PART1 (SI-1 (SUGGESTED-BY (LOVE))))
    (PART2 (SI-2 (SUGGESTED-BY (LOVE)))))
Rule RULE3 produced new information.
We win -- prediction made.
(SI-1 (PART-OF (SI))
      (AKO (PERSON))
      (MARRY (SI-2))
      (LOVE (SI-2 (INSERTED-BY-RULE (RULE3)))))
```

So we have the ability to match and react to entire plots or tiny fragments or anything in between. The if-needed demon is a compiled special case.

APPENDIX 2

(COMMENT -*-text-*-

This is the general story file -- some details may differ from versions used to illustrate points in the body of the paper -- 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 AKO tree -- the first story is MA short for Macbeth --

END OF COMMENT)

S has part MA HA OT JU TA DH HG PY CI AD. SH has part MA HA OT JU TA.

(setq s (fgfsvs 's 'part)) (setq sh (fgfsvs 'sh 'part)) (setq fv-matchtypes '(person scene prop))

HA similar-to MA. OT similar-to JU HG CI DH. PY similar-to HG CI. TA similar-to CI.

JU similar-to OT MA. MA similar-to JU HA. HG similar-to PY OT. DH similar-to OT. CI similar-to TA PY OT.

Inverse has if-added (fpfsv value 'inverse frame) and (fpfsv frame 'if-added (list 'fpfsv 'value (list 'quote value) 'frame)).

AKO has inverse instance. Part has inverse part-of. Cause has inverse caused-by. Method has inverse purpose. After has inverse before.

Meet has inverse meet. Marry has inverse marry. Marry has if-added (FPFSV frame 'married-to value). Married-to has inverse married-to. Sibling has inverse sibling. Parent has inverse child. Friend-of has inverse friend-of.

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 ako 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.

(comment the following is for abstraction experiment only Kill has if-added (fpfsv frame 'hurt value) Hurt has if-added (fpfsv frame 'has-conflict-with value) comment end of temporary stuff)

(comment the following is for cause-driven match experiment only) Cause has if-added (fpfsv frame 'hq 'important) (fpfsv value 'hq 'important) (fpfsv (fgfsvcr frame slot value 'see) 'hq 'important). Prevent has if-added (fpfsv frame 'hq 'important) (fpfsv value 'hq 'important) (fpfsv (fgfsvcr frame slot value 'see) 'hq 'important). Motive has if-added (fpfsv frame 'hq 'important) (fpfsv value 'hq 'important) (fpfsv (fgfsvcr frame slot

value 'see) 'hq 'important). Reason has if-added (fpfsv frame 'hq 'important) (fpfsv value 'hq 'important) (fpfsv (fgfsvcr frame slot value 'see) 'hq 'important). Desire has if-added (fpfsv frame 'hq 'important) (fpfsv value 'hq 'important) (fpfsv (fgfsvcr frame slot value 'see) 'hq 'important).

Murder has if-added (fpfsvcr frame 'hq 'evil 'because '(person murders)) (fpfsv (fgfsvcr frame 'murder value 'see) 'cause (fgfsvcr frame 'hq 'evil 'see)).

Murder has if-added (fpfsv frame 'kill value) (fpfsv (fgfsvcr frame 'murder value 'see) 'cause (fgfsvcr frame 'kill value 'see)).

Kill has if-added (fpfsv value 'hq 'dead) (fpfsv (fgfsvcr frame 'kill value 'see) 'cause (fgfsvcr value 'hq 'dead 'see)). (comment end of temporary stuff)

Stronger-than has inverse weaker-than. Murder has if-added (fpfsvcr frame 'kill value 'because '(murder asserted)). Murder has if-added (fpfsvcr frame 'hq 'evil 'because '(person murders)). Kill has if-added (fpfsvcr value 'hq 'dead 'because (list frame 'kill value)).

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

Ako has if-added (constrain).

MA is ako story - part Macbeth Lady-macbeth Duncan Macduff Wierd-sisters - subpart heath-scene murder-scene-ma battle-ma.

MA has tragic-figure Macbeth - rule rule-ma.

Macbeth is ako noble [see ako-ma-1] king [see ako-ma-2] - hq happy [see hq-ma-1] - married-to Lady-macbeth. Ako-ma-2 after ako-ma-1.

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

Heath-scene is ako scene - before murder-scene-ma. Murder-scene-ma is ako scene - before battle-ma. Battle-ma is ako scene.

Wierd-sisters speak-to Macbeth [see speak-to-ma] - predict ako-ma-2 [see predict-ma] kill-ma [see predict-ma]. Speak-to-ma content predict-ma - time heath-scene.

Lady-macbeth persuade Macbeth [see persuade-ma] - cause murder-ma [see cause-ma]. Persuade-ma act murder-ma. Cause-ma method persuade-ma.

Macbeth murder Duncan [see murder-ma] - desire ako-ma-2. Murder-ma has coagent Lady-macbeth - motive ako-ma-2 - instrument knife - time murder-scene-ma.

Macbeth hq unhappy [see hq-ma-2]. Hq-ma-2 after hq-ma-1. Lady-macbeth hq dead.

Macduff kill Macbeth [see kill-ma]. Kill-ma reason murder-ma - time battle-ma.

RULE Rule-ma IF Situation has tragic-figure >Macbeth. THEN <Macbeth hq unhappy and dead. END

HA is ako story - part Hamlet Ghost Claudius Gertrude Laertes.

Hamlet is ako prince - hq unhappy [see hq-ha-1] - parent Ghost Gertrude. Hq-ha-1 caused-by hq-ha-2.

Ghost is ako king - hq dead [see hq-ha-2] - sibling Claudius - married-to Gertrude [see married-to-ha-1]. Married-to-ha-1 before hq-ha-2.

Claudius is ako king [see ako-ha] - married-to Gertrude [see married-to-ha-2]. Married-to-ha-2 after married-to-ha-1.

Gertrude is ako queen. Laertes is ako man.

Claudius murder Ghost [see murder-ha]. Murder-ha motive ako-ha - cause order-ha.

Ghost order Hamlet [see order-ha]. Order-ha act kill-Claudius.

Claudius persuade Laertes [see persuade-ha] - cause kill-Hamlet [see cause-ha]. Persuade-ha act kill-Hamlet. Cause-ha method Persuade-ha.

Gertrude hq unhappy - kill Gertrude [see kill-Gertrude]. Kill-Gertrude instrument poisen.

Laertes kill Hamlet [see kill-Hamlet]. Kill-Hamlet instrument sword.

Hamlet kill Laertes [see kill-Laertes] Claudius [see kill-Claudius]. Kill-Laertes caused-by kill-Hamlet - instrument sword. Kill-Claudius caused-by Ghost - instrument sword.

OT is ako story - part Othello Desdamona Iago Cassio.

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

Othello love Desdamona - marry Desdamona. Desdamona love Othello.

Iago speak-to Othello [see speak-to-ot]. Speak-to-ot ako lie - content love-ot. Iago persuade othello [see persuade-ot] - cause jealous-of-ot

[see cause-ot]. Persuade-ot method speak-to-ot - relation jealous-of-ot. Cause-ot method persuade-ot - motive hate-ot help-ot. Desdamona love Iago [see love-ot]. Love-ot hq false.

Othello jealous-of Cassio [see jealous-of-ot] - help Iago [see help-ot] - hate Cassio [see hate-ot].

Jealous-of-ot cause hate-ot kill-ot.

Othello cause hq-ot. Cassio hq dead [see hq-ot].

Othello kill Desdamona [see kill-ot] and Othello [see kill-ot].

JU is ako story - part Caesar Brutus Antony Cassius - subpart Ides-of-march Funeral Battle-at-philippi.

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

Ides-of-march is ako scene - location senate - before funeral. Funeral is ako scene - before battle-at-philippi. Battle-at-philippi is ako scene.

Cassius hate Caesar - persuade Brutus [see persuade-ju] - cause murder-Caesar [see cause-ju]. Persuade-ju act murder-Caesar. Cause-ju method persuade-ju.

Brutus love Rome - kill Caesar [see murder-Caesar]. Cassius murder Caesar [see murder-Caesar]. Murder-Caesar instrument knife - time ides-of-march.

Antony love Caesar - speak-to people [see speak-to-ju] - attack Brutus [see attack-ju] Cassius [see attack-ju]. Speak-to-ju cause attack-ju time funeral. Attack-ju time battle-at-philippi.

Brutus attack Antony [see attack-ju]. Cassius attack Antony [see attack-ju].

Brutus kill Brutus [see kill-ju-2].

Cassius kill Cassius [see kill-ju-2].

Kill-ju-2 time battle-at-philippi.

TA is ako story - part Petruchio Katharina Bianca Lucentio.

Petruchio is ako man - hq strong smart. Katharina is ako shrew - hq rich masochist [see hq-ta]. Lucentio is ako student man. Bianca is ako woman - sibling Katharina.

Petruchio marry Katharina - forbid Katherina [see forbid-ta] - prevent eat-ta [see prevent-ta] sleep-ta [see prevent-ta]. Prevent-ta method forbid-ta - motive love-ta. Forbid-ta act eat-ta sleep-ta.

Katharina eat * [see eat-ta] - sleep * [see sleep-ta] - love Petruchio [see love-ta]. Love-ta caused-by hq-ta prevent-ta.

Petruchio control Katharina [see control-ta-1]. Control-ta-1 caused-by love-ta - hq strong.

Bianca love Lucentio - marry Lucentio. Lucentio love Bianca - control Bianca [see control-ta-2]. Control-ta-2 hq weak.

Control-ta-1 stronger-than control-ta-2.

DH is ako story - part Nora Torvald Krogstad Christina - subpart letter-dh.

Nora is ako woman - hq sensitive proud - friend-of Christina - married-to Torvald.

Torvald is ako man - hq crude weak. Krogstad is ako man - hq cruel. Christina is ako woman - hq good.

Letter-dh is ako letter prop.

Torvald love Nora - help Christina [see help-dh] - hate Krogstad [see hate-dh] - hurt Krogstad [see hurt-dh]. Help-dh motive hurt-dh.

Krogstad prevent help-dh [see prevent-dh-1] hurt-dh [see prevent-dh-1] - threaten Nora [see threaten-dh] - give letter-dh [see give-dh]. Prevent-dh-1 method threaten-dh. Threaten-dh act give-dh. Give-dh destination Torvald.

Christina prevent give-dh [see prevent-dh-2] - marry Krogstad [see marry-dh]. Prevent-dh-2 method marry-dh - hq failed.

Torvald attack Nora [see attack-dh]. Attack-dh caused-by give-dh letter-dh.

Nora leave Torvald [see leave-dh]. Leave-dh caused-by attack-dh.

HG is ako story - part Hedda George Lovberg Elvsted - subpart pistol-hg manuscript-hg.

Hedda is ako woman - hq proud beautiful. George is ako man - hq dull. Lovberg is ako man - hq smart. Elvsted is ako woman - hq dull ugly.

Pistol-hg is ako pistol prop.

Manuscript-hg is ako manuscript prop.

Hedda marry George - hate George.

George want ako-hg - ako professor [see ako-hg] - jealous-of Lovberg.

Lovberg love Hedda - write manuscript-hg [see write-hg] - lose manuscript-hg.

Elvsted help Lovberg [see help-hg] - control Lovberg [see control-hg-1]. Help-hg act write-hg.

Hedda hate Elvsted [see hate-hg]. Hate-hg caused-by control-hg-1 want-hg.

Hedda want control-hg-2 [see want-hg] - control Lovberg [see control-hg-2].

Hedda persuade Lovberg [see persuade-hg] - cause destroy-hg [see cause-hg]. Cause-hg motive control-hg-2 - method persuade-hg.

Lovberg destroy manuscript-hg [see destroy-hg] - kill Lovberg [see kill-hg-2]. Hedda cause kill-hg-2. Kill-hg-2 hq failed - instrument pistol-hg.

Hedda kill Hedda [see kill-hg-1]. Kill-hg-1 instrument pistol-hg.

PY is ako story - part Eliza Higgens Pickering and Freddy.

Eliza is ako girl beautiful - hq proud stubborn sensitive. Higgens is ako professor man - hq smart crude [see hq-py-1] - friend-of Pickering. Pickering is ako gentleman colonel - hq wise old proper - friend-of Higgens. Freddy is ako man - hq poor young foolish.

Higgens teach Eliza [see teach-py] - help Eliza [see help-py]. Teach-py content English. Help-py method teach-py. Higgens hurt Eliza [see hurt-py] - love Eliza [see love-py-1]. Hurt-py caused-by hq-py-1. Love-py-1 hq strange.

Pickering help Eliza.

Eliza is ako lady [see ako-py]. Higgens cause ako-py [see cause-py]. Cause-py method teach-py.

Eliza love Higgens [see love-py-2]. Love-py-2 caused-by help-py.

Eliza leave Higgins [see leave-py] - marry Freddy [see marry-py] - hq unhappy [see hq-py-2]. Higgens hq unhappy [see hq-py-2].

Hurt-py cause leave-py marry-py. Marry-py cause hq-py-2.

CI is ako story - part Cindy Charming Godmother Stepmother - subpart godmother-meeting ball foot-search glass-slipper - cinderella cindy - charming charming - rule rule-ci.

Cindy is ako servant girl - hq beautiful sweet unhappy. Charming is ako prince - hq beautiful. Godmother is ako fairy and woman - hq wise old and kind. Stepmother is ako bitch - hq cruel.

Godmother-meeting is ako scene - before ball. Ball is ako ammusement scene - before foot-search. Foot-search is ako scene.

Glass-slipper is ako prop and shoe.

Step-mother hates Cindy.

Godmother help Cindy [see help-ci] - give dress [see give-ci] coach [see give-ci]. Help-ci act give-ci - motive attend-ci - time godmother-meeting.

Cindy attend ball [see attend-ci]. Attend-ci time ball. Cindy meet Charming [see meet-ci] - love Charming. Meet-ci time ball.

Charming love Cindy - lose Cindy.

Charming find Cindy [see find-ci]. Find-ci instrument glass-slipper - time foot-search.

Cindy marry Charming - hq happy.

RULE Rule-ci IF Situation has cinderella >c - charming >p. THEN <c marry <p - hq happy. END

AD is ako story - part God Devil Adam Eve - subpart apple-feast eviction evil-apple.

God is ako spirit - hq good strong. Spirit is ako person. Devil is ako spirit - hq bad strong cruel. Adam is ako man - hq happy. Eve is ako woman - hq happy foolish.

Apple-feast is ako scene - before eviction. Eviction is ako scene.

Evil-apple is ako apple prop. Apple is ako fruit. God forbid Adam [see forbid-ad]. Forbid-ad act eat-ad - hq failed [see hq-ad].

Devil persuade Eve [see persuade-ad-1]. Persuade-ad-1 act persuade-ad-2. Eve persuade Adam [see persuade-ad-2]. Persuade-ad-2 act eat-ad.

Adam eat evil-apple [see eat-ad].

God forbid eat-ad [see forbid-ad] - move Adam [see move-ad] Eve [see move-ad].

Adam eat evil-apple [see eat-ad]. Eat-ad time apple-feast.

God punish Adam [see punish-ad] Eve [see punish-ad]. Punish-ad reason hq-ad eat-ad - act move-ad.

God move Adam [see move-ad] Eve [see move-ad]. Move-ad source Paradise - destination Outer-darkness - time eviction.

(frfsv 'ako 'if-added '(constrain))

APPENDIX 3

```
(COMMENT -*-text-*-

This is the mechanical-electrical problem file --

END OF COMMENT)

(setq base (setq ibase 10))

Inverse has if-added (fpfsv value 'inverse frame) (fpfsv frame 'if-added
        (list 'fpfsv 'value (list 'quote value) 'frame)) (fpfsv value 'if-
added
        (list 'fpfsv 'value (list 'quote frame) 'frame)).

AKO has inverse instance. Part has inverse part-of. Cause has inverse
caused-by. Proportional-to has inverse proportional-to.

Related-to has inverse related-to. Derivitive has inverse integral.
Mathematical-inverse has inverse mathematical-inverse.

AKO has if-added (mapc '(lambda (e) (fpfsv frame 'related-to e))
      (delete frame (fgfsvs value 'instance))).

(setq e 'electrical-situation-group)

Electrical-situation-group has part resistor-situation capacitor-situation
inductor-situation.

Resistor-situation has part resistance conductance resistor-voltage
resistor-current. Resistor-voltage proportional-to resistor-current [see
proportional-to-r-1]. Resistor-current proportional-to resistor-voltage
[see proportional-to-r-2]. Proportional-to-r-1 constant resistance.
Proportional-to-r-2 constant conductance. Resistance mathematical-inverse
conductance.

Capacitor-situation has part capacitance capacitor-voltage capacitor-
current. Capacitor-current proportional-to-derivitive-of capacitor-voltage
[see proportional-to-c]. Proportional-to-c constant capacitance.

Inductor-situation has part inductance inductor-voltage inductor-current.
Inductor-voltage proportional-to-derivitive-of inductor-current [see
proportional-to-l]. Proportional-to-l constant inductance.

(setq m 'mechanical-situation-group)

Mechanical-situation-group has part damper-situation spring-situation
moving-mass-situation.

Damper-situation has part b 1-over-b force-on-damper damper-velocity.
Force-on-damper proportional-to damper-velocity [see proportional-to-d-1].
Damper-velocity proportional-to force-on-damper [see proportional-to-d-2].
Proportional-to-d-1 constant b. Proportional-to-d-2 constant 1-over-b. B
mathematical-inverse 1-over-b.
```

Spring-situation has part k 1-over-k force-on-spring spring-position spring-velocity. Force-on-spring proportional-to spring-position [see proportional-to-s1]. Spring-position proportional-to force-on-spring [see proportional-to-s2]. Spring-velocity proportional-to-derivitive-of force-on-spring [see proportional-to-s3]. Proportional-to-s1 constant k. Proportional-to-s2 constant 1-over-k. Proportional-to-s3 constant 1-over-k. K mathematical-inverse 1-over-k.

Moving-mass-situation has part mass force-on-mass mass-velocity. Force-on-mass proportional-to-derivitive-of mass-velocity [see proportional-to-m]. Proportional-to-m constant m.

Voltage has instance resistor-voltage capacitor-voltage inductor-voltage. Current has instance resistor-current capacitor-current inductor-current. Constant has instance resistance conductance capacitance inductance.

Force has instance force-on-damper force-on-spring force-on-mass. Velocity has instance damper-velocity spring-velocity mass-velocity. Constant has instance b 1-over-b k 1-over-k m.

APPENDIX 4

These are the results of hypothesizing the identity of all the stories in the story file. the story file itself constitutes the set of possiblities.

In MA the following parts are prominent: 6. MACBETH 6. LADY-MACBETH 6. WIERD-SISTERS 4. MACDUFF 3. DUNCAN

The following stories come to mind: 35.9 MA 13.9 HA 8.1 JU 5.1 OT 5.1 TA 5.1 DH 5.1 HG 5.1 PY 5.1 CI 5.1 AD

In HA the following parts are prominent: 9. CLAUDIUS 8. GHOST 6. GERTRUDE 5. HAMLET 4. LAERTES

The following stories come to mind: 31.1 HA 22.6 MA 13.4 JU 8.9 CI 6.4 OT 6.4 TA 6.4 DH 6.4 HG 6.4 PY 6.4 AD

In OT the following parts are prominent: 11. OTHELLO 6. DESDAMONA 6. IAGO 3. CASSIO

The following stories come to mind: 25.4 OT 13.7 JU 8.9 PY 5.2 MA 5.2 HA 5.2 TA 5.2 DH 5.2 HG 5.2 CI 5.2 AD

In JU the following parts are prominent: 9. CASSIUS 6. ANTONY 5. BRUTUS 3. CAESAR

The following stories come to mind: 11.1 JU 7.1 OT 5.6 PY 5.6 MA 5.6 HA 4.6 TA 4.6 DH 4.6 HG 4.6 CI 4.6 AD

In TA the following parts are prominent: 9. KATHARINA 7. PETRUCHIO 6. BIANCA 6. LUCENTIO

The following stories come to mind: 25.2 TA 10.2 CI 5.7 MA 5.7 HA 5.7 OT 5.7 DH 5.7 HG 5.7 PY 5.7 AD 4.0 JU

In DH the following parts are prominent: 9. TORVALD 6. NORA 6. KROGSTAD 5. CHRISTINA

The following stories come to mind: 5.3 MA 5.3 HA 5.3 OT 5.3 TA 5.3 DH 5.3 HG 5.3 PY 5.3 CI 5.3 AD 4.1 JU

In HG the following parts are prominent: 11. HEDDA 8. LOVBERG 7. GEORGE 5. ELVSTED

The following stories come to mind: 9.8 HG 9.8 PY 6.3 MA 6.3 HA 6.3 OT 6.3 TA 6.3 DH 6.3 CI 6.3 AD 4.5 JU

In PY the following parts are prominent: 9. HIGGENS 7. ELIZA 5. PICKERING 5. FREDDY

The following stories come to mind: 38.9 PY 9.7 HG 8.7 CI 6.9 OT 6.1 JU 5.2 MA 5.2 HA 5.2 TA 5.2 DH 5.2 AD

In CI the following parts are prominent: 9. CHARMING 8. CINDY 5. GODMOTHER

3. STEPMOTHER

The following stories come to mind: 28.1 CI 9.6 HA 9.1 PY 6.6 TA 5.1 MA 5.1 OT 5.1 DH 5.1 HG 5.1 AD 3.4 JU

In AD the following parts are prominent: 6. GOD 4. DEVIL 4. ADAM 4. EVE

The following stories come to mind: 12.6 AD 2.6 MA 2.6 HA 2.6 OT 2.6 TA 2.6 DH 2.6 HG 2.6 PY 2.6 CI 2.2 JU

Appendix 5

```
(COMMENT -*-text-*-

This is the version of Macbeth used to check a conclusion in a matching
situation -- the free variable FV-CAREFUL turns on careful mode in which
all slot-value combinations get comments that point back to the frame slot
and value

END OF COMMENT)

(setq fv-matchtypes '(man person scene prop))

Inverse has if-added (fpfsv value 'inverse frame) and (fpfsv frame 'if-
added (list 'fpfsv 'value (list 'quote value) 'frame)).

AKO has inverse instance. Part has inverse part-of. Cause has inverse
caused-by. Desire has inverse desired-by. Method has inverse purpose.
After has inverse before.

Meet has inverse meet. Marry has inverse marry. Marry has if-added (FPFSV
frame 'married-to value). Married-to has inverse married-to. Sibling has
inverse sibling. Parent has inverse child. Friend-of has inverse friend-
of.

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 ako 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.

Stronger-than has inverse weaker-than.

Murder has if-added (fpfsvcr frame 'hq 'evil 'because '(person murders)).

Murder has if-added (fpfsv frame 'kill value)

(fpfsv (fgfsvcr frame 'murder value 'see)
       'cause
       (fgfsvcr frame 'kill value 'see)).

Kill has if-added (fpfsv value 'hq 'dead)

(fpfsv (fgfsvcr frame 'kill value 'see)
       'cause
       (fgfsvcr value 'hq 'dead 'see)).

Hq has if-added (cond ((memq value '(cruel))
                       (fpfsvcr frame 'hq 'evil 'because
                                (list frame 'hq value)))
                      ((memq value '(loyal honest))
```

```
(fpfsvcr frame 'hq 'good 'because
        (list frame 'hq value))).
```

```
(setq fv-careful t)
```

MA is ako story - part Macbeth Lady-macbeth Duncan Macduff - subpart heath-scene murder-scene-ma battle-ma.

Macbeth is ako noble [see ako-ma-1] king [see ako-ma-2] - hq happy [see hq-ma-1] - married-to Lady-macbeth. Ako-ma-2 after ako-ma-1.

Lady-macbeth is ako woman - hq greedy ambitious. Duncan is ako king. Macduff is ako noble - hq loyal angry.

Heath-scene is ako scene - before murder-scene-ma. Murder-scene-ma is ako scene - before battle-ma. Battle-ma is ako scene.

Lady-macbeth pursuade Macbeth [see pursuade-ma] - cause murder-ma [see cause-ma]. Pursuade-ma act murder-ma. Cause-ma method pursuade-ma.

Macbeth murder Duncan [see murder-ma] - desire ako-ma-2 [see desire-ma]. Murder-ma has coagent Lady-macbeth - motive ako-ma-2 - instrument knife - time murder-scene-ma.

Macbeth hq unhappy [see hq-ma-2]. Hq-ma-2 after hq-ma-1. Lady-macbeth hq dead.

Macduff kill Macbeth [see kill-ma]. Kill-ma reason murder-ma - time battle-ma.

XB is ako story - part Macbeth-xb Duncan-xb Lady-Macbeth-xb Macduff-xb.

Man has instance Macbeth-xb Duncan-xb Macduff-xb. Woman has instance Lady-Macbeth-xb.

Macbeth-xb married-to Lady-Macbeth-xb. Lady-Macbeth-xb hq greedy. Macduff-xb hq loyal.

APPENDIX 6

```
(COMMENT -*-text-*-

This is Meldman's law file --

END OF COMMENT)

(setq fv-matchtypes '(person weapon legal-term))

Inverse has if-added (fpfsv value 'inverse frame) and (fpfsv frame 'if-
added (list 'fpfsv 'value (list 'quote value) 'frame)).

AKO has inverse instance. Part has inverse part-of. Cause has inverse
caused-by. Prevent has inverse prevented-by. Method has inverse purpose.
After has inverse before.

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.

Firearm is ako weapon. Legal-term has instance contact apprehend intend.

Hq has if-added (cond ((memq value '(cruel))
                       (fpfsvcr frame 'hq 'evil 'because
                                (list frame 'hq value)))
                      ((memq value '(loyal honest))
                       (fpfsvcr frame 'hq 'good 'because
                                (list frame 'hq value)))).

Punch has if-added (fcause 'hit). Hit has if-added (fcause 'contact).
Kick has if-added (fcause 'contact).

Want has if-added (fcause 'intend).

Slot has instance apprehend contact intend.

Ako has if-added (constrain).

(setq fv-careful t)

(setq as 'assault1)

Assault1 is ako doctrine assault [see ako-as] - has part A1 A2 apprehend-as
intend-as.

Person instance a1 a2. Apprehend-as is ako apprehend.

A1 contact A2 [see contact-as]. A2 apprehend contact-as [see apprehend-
as]. A1 intend apprehend-as [see intend-as]. Ako-as caused-by apprehend-
as intend-as.

(setq ba 'battery1)
```

Battery1 is ako doctrine battery [see ako-ba] - has part B1 B2 contact-ba intend-ba.

Person instance b1 b2. Contact-ba is ako contact.

B1 contact b2 [see contact-ba]. B1 intend contact-ba [see intend-ba]. Ako-ba caused-by contact-ba intend-ba.

(setq SW 'Adams-vs-Zent)

Adams-vs-Zent is ako case - part Adams Zent contact-sw glasses-sw - defendant Adams - plaintif Zent - rule rule-sw.

Rule rule-sw if situation has defendant >d - plaintif >p. then <d contact <p. end

Person instance Adams Zent. Contact-sw is ako contact.

Adams contact Zent [see contact-sw] - knock-off glasses-sw [see knock-off-sw] - intend contact-sw [see intend-sw].

Knock-off-sw cause contact-sw. Adams intend knock-off-sw [see intend-sw-2]. Intend-sw-2 cause intend-sw. Zent wear glasses-sw. Glasses-sw is ako clothing [see ako-sw]. Ako-sw cause contact-sw.

(setq AZ 'Smith-vs-Wesson)

Smith-vs-Wesson has part Smith Wesson contact-az apprehend-az rifle - defendant Smith - plaintif Wesson - act contact-az - rule rule-az.

Rule rule-az if situation has defendant >d - plaintif >p - act >a. then <p apprehend <a. end

Person has instance Smith Wesson. Apprehend-az is ako apprehend. Intend has instance intend-az-1 intend-az-2.

Smith contact Wesson [see contact-az].

Smith point rifle [see point-az]. Point-az target Wesson [see target-az].

Smith intend apprehend-az [see intend-az-2]. Intend-az-2 caused-by intend-az-1. Smith intend frighten-az [see intend-az-1].

Smith frighten Wesson [see frighten-az]. Frighten-az caused-by ako-az point-az target-az.

Rifle ako firearm [see ako-az] - hq unloaded [see hq-az-1] harmless [see hq-az]. Hq-az-1 cause hq-az.

Wesson apprehend contact-az [see apprehend-az]. Wesson know hq-az [see know-az]. Know-az prevent frighten-az. Wesson not-know hq-az [see not-know-az]. Not-know-az prevent know-az - cause frighten-az [see cause-az]. Cause-az hq enablement. Apprehend-az caused-by frighten-az.

```
(frfsv 'ako 'if-added '(constrain))

(setq VV 'Villain-vs-Victim)

Villain-vs-Victim is ako case - part Villain Victim hat-vv pistol contact-
vv apprehend-vv.

Contact-vv is ako contact.  Apprehend-vv is ako apprehend.

Person instance Villain Victim.

Villain contact Victim [see contact-vv].  Victim apprehend contact-vv [see
apprehend-vv].

Villain knock-off hat-vv [see knock-off-vv].  Victim wear hat-vv.  Hat-vv
is ako hat.  Hat is ako clothing.

Villain point pistol [see point-vv].  Pistol ako firearm toy - hq harmless.
Point-vv target Victim - motive frighten-vv.  Villain frighten Victim [see
frighten-vv].

(setq GI 'Guilty-vs-Innocent)

Guilty-vs-Innocent is ako case - has part Guilty Innocent.

Person instance Guilty Innocent.

Guilty kick Innocent [see kick-gi] - want kick-gi.
```