INDUCE 4: A Program for Incrementally Learning Structural Descriptions from Examples

by

John A. Bentrup, Gary J. Mehler, Joel D. Riedesel

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

File No. UIUCDCS-F-87-958
ISG Report 87-2

February, 1987

This research was supported in part by the Office of Naval Research under Grant No. 00014-82-K0186, the National Science Foundation under Grant No. NSF DCR 84-06801, the Defense Advanced Research Project Agency under Grant No. N00014-85-K0878, and grant 1850058 from the Fund for the Improvement of Post-Secondary Education, U.S. Department of Education.

TABLE OF CONTENTS

ABSTRACT

The program *INDUCE 4* is a general-purpose *incremental inductive learning* program that transforms symbolic descriptions of real-world events into more general and more useful descriptions of these events. These events may be specified in terms of attribute as well as *structural* descriptors. The program produces such descriptions by performing various generalizing, simplifying and *constructive* transformations on the input descriptions, under the guidance of background knowledge specified by the user. As new events are made available, *INDUCE 4* can incrementally modify what has been proviously generated to reflect this new knowledge.

Key words:
Machine learning, Concept learning, Inductive inference, Learning from examples, Incremental learning, Structural descriptors, Constructive induction.

ACKNOWLEDGEMENTS

We would like to thank Ryszard S. Michalski for stimulating our interest in Machine Learning through this project. It has been very useful and interesting to have the facilities to implement and research new areas in Machine Learning. It has also been very worthwhile to have had the opportunity to contribute new research as a result of a class project. In spite of the fact the program we worked with was modified once too many times, we were still able to come away from it with a sense of accomplishment. We would also like to thank Debra Place for her help in providing necessary materials. Thanks are also extended to Carl Kadie for his robot examples.

Furthermore, we would like to thank Prof. R.S. Michalski, Carl Uhrik, Carl Kadie, and Mark Klein for providing valuable comments and criticisms.

This research was supported in part by the Office of Naval Research under Grant No. 00014-82-K-0186, the National Science Foundation under Grant No. NSF DCR 84-06801, the Defense Advanced Research Project Agency under Grant No. N00014-85-K-0878, and grant 1850058 from the Fund for the Improvement of Post-Secondary Education, U.S. Department of Education.

1. INTRODUCTION

INDUCE 4 is a general purpose learning program that learns structural descriptions from examples [Michalski 1980, 1983]. It may be run in both an incremental and a batch mode. The incremental mode of learning is incremental in the sense that *learning packets* may be processed by the program as they arrive. It is in this way that the program updates its descriptions of classes of examples. This is in contrast to constructive induction (also called incremental learning) in which intermediate hypotheses are generated in the process of learning a concept. A learning packet is any set of examples from one or more classes about which we desire to learn.

This paper is a description of the program *INDUCE 4*. The predecessors of this program include *INDUCE 3* [Michalski & Stepp, in preparation], *INDUCE 2* [Hoff, Michalski & Stepp, 1983], and *INDUCE 1* (described by the PhD thesis of James B. Larson [Larson 1977]). Enhancements were made to *INDUCE 1* by Thomas Dietterich in 1978 [Dietterich & Michalski 1978], and by Mihran Tuceryan in 1980. The major difference between *INDUCE 3* and *INDUCE 4* is the addition of a facility for performing incremental learning. Other changes include performance enhancements and input parsing modifications.

1.1 TERMINOLOGY

This information is presented to familiarize the reader with key terms and ideas. A more thorough treatment of this material may be found in [Larson 1977, Michalski 1983].

Objects and concepts are described by the language VL_2, which is an extension of first-order predicate logic [Michalski 1983]. Each object or concept is considered to be an *event*, which is a conjunction of selectors. A *selector* is a relational statement that contains a predicate *descriptor* (with variables as arguments) and a list of values that the descriptor may assume, or only a predicate descriptor if it is a boolean predicate. For example, Fig. 1 can be described in VL_2 as:

```
[contains(figure1,p1)][contains(figure1,p2)][contains(figure1,p3)]
[contains(figure1,p4)][size(p1)=large][size(p2)=medium][size(p3)=medium]
[size(p4)=medium][shape(p1)=rectangle][shape(p2)=circle][shape(p3)=circle]
[shape(p4)=triangle][color(p1)=clear][color(p2)=shaded][color(p3)=shaded]
[color(p4)=striped][ontop(p1,p2)][ontop(p1,p3)][ontop(p4,p1)].
```

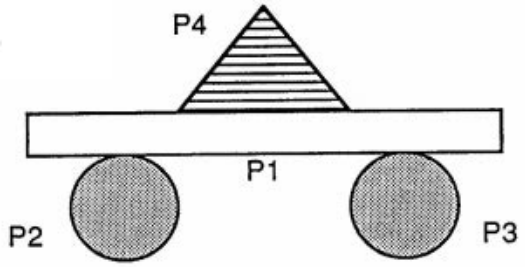


Fig. 1.

While in english Fig. 1 could be described (translation from VL_2 description):

Fig. 1 contains 4 parts of which one is a large rectangle and is on top of two medium sized circles. There is a medium triangle on the rectangle. The circles are shaded, the triangle is striped and the rectangle is clear.

Descriptors may be divided into two classes, null- or one-argument predicates (attributes), and two or more argument (non-unary) predicates. Null- or one-argument descriptors typically represent attributes such as `size` or `shape`, and are called *attribute descriptors*. Non-unary descriptors are used to represent structural or relational information between objects such as `ontop` or `inside`, and are called *relational descriptors*.

A *complex* is a conjunction of selectors. A complex is satisfied by an event if every selector of the complex is satisfied by the event. The event description on the preceding page is an example of a complex. Each event is assigned to a class by the user. If one wishes to assign a complex to a class, this would be specified as: `[complex] => [class=class_name].`

A *concept description* is the disjunction of a set of complexes. A concept description *covers* an event if at least one complex of the set is satisfied. Concept descriptions, also known as *hypotheses*, are generated to describe a class of events. A set of candidate concept descriptions are generated from a *star*. A star of an event e against a set of events Γ is a maximally generalized set of complexes which is satisfied by the event e but are not satisfied by any member of the set of events Γ.

A concept description is *consistent* if it does not cover an event from any other concept. A concept description is *complete* if it covers all of the events from the concept being learned.

A *characteristic* concept description is an expression that satisfies the completeness condition, while a *discriminant* concept description is an expression that satisfies both the consistency and completeness conditions [Michalski 1983, Reinke & Michalski 1985]. Ideally, the best characteristic description is *maximal* while the best discriminant description is *minimal* [Reinke & Michalski 1985]. Here maximal means the largest possible description that characterizes the class, while minimal means the smallest possible description that discriminates between classes.

1.2. INCREMENTAL LEARNING

Incremental learning is useful in many learning situations. It is sometimes useful to generate a hypothesis for a small number of events and then modify it as new events become available. In this way, the hypothesis is constructed, based on a small number of events and is *tuned* (or repaired, as in [Michalski 1985]), by generalizing or specializing if necessary to accomodate new events. If the new information is already described by the hypothesis, no modifications are necessary. This is known as *accretion* [Rumelhart & Norman 1977]. In the early learning stages restructuring is often necessary and is best performed with a small data set, which is representative of the entire data set. Given this hypothesis, further tuning is relatively easy to perform, and accretion of new data is trivial. This results in a substantial savings in resources over batch learning, in which a new hypothesis must be generated every time a cover is attempted.

One method of incremental learning utilizes *full memory* [Reinke 1984]. In this method, all input events are retained (remembered) to ensure that later modifications to hypotheses do not create inaccuracies. Another method of incremental learning utilizes *partial memory*. Partial memory implies saving few representative events (or generalizations of small numbers of events) from the event space and "forgetting" the rest. Problems may arise during refinement of the hypotheses when contradictory events are encountered, as illustrated in Fig. 2.

Fig. 2 shows an example of the differences between partial memory and full memory. On the left is the full memory scheme FM, while on the right is the partial memory scheme PM. In order to differentiate the circle events from the square events, two boxes (generalizations) are formed. It can be seen that both methods produce the same hypotheses initially, but upon receiving two additional examples from the class square, partial memory loses consistency at the expense of completeness. In order to achieve complete coverage of the new information, consistency with the original data can be lost. Full memory avoids this problem by retaining all of the input events.

Another important feature of learning is the quality of the hypotheses. One measure of this is *comprehensibility*. A measure of comprehensibility is the inverse of the sum of the number of different attributes, selectors and complexes in the rule [Reinke & Michalski 1985]. For example, in figure 1 the number of selectors is sixteen, the number of different attributes is five, and the number of complexes is one. The sum of these is the *complexity* of the description: 22. The comprehensibility is the inverse of the complexity: 1/22 or .045. Hypotheses generated by incremental learning should not be substantially more complex than those generated by batch learning.

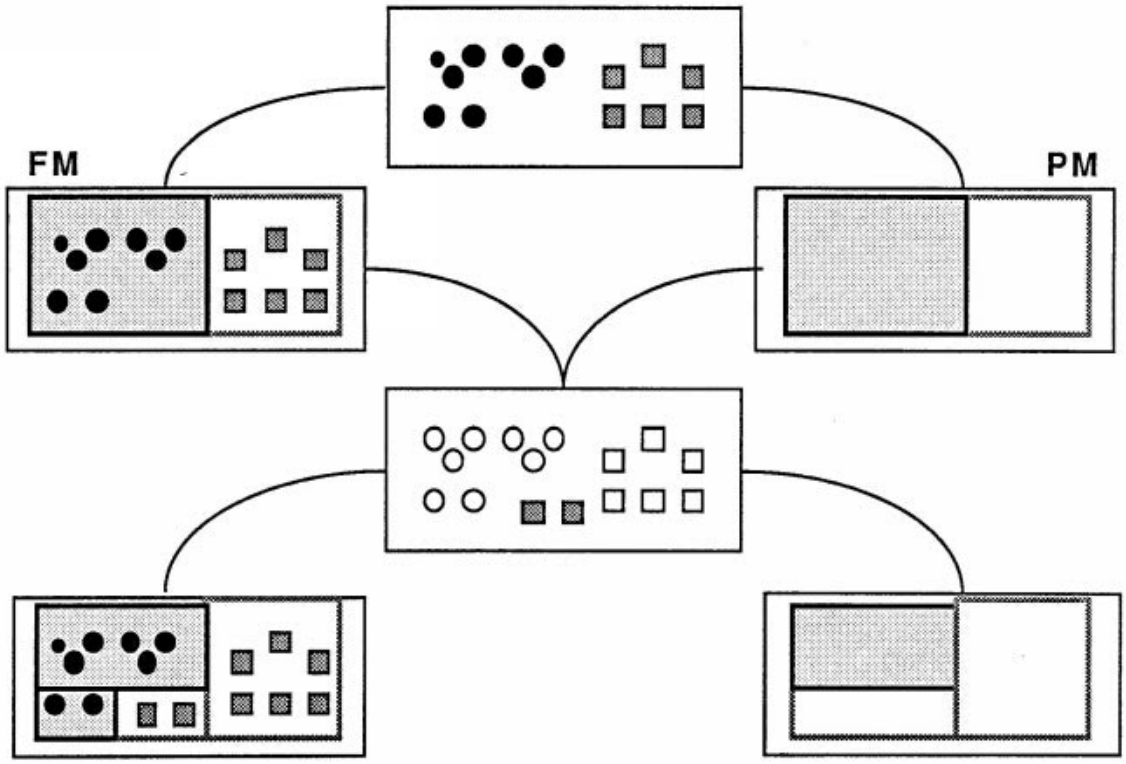


Fig. 2. Full Memory and Partial Memory

Fig. 3 shows a schematic version of the incremental hypothesis generation process. Suppose a new event is entered into the system. If the new event belongs to the class that is being described by the hypothesis, the event is a new positive event. Therefore, if the hypothesis does not cover this new event it must be generalized so that it does cover the event. Similarly, if the new event does not belong to the same class as the hypothesis and is covered by the hypothesis, the hypothesis must be specialized to exclude that event from its hypothesis space. This is similar in principle to a version space [Dietterich 1982, Mitchell 1978].

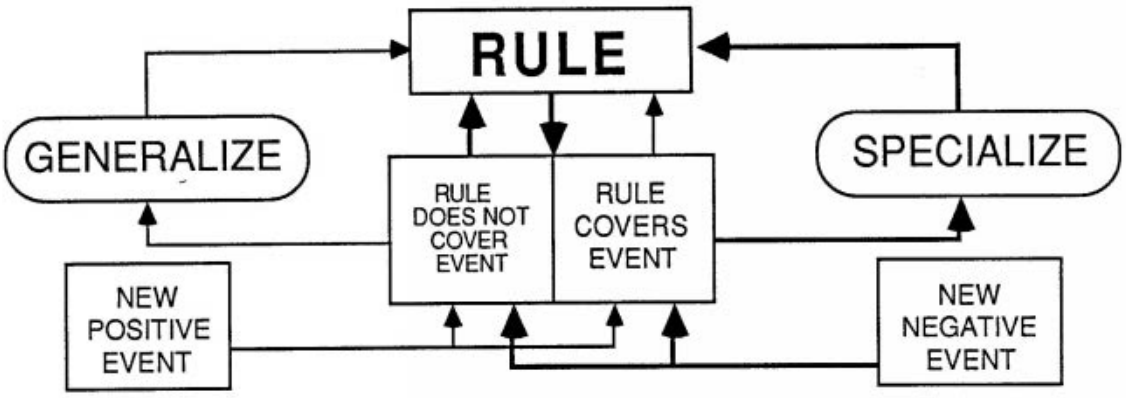


Fig. 3. Incremental hypothesis generation process [Reinke & Michalski 1985]

2. IMPLEMENTATION

INDUCE 4 uses VL_2 logic rules to represent events and hypotheses. The A^q algorithm is used to learn generalized descriptions of events. *INDUCE* is able to learn structural descriptions as well as attribute descriptions. *NEWGEM* [Mozetic 1985], which is able to learn attribute descriptions incrementally, is the basis and inspiration for the incremental additions to the *INDUCE* algorithm. This algorithm is described in Fig. 3 and in the preceding section. The difference between *NEWGEM* and *INDUCE 4* is that the *INDUCE* programs can handle structural descriptions (e.g. `[ontop(x1,x2)]`). Because of this, *INDUCE 4*'s algorithm for incremental learning is slightly different than *NEWGEM*'s.

INDUCE 4 uses a two-phase approach for producing generalized descriptions of events; one for relational predicates and another for unary predicates. Phase one searches the structure-specifying descriptors. Once consistent generalizations are found in the structure-only space, the attribute descriptor space is searched to complete the generalizations.

2.1. ALGORITHM DESCRIPTION

Fig. 4 indicates the primary distinction of incremental and non-incremental modes of operation. When the user specifies a cover of a particular event set, the algorithm of Fig. 4 is performed. The covering procedure calls two main subprocedures. (1) In incremental mode, hypothesis repair is performed. (2) After repairing hypotheses, any remaining events are covered, producing a disjunction. Finally, a Subset Elimination procedure is invoked to prune redundant hypotheses. These algorithms are described further below.

A. **Repair Hypotheses**: This procedure is the core of the incremental learning algorithm. The procedure is as follows:

I. **Make Relevant Hypothesis List.** This procedure makes a list of all the relevant hypotheses for the current event set being covered. There may be more than one hypothesis if a disjunction of descriptions is used to describe an event class.

II. **Make:** This procedure sets up three data structures to be used in subsequent steps. *False negative list* is a list of the negative events that the hypotheses currently cover. *Negative list* is a list of all the negative events in the event space: events that are not members of the current event set. *Positive list* is a list of all the positive events: events that belong to the current event set.

III. **Specialize**: This procedure is called if any false negative events exist. This means that the hypotheses cover some negative events and therefore needs to be specialized. This is done by calling the A^q algorithm with a restricted universe consisting of the false negative events and the positive events. A^q is given the relevant hypotheses list as the initial star, resulting in the specialization of the hypotheses. This works because the A^q procedure restricts the current star against the false negative events. The star is passed to the A^q procedure as the generalized descriptions which need to be extended against the negative events.

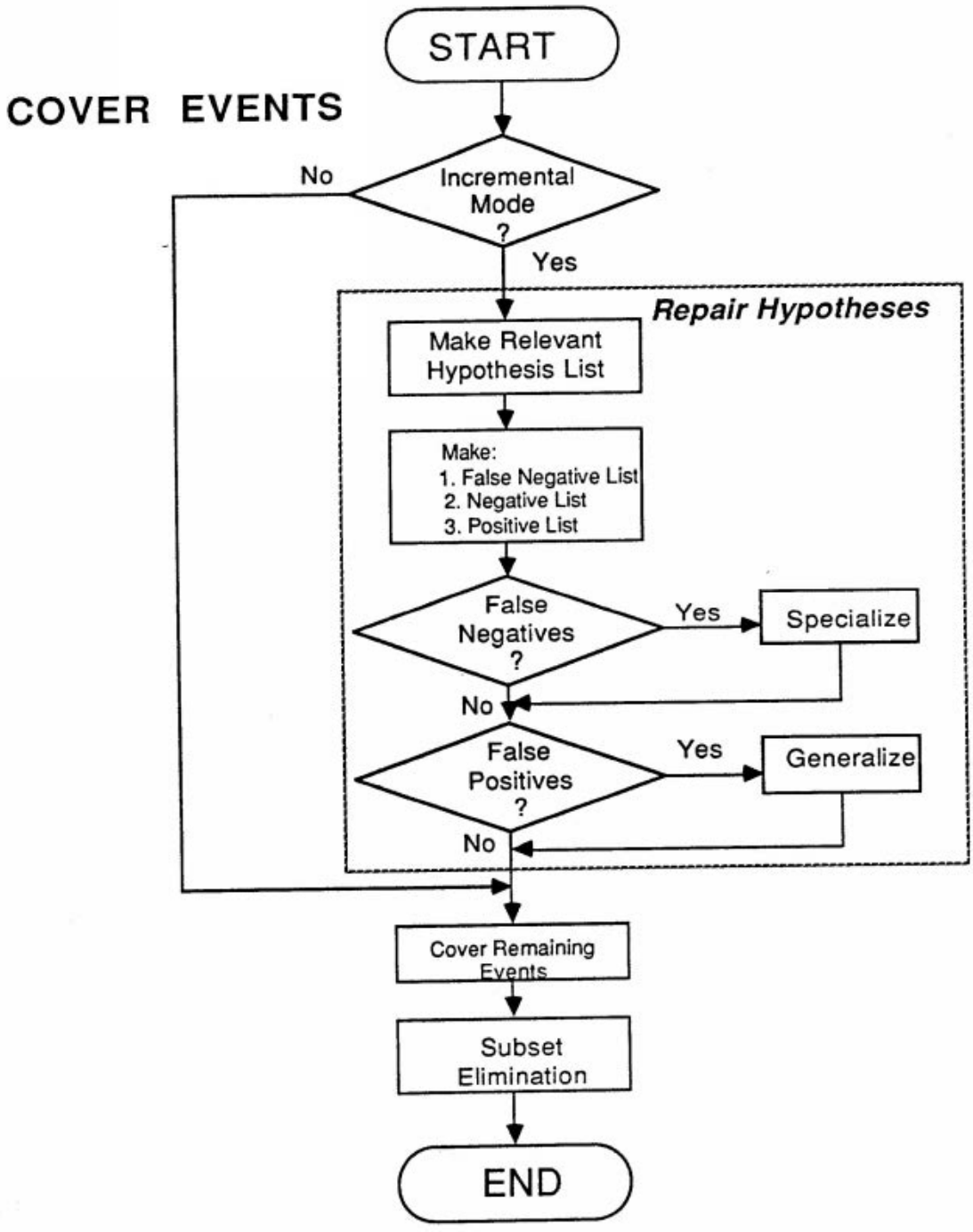


Fig. 4

A FLOWCHART OF THE ALGORITHM

IV. **Generalize**: This procedure is called if there are any new positive events that are not currently covered and should be (false positive). In this case A^q is called with the complete universe to keep it from generalizing too much and thereby covering some of the negative events. The universe consists of all the negative events (negative list) and all the positive events (positive list).

B. **Cover Remaining Events**: This procedure is the original non-incremental *INDUCE* algorithm. It is used by the incremental version in two situations: (1) **Cover Events** is called to produce the initial hypotheses. (2) If the incremental algorithm cannot modify the hypotheses to completely cover all the positive events, new hypotheses must be generated. **Cover Events** may then produce a disjunctive concept description.

C. **Subset Elimination**: In certain cases, the cover may return superfluous hypotheses. The subset elimination procedure removes hypotheses that cover events which are a proper subset of the events covered by another hypothesis. For example, Fig. 5A shows hypothesis H1 covering events 1, 2 and 3, while H2 covers events 2, 3 and 4. Fig. 5B shows the result of deleting event 4. H1 still covers events 1, 2 and 3, but since H2 now covers only events 2 and 3, H2 is subsumed by H1.

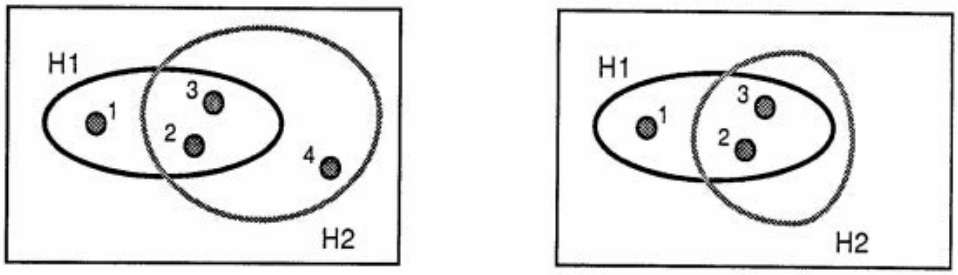


Fig. 5.

In Fig. 5A, H1 (denoted by the black oval) covers events 1, 2, and 3, while H2 (the grey oval) covers events 2, 3, and 4. Fig. 5B is the result of deleting event 4. Since no new events are covered by the additional hypothesis, H2 may be subsumed by H1.

3. ADDITIONAL MODIFICATIONS

3.1. NEW COMMANDS

Two new commands were added to *INDUCE 4* that weren't available in previous version of the system.

1. An additional command to add or delete hypotheses. Similar to the 'm' command in *INDUCE 3* that allows the user to add or delete any event, *INDUCE 4* has the 'y' command that allows the same operations on hypotheses. With this command, the user may guide the learning process by either entering or removing a hypothesis.

2. The addition of a command, 't', that prints both the cumulative and the elapsed CPU time used. This can be used to determine the time required to generate a particular hypothesis.

3.2. PERFORMANCE ENHANCEMENTS

Several changes were made to enhance the performance of the system in the incremental mode of operation.

1. A new data structure was added that indicates whether an event set must be re-covered. In non-incremental mode, a cover operation is performed whenever specified by the user. However, using previously-computed hypotheses, a cover may not be necessary. If no new events have been added by the user, the hypotheses do not need to be modified. Likewise, if an event is deleted from an event set other than the one being covered, the hypotheses are still valid. The data structure that facilitates this tracking of events is an array of flags, one flag for each event set. This alleviates the necessity for checking the hypotheses against all of the events.

2. The hypothesis data type was modified to include a list specifying the events it covers. Previously, this information was computed many times in the cover procedure. In the present version, this information is computed once and stored with the hypothesis.

3. A flag was added that specifies whether incremental mode is enabled. This flag may be modified by the user as a parameter setting using the command `incr=true`. Initially, this flag is set to false for compatibility with *INDUCE 3*.

4. The *INDUCE* input parsing mechanism has been simplified so that less computation is required when reading single characters.

5. Symbolic event set names (e.g. `sulfates`) may now be specified, rather than merely event set numbers when performing covering operations.

3.3. CODE MODULARIZATION FOR COMPREHENSIBILITY

Various procedures of *INDUCE* were modularized to increase their comprehensibility. These include the main command processing loop, the main cover procedure, and the main *VL* rule parsing procedure.

3.4. MISCELLANEOUS WORK

Major and minor "bugs" were repaired as encountered. The major fixes included:

1. Proper exiting from the rule/hypothesis modification mode. Previously, the program terminated when exiting the mode with the `q` command.

2. Correct parsing of numbers. Previously, any item beginning with a number was parsed as a number. For example, the input `3L9` would have been incorrectly parsed as a number.

3. The correction of an error that caused the program to grow monstrously large for certain inputs. The failure to free certain data structures resulted in unbounded growth which eventually led to program termination due to insufficient memory.

3.5. POSSIBILITIES and PROBLEMS

1. *INDUCE 4* uses full memory learning, which does not lend itself to large numbers of events. A partial memory method would be able to handle large numbers of events at the cost of possible inaccuracy. A scheme is needed to deal with uncertainty that may arise due to the *forgetting* of events.

2. A possibility is to allow the attachment of functions to predicates. This would enable internal operations to be performed. For example, a predicate could cause an internal table to be updated each time the predicate is used, or a counter could be decremented each time a predicate is encountered.

3. Although *INDUCE* can perform arithmetic operations, its range of values is limited and the arithmetic rules are difficult to use. The arithmetic values are mapped into a table, obviously limiting their usefulness. A re-implementation of portions of the system that perform these operations seems necessary. The arithmetic rules are limited to the basic operations; the ability to define functions would be useful.

4. The current implementation is approximately ten thousand lines of Pascal. This code has been repeatedly modified and has become quite difficult to understand. Perhaps a rewriting into LISP would aid readability.

5. Subset elimination is only performed for proper subsets. One way to modify this is to use the A^q algorithm to find a cover for all the hypotheses in an event set. This would produce a minimal hypothesis list.

6. An interface that presents the examples in a pedagogical order (i.e. *ESEL* [Michalski & Larson 1978]). In this order, the initial hypotheses would be most representative and easy to tune to accommodate later examples.

7. The *INDUCE* methodology doesn't support uncertainty of attributes. This is important in certain domains, but would require major changes in *INDUCE*.

4. RESULTS

INDUCE 4 has been tested on a variety of data sets with events taken from four domains. These tests involved the correct formulation of hypotheses to describe event classes from the domains of organic chemistry, railroad trains, dinner table place settings, and the sensory input of a robot.

Several "standardized" input data sets were employed to compare the non-incremental (or batch mode) approach of *INDUCE 3* with the incremental operation of *INDUCE 4*. These standardized data sets demonstrate different ways in which large numbers of events may be presented for consideration. For example, batch mode involves a *one-step* cover operation during which all events are covered at one time. Alternatively, a hypothesis may be generated for a small number of events and then modified as additional events are encountered. Four incremental formats were used: *two-step*, *layered*, *random*, and *haphazard*.

First, let us describe how each of these methods is utilized in the *characteristic* mode of operation, whereby only a single class of events is considered at a time. The result of such a cover is the creation of a characterization of the class.

In *one-step* covering, after all of the events of the class have been entered, a cover is performed. This is equivalent to the batch mode operation of *INDUCE 3*.

In *two-step* covering, a hypothesis is generated after two events have been entered. Then the remainder of events for the class are entered and another cover is performed, making use of both the events and the previously generated hypothesis. This covering method seems like it could be the best possible strategy if the initial hypothesis is generated from sufficiently representative events. In theory, if events are selected for maximally effective learning, the hypotheses learned during the first cover will need little modification when new events are subsequently covered. If this is the case, this method could produce the best results.

The *layered* method creates a hypothesis after two events have been entered and then a new cover is performed after the addition of every new event. This is a worst case scenario in which events become available one by one and it is important to maintain an accurate hypothesis.

There is no *random* covering scheme in the characteristic mode. It is subsumed by the *layered* and *haphazard* methods.

The *haphazard* method is similar to the approach used by Reinke [Reinke & Michalski 1985]. Initially twenty percent of the events are submitted and the first hypothesis is generated. Then a random percentage of the remaining events is submitted and a new cover is made until the pool of events has been exhausted.

In *discriminant* mode, each hypothesis formed describes how the particular class may be distinguished from all other classes. Thus, all of the classes are viewed simultaneously and the input data sets must be arranged by the user to reflect this fact.

In discriminant *one-step* covering, a cover is performed on each event class after all of the events of all of the classes have been entered. Again, this is equivalent to the batch mode operation of *INDUCE 3*.

In *two-step* covering, hypotheses are generated after two events from each class have been entered. Then the remainder of events for the classes are entered and another set of covers is performed, making use of the events as well as the previously generated hypotheses.

The *layered* method creates initial hypotheses after two events from each class have been entered. After this, a new set of covers is performed following the addition of one new event from each class. This continues until all of the events have been entered.

In *random* covering, two events from different classes are selected and an initial set of hypotheses is formed. Then, until all of the events have been used, an event drawn from a randomly selected event set is submitted to the system and covers are performed on all of the known classes. This is the worst-case method in discriminant mode since this method requires the largest number of covers.

In the *haphazard* method, twenty percent of the events from each class are submitted and the first hypotheses are generated. Then a different random percentage of the remaining events of each

class is submitted and a new set of covers is made until all of the events have been utilized.

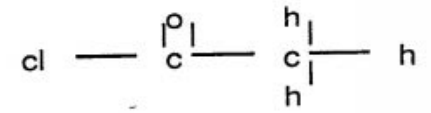
Runs were made in incremental mode as well as non-incremental mode of *INDUCE 4* to verify that both modes would generate similar output for each of these methods. The non-incremental (or batch) mode is what a user would have seen if using *INDUCE 3*. All runs shown were performed on a SUN-2 workstation. *INDUCE* may also be used on VAX and Pyramid computers.

The results may be analyzed based on the criteria of resources used and the complexity of the generated hypotheses. Resource usage is based on the amount of CPU time used. A rule's complexity, assumed to be the inverse of it's comprehensibility, was defined as the number of selectors, number of different attributes and number of complexes in the rule. The complexity of a hypothesis is the average of the complexities of it's disjunctions [Reinke & Michalski 1985].

4.1. ORGANIC CHEMISTRY

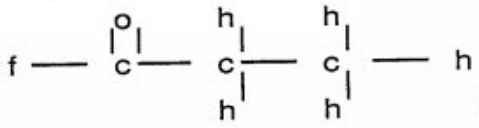
There were 84 input events. These events were from the following classes: alcohol, ether, alkene, alkyne aldehyde, ketone, carbacid, ester, acid anhydride, amide, nitrile, amine, thiol, sulfide, sulfonic acid, sulfate, phosphine phosphate, alkyl halide, acyl halide, and organometallic. Fig. 6 shows the input examples for acylhalide. Fig. 7 shows the resulting rules obtained by both the incremental and nonincremental runs for the 2-step cover. All the results are then summarized in the graph of Fig. 8.

ACYLHALIDE



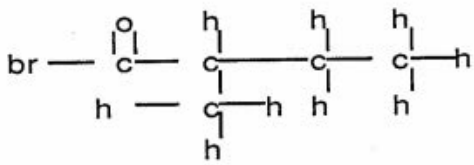
```
[t(a1)=cl][t(a2)=c][t(a3)=o][sb(a1.a2)]
[db(a2.a3)][sb(a2.a4)][t(a4)=c]
[t(a5)=h][t(a6)=h][t(a7)=h][sb(a4.a5)]
[sb(a4.a6)][sb(a4.a7)] -> [d=acylhal].
```

There are seven atoms; the type of atom 1 is chlorine, 2 is carbon, 3 is oxygen, 4 is carbon, 5 is hydrogen, 6 is hydrogen, and 7 is hydrogen. Atom1 has a single bond to atom 2. Atom 2 has a double bond to atom three and a single bond to atom 4. And atom 4 has a single bond to atoms 5, 6 and 7. The domain is acylhal.



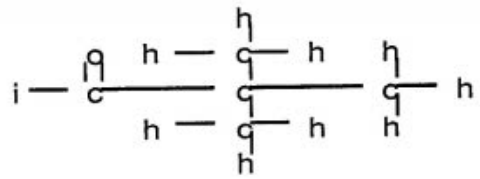
```
[t(a1)=f][t(a2)=c][t(a3)=o][sb(a1.a2)]
[db(a2.a3][sb(a2.a4)][t(a4)=c][t(a5)=h]
[t(a6)=h][sb(a4.a5)][sb(a4.a6)]
[sb(a4.a7)][t(a7)=c][t(a8)=h][t(a9)=h]
[t(a10)=h][sb(a7.a8)][sb(a7.a9)]
[sb(a7.a10)] => [d=acylhal].
```

Here there are ten atoms. Atom 1 is of type f, 2, 4 and 7 are of type carbon, 3 is of type oxygen, and 5, 6, 8, 9 and 10 are of type hydrogen. Atom 1 has a single bond to atom 2. Atom 2 has a single bond to atom 4 and a double bond to atom 3, atom 4 has single bonds to atoms 5, 6 and 7, and atom 7 has single bonds to atoms 8, 9 and 10. The domain is acylhal.



```
[t(a1)=br][t(a2)=c][t(a3)=o][sb(a1.a2)]
[db(a2.a3)][sb(a2.a4)][t(a4)=c][t(a5)=h]
[sb(a4.a5)][sb(a4.a6)][sb(a4.a10)]
[t(a6)=c][t(a7)=h][t(a8)=h][t(a9)=h]
[sb(a6.a7)][sb(a6.a8)][sb(a6.a9)]
[t(a10)=c][t(a11)=h][t(a12)=h]
[sb(a10.a11)][sb(a10.a12)][sb(a10.a13)]
[t(a13)=c][t(a14)=h][t(a15)=h][t(a16)=h]
[sb(a13.a14)][sb(a13.a15)][sb(a13.a16)]
=> [d=acylhal].
```

This example has 16 atoms. Atom 1 is of type bromine, atom 3 is of type oxygen, atoms 2, 4, 6, 10 and 13 are of type carbon and atoms 5, 7, 8, 9, 11, 12, 14, 15 and 16 are of type hydrogen. Atom 1 has a single bond to atom 2. Atom 2 has a double bond to atom 3 and a single bond to atom 4. Atom 4 has single bonds to atoms 4, 6 and 10. Atom 6 has single bonds to atoms 7, 8 and 9. Atom 10 has single bonds to atoms 11, 12 and 13. And, finally, atom 13 has single bonds to atoms 14, 15 and 16.



```
[t(a1)=i][t(a2)=c][t(a3)=o][sb(a1.a2)]
[db(a2.a3)][sb(a2.a4)][t(a4)=c][sb(a4.a5)]
[sb(a4.a9)][sb(a4.a13)][t(a5)=c][t(a6)=h]
[t(a7)=h][t(a8)=h][sb(a5.a6)][sb(a5.a7)]
[sb(a5.a8)][t(a9)=c][t(a10)=h][t(a11)=h]
[t(a12)=h][sb(a9.a10)][sb(a9.a11)]
[sb(a9.a12)][t(a13)=c][t(a14)=h][t(15)=h]
[t(a16)=h][sb(a13.a14)][sb(a13.15)]
[sb(a13.a16)] => [d=acylhal].
```

Here there are also 16 atoms. Atom 1 is of type iodine. Atom 3 is of type oxygen. Atoms 2, 4, 5, 9 and 13 are of type carbon. And atoms 6, 7, 8, 10, 11, 12, 14, 15 and 16 are of type hydrogen. Atom 1 has a single bond to atom 2. Atom 2 has a double bond to atom three and a single bond to atom 4. Atom 4 has single bonds to atoms 5, 9 and 13. Atom 5 has single bonds to atoms 6, 7 and 8. Atom 9 has single bonds to atoms 10, 11 and 12. And atom 13 has single bonds to atoms 14, 15 and 16.

Fig. 6. Acylhalide Examples

In this figure, t stands for type, sb stands for single bond, db stands for double bond, and d stands for the domain of the example.

Output Rules for Acylhalide

Non-Incremental mode:

```
[#as#0,1,2,3,4,5,6,8,11,14,29][t(a1)]
```

Incremental mode:

```
[#as=13,15,16,17,18,19,20,21,22,23,24,25,26,27,28][t(a1)]
or
[#as=6,7,8,9,10][t(a1)]
```

Fig. 7. Output rules for Fig. 6

Here, '#as' is the result of a meta rule that counted the number of a's in the examples. An 'a' is an atom, as in figure 6 description.

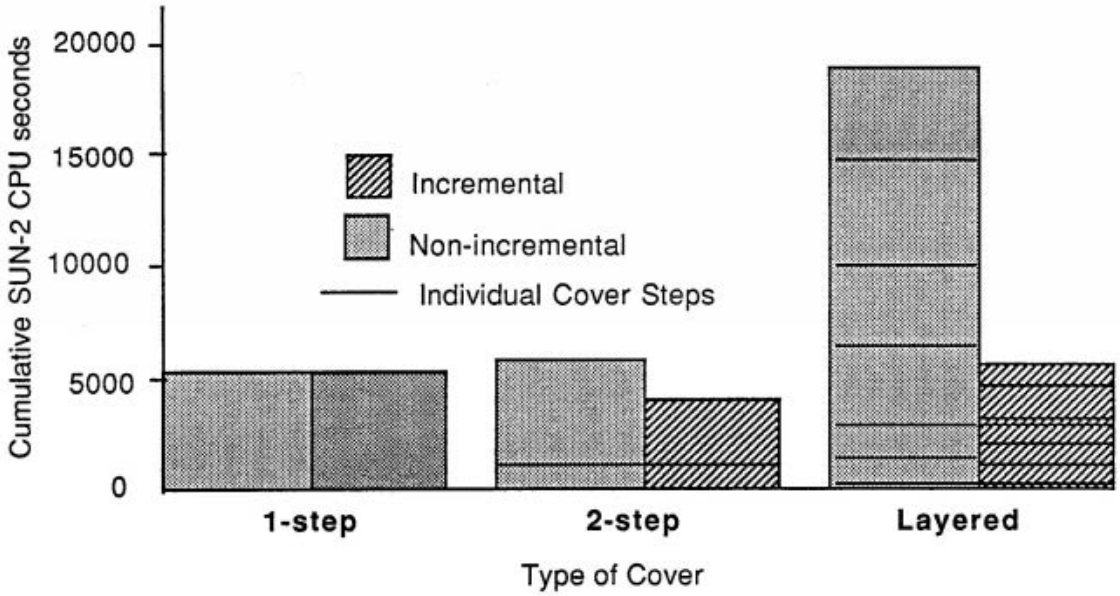


Fig. 8. Discriminant Mode - Organic chemistry examples

The chemistry examples were run in both incremental and non-incremental mode with meta-selectors (constructive induction rules) enabled. The average complexity of the 2-step cover was 11.48 for non-incremental mode and 11.095 for incremental mode, a 3.35% increase in comprehensibility. The average complexity of the layered cover was 11.095 for non-incremental mode and 10.095 for incremenatl mode, a 9% increase in comprehensibility. Although these figures are not significant, it is interesting to note that the incremental mode is not more complex. Also of interest is that the two-step cover of the examples outperformed the one-step method of learning. This suggests that in some cases, when presented with events in a pedagogical order, two-step incremental learning may outperform other methods of learning. It can also be seen that when events occur over a period of time, there are true gains to be made by utilizing the incremental mode of learning as is demonstrated in the layered method example.

4.2. ROBOTS

These examples were contributed by Carl Kadie and serve well to show the benefit of incremental learning. There were 99 events from five different classes in this domain. Each event depicts sensory information of a robot learning to navigate in a restricted environment. The graph shown in Fig. 9 depicts *INDUCE 4* learning rules for the robot's movement events presented to the system in the random method. A cover was performed for every learning class as each new event was entered.

It is interesting to note that after about 190 covers (when approximately 40 events had been presented to the system) the hypotheses seem to have stabilized. The increasing time per cover is a result of checking to ensure that all events are appropriately covered by the existing hypotheses. A partial memory method that remembered only *interesting* (or surprising) events might be used to produce a flatter CPU time per cover line. Interesting events would be those that modify existing hypotheses. These interesting events are indicated by large spikes in the graph.

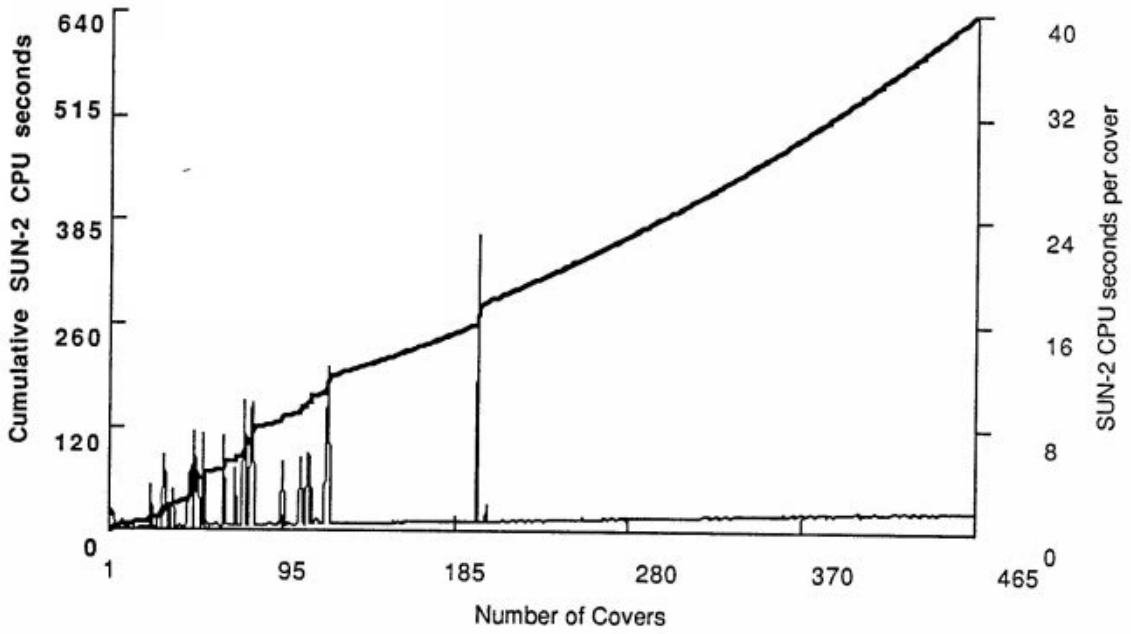


Fig. 9. Discriminant Mode, Random Method - Robot examples

The heavy black line indicates cumulative CPU seconds (the left axis) while the lighter line (lower) corresponds to the time required per cover (the right axis). Note that the left and right axes have different scales.

Whereas the graph in Fig. 9 shows CPU times irrespective of particular event sets, Fig. 8 disseminates this information for each of the event sets. Note that the abscissa of Fig. 10 is different than that of Fig. 9: this is due to the fact that there were one hundred events, and that 464 covers were performed on these events. This allows the reader to see how the hypotheses evolve over time. For example, covers for event set eighteen require a large amount of CPU time initially, but subsequent covers require very little time. The largest spike is the fourth cover of event set eighteen which corresponds to the largest spike of Fig. 9 that occurs on cover 190.

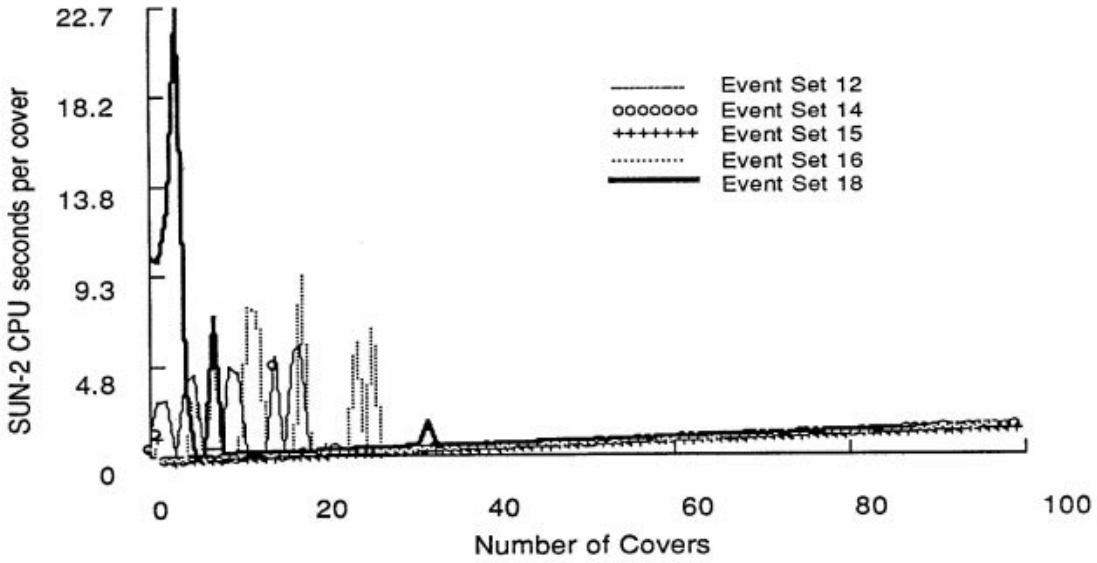


Fig. 10. Discriminant Mode, Random Method - Robot examples

The height of a spike in Fig. 9 and Fig. 10 is proportional to the amount of modification performed on a hypothesis. Initially, large spikes occur before the hypotheses converge to consistent concept descriptions. Once sufficiently accurate descriptions of the actual concepts are formed (those that are consistent with *all* events, including examples not yet seen), new events are easily accommodated (no modifications are necessary). In Fig. 10, after approximately thirty covers were performed on each of the event sets, no substantial modifications were required to be made to the hypotheses. Based on a small number of events (roughly one-third) sufficiently accurate hypotheses were formed so that the remaining events did not disturb the concept descriptions.

This implies that the *predictive power* of *INDUCE 4*'s hypotheses can be quite good. In this example, one hundred percent predictability occurs after one-third (35) of the events have been covered. This means that all of the remaining events would be correctly classified based on the initial third of all examples. While this example gives a rough idea of the predictive power of *INDUCE 4*'s hypotheses, predictive power is not a focus of this paper.

The following edited transcript shows steps one, two, three and the last step of the input and output for the robots example. It gives some idea of the problem that *INDUCE 4* is covering.

Beginning of Transcript

Step1:

Input:
```
[shock(z1)=15][heading(z1)=15][distance(d1)=15]
[distance(f1)=19][distance(l1)=22][distance(r1)=17]
[deltadist(l1)=12][deltadist(r1)=17][touch(u1)=t][touch(l1)=nil]
[touch(d1)=nil][operation(z1)=nothing]
=>
[value=15]

[shock(z1)=15][heading(z1)=16][distance(d1)=15]
[distance(f1)=17][distance(l1)=19][distance(r1)=20]
[deltadist(l1)=13][deltadist(r1)=12][touch(u1)=t][touch(l1)=nil]
[touch(d1)=nil][operation(z1)=right]
=>
[value=14]
```

Output:

Class14:
```
[heading(z1≠15]
```

Class15:
```
[heading(z1)≠16]
```

Step2:

Input:
```
[shock(z1)=15][heading(z1)=16][distance(d1)=15]
[distance(f1)=17][distance(l1)=19][distance(r1)=20]
[deltadist(l1)=13][deltadist(r1)=12][touch(u1)=t][touch(l1)=nil]
[touch(d1)=nil][operation(z1)=nothing]
=>
[value=15]
```

Output:

Class14:
```
[operation(z1)=right]
```

Class15:
```
[operation(z1)=nothing]
```

Step 3:

Input:
```
[shock(z1)=15][heading(z1)=16][distance(d1)=15]
[distance(f1)=16][distance(l1)=19][distance(r1)=20]
[deltadist(l1)=13][deltadist(r1)=11][touch(u1)=t][touch(l1)=nil]
[touch(d1)=nil][operation(z1)=forward]
=>
[value=15]
```

Output:

Class14:
```
[operation(z1)=right]
```

Class15:
```
[operation(z1)≠right]
```

Step N:

Input:
```
[shock(z1)=15][heading(z1)=16][distance(d1)=15]
[distance(f1)=23][distance(l1)=23][distance(r1)=16]
[deltadist(l1)=15][deltadist(r1)=22][touch(u1)=t][touch(l1)=nil]
[touch(d1)=nil][operation(z1)=right]
=>
[value=14]
```

Output:

Class14:
```
[heading(z1)≠15][operation(z1)=right]
```

Class15:
```
[operation(z1)=nothing,forward]
```

End of Transcript.

4.3. PLACE SETTINGS

The examples were from two classes: those that were good place settings, and those that were bad. There were ten place settings for each class. Fig. 11 shows representative input examples, and Fig. 12 shows the results for the various methods of covering. Fig. 13 is a graph showing all the results for the various covering methods in the discriminant mode. Here, two-step learning does not outperform one-step learning as it did with the chemistry examples.

The complexity of the results between incremental and non-incremental mode are significant for this example. The complexity for the non-incremental mode is 29 in all cases. In incremental mode the complexities are 28.5, 24.5, 25.5, and 24 for 2-step, layered, random, and haphazard respectively. In the case of the haphazard cover there is an increase of 17.24% in comprehensibility when using incremental mode. As can be seen in Fig. 12, a complexity of 24 is not that good, it gives a comprehensibility rating of 0.04167. A comprehensibility rating much closer to 0.25 is desired. The chemistry example with a complexity of 10.095 gives a comprehensiblity of 0.099058.

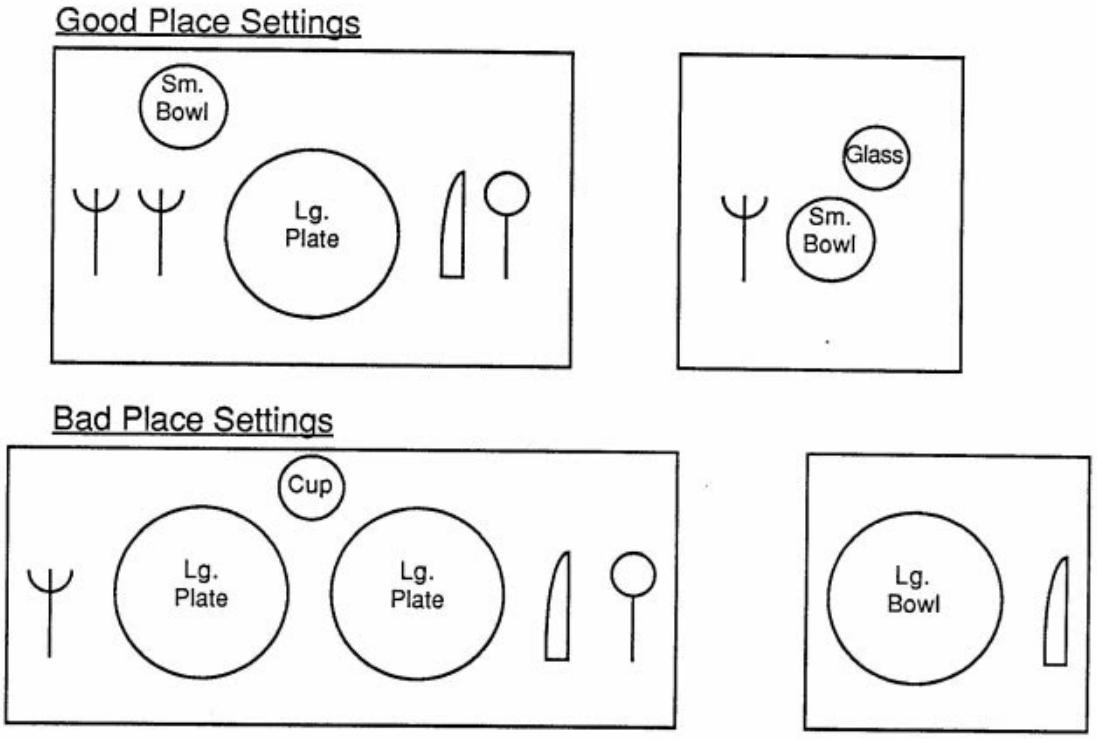


Fig. 11. Place Setting Examples

Non-incremental:

Good Place Settings

```
[pos(p1,p2)≠right,abvleft][size(p2)=small][type(p1)≠spoon][type(p2)≠bowl] or
[pos(p2,p1)=blwrght,abvrght,blwleft][size(p1)=small][type(p2)≠spoon] or
[pos(p1,p2)≠right,abvleft][size(p2)=large][type(p2)≠plate] or
[pos(p1,p2)≠abvleft,abvrght][pos(p2,p3)=left][pos(p2,p4)=abvrght] and
[type(p1)≠knife][type(p3)≠plate][type(p4)=cup] or
[pos(p1,p2)≠abvleft][pos(p3,p2)≠left,abvleft][type(p1)≠plate][type(p3)=knife] or
[pos(p1,p2)=left][pos(p2,p3)=left][pos(p4,p3)≠left,right][type(p4)≠cup]
```

Paraphrase:

Piece one which is not a spoon is not to the right or above left of piece 2 which is small and is not a bowl. Or piece two which is not a spoon is below right, above right or below left of piece one which is small. Or there is a piece one not to the right or above left of another piece two which large and is not a plate. Or There is a piece one which is not a knive that is not above left or above right of a peice two; this piece two is left of a piece three that is not a plate and piece, and furthermore, piece two is also above right of a piece four which is a cup. Or there is a piece one that is not a plate not above left of a piece two which in turn is related to piece three; piece three is a knife and is not to the left or above left of piece two. Or, finally, there is a piece one left of piece two which is left of piece three; there is also a piece four that is not a cup that is not to the left or right of piece three.

Bad Place Settings

```
[pos(p1,p2)≠abvleft][pos(p3,p2)≠left,abvrght][type(p2)≠plate] or
[pos(p1,p3)≠left,right][pos(p2,p3)=abvleft][size(p2)=small][type(p1)≠glass] or
[pos(p1,p2)=left][pos(p2,p3)=left][pos(p3,p4)=left][type(p2)≠plate] and
[type(p3)≠plate] or
[pos(p1,p2)≠abvrght][type(p1)=bvgware] or
[pos(p1,p2)≠right,abvleft][type(p1)=bowl][type(p2)≠spoon]
```

Paraphrase:

There is a piece one that is not above left of a piece two; piece two is not a plate, and a piece three is not to the left or above right of piece two. Or there is a piece one that is not a glass not to the left or right of a piece three; there is also a piece two that is small and is above left of piece three. Or there is a piece one left of a piece two which is not a plate and is left of a piece three which is also not a plate is left of a piece four. Or there is a piece one which is beverageware that is not above right of a piece two. Or there is a piece one that is a bowl that is not right or above left of a piece two that is not a spoon.

Incremental: Haphazard

Good Place Settings

```
[pos(p2,p1)≠right,abvleft][size(p1)=small][type(p2)≠plate] or
[pos(p1,p2)≠right][pos(p3,p2)≠abvrght][size(p2)=large][type(p1)=fork] or
[pos(p1,p2)=left][pos(p2,p3)=left][size(p2)=large][type(p1)=fork] and
[type(p3)≠plate] or
[pos(p1,p2)≠right][size(p2)=large][type(p2)≠plate] or
[pos(p1,p2)≠right,abvleft][size(p2)=small][type(p1)≠spoon][type(p2)≠bowl]
```

Paraphrase:

There is a piece two that is not a plate that is not to the right or above left of a piece one that is not small. Or there is a piece one that is a fork that is not to the right of a piece two that is large, and there is a piece three that is not above right of piece two. Or there is a piece one that is a fork that is left of a piece two which is large and is left of a piece three that is not a plate. Or there is a piece one that is not right of a piece two that is large and is not a plate. Or there is a piece one that is not a spoon that is not right or above left of a piece two which is small and not a bowl.

Bad Place Settings

```
[pos(p1,p3)≠right][pos(p2,p3)≠right][type(p1)=cup][type(p2)≠fork] or
[pos(p1,p2)≠abvleft,abvrght][type(p1)=bvgware][type(p2)≠spoon] or
[pos(p1,p2)≠right,abvrght][pos(p2,p3)=left][type(p1)≠plate][type(p2)≠fork] and
[type(p3)≠knife] or
[pos(p1,p2)≠abvrght][pos(p3,p2)=right,blwrght,blwleft][type(p2)≠plate]
```

Paraphrase:

There is a piece one that is a cup and is not right of a piece three, and there is a piece two that is not a fork and is not right of piece three. Or there is a piece one that is beverage ware and is not above left or above right of a piece two that is not a spoon. Or there is a piece one that is not a plate and is not right or above right of a piece two which is not a fork and is left of a piece three which is not a knife. Or there is a piece one that is not above right of a piece two which is not a plate, and there is a piece three that is right, below right, or below left of piece two.

Fig. 12. Output Rules for Place Settings

In this example, the place settings are described as follows: Each piece is denoted as pN, where N is the (arbitrary) number assigned to each piece of tableware. Each piece has a type, which can be one of: plate, bowl, cup, glass, knife, spoon, or fork. Each piece also has an associated size, which is either small or large. Finally, the relative positions of the pieces are given using the pos predicate. The relative placements are: left, right, above-left, above-right, below-left, and below-right. Abbreviations are used above for each term.

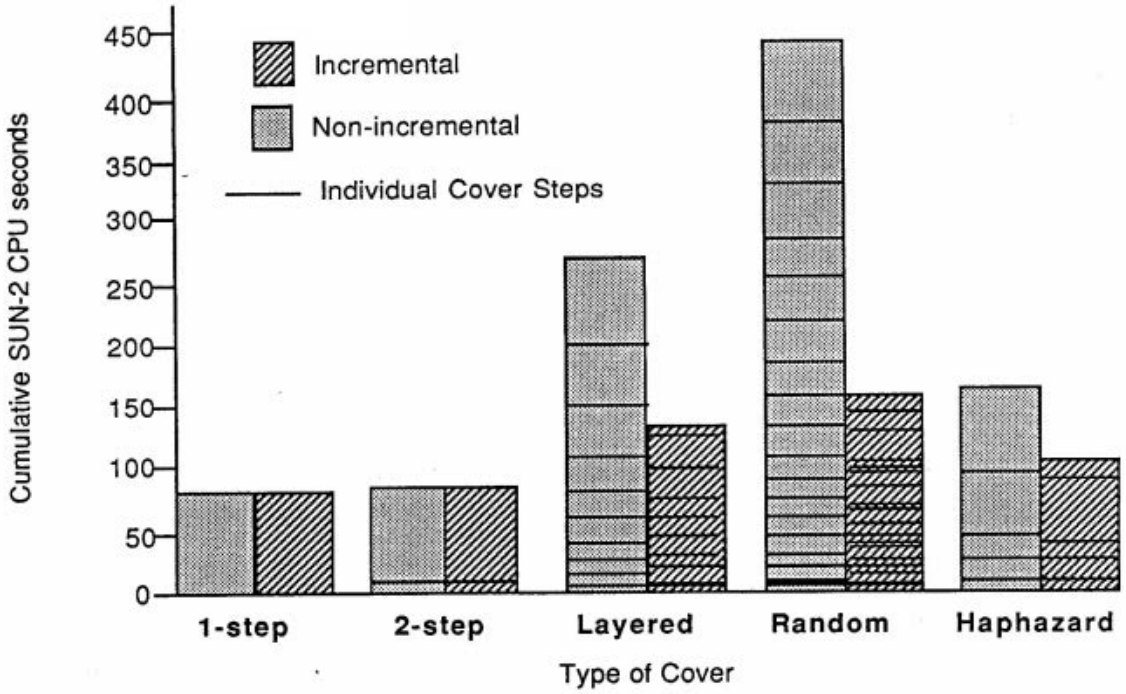


Fig. 13. Discriminant Mode - Place Settings

In the above figure, the time required to generate hypotheses is identical for both incremental and non-incremental mode of the 1-step cover. Even when two covers were performed as specified by the definition of 2-step, roughly the same amount of time was required for both modes. In the other three approaches to covering the events, the incremental mode of operation requires more time than a 1- or 2-step cover, but much less time than the non-incremental mode when performing large numbers of covering operations. For each mode of Layered, Random, and Haphazard, the incremental mode of operation results in the same type of resource usage. In non-incremental mode, however, the time required varied widely as the covering approach varied.

4.4. RAILROAD TRAINS

This data set consisted of East-bound and West-bound trains. Each of the two classes contained five events, as depicted in Fig. 14. The task was to distinguish between East-bound and West-bound trains.

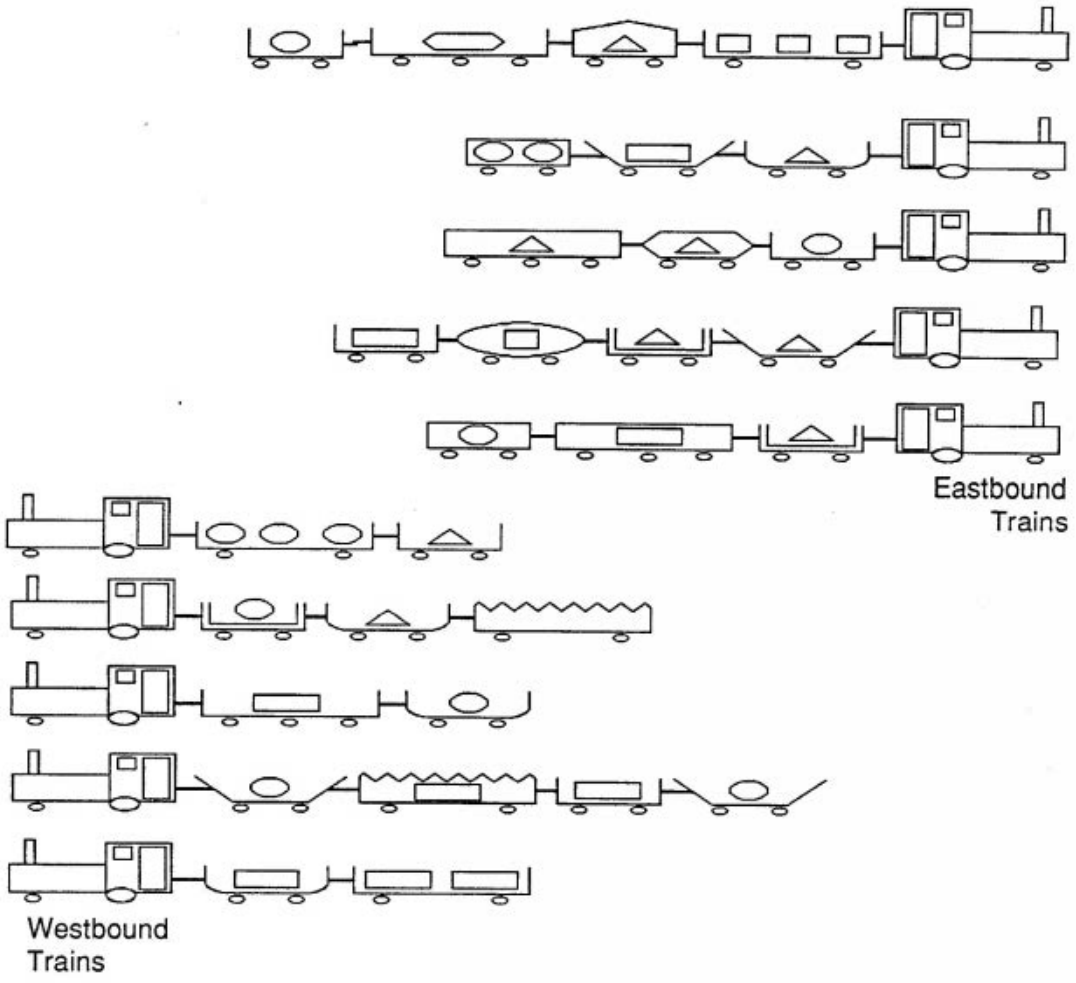


Fig. 14. Railroad Trains

The results (shown in Fig. 15) were essentially similar to the results of the place setting examples. Again the two-step cover was the most efficient when all of the input events were not initially available for covering as in the one-step method. The complexity of the results were exactly the same in both the incremental and non-incremental modes of operation for all types of covers. The complexity was 6, giving a comprehensibility of .1667.

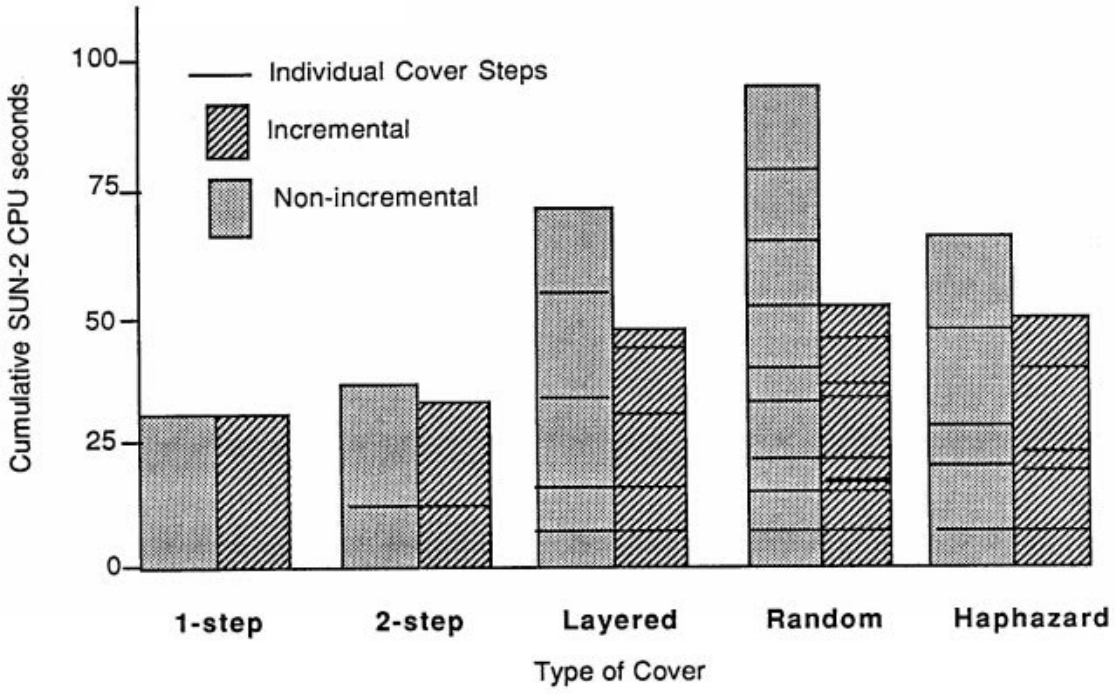


Fig. 15. Discriminant Mode - Railroad Trains

In this example set, the incremental mode outperformed non-incremental in the 2-step approach to covering, as well as in the Layered, Random, and Haphazard approaches.

The following edited transcript shows the example input and output for the Reinke Random method of covering the train examples. This example is a good example showing how much faster the incremental method works as compared to the non-incremental method. Part a. is the non-incremental run while part b. is the incremental run. Each step is a set of input followed by the output as produced by *INDUCE 4*.

Part a. (Non-incremental run)

Step 1:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=3][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=closedrect]
[cshape(car4)=closedrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=triangle][lshape(lod2)=rectanglod][lshaple(lod3)=circlelod]
=>
[dir=east]
```

Paraphrase:

There is an eastbound train containg four cars; one, two, three and four. Car one is in front of car two which is in front of car three, in front of car four. Car one is in location one, car two in location two and so forth. Car one, two, three, and four have two, two, three, and two wheels respectively. The shape of car one is engine shape, car two is double open rectangle shape, car three is closed rectangle shape and car four is also closed rectangle shape. The number of places where car one has a load is zero, car two has one as with car three and car four. Car two contains a triangle load, car three a rectangle load and car four a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=jaggedtop]
[cshape(car4)=openrect][cshape(car5)=opentrap]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=circlelod]
=>
[dir=west]
```

Paraphrase:

There is a westbound train that contains five cars; one, two, three, four and five. Car one is in front of car two and so on down the line. The location of car one is in position one, car two in position two, etc. Each car has two wheels. The length of cars one and three is long, the other cars are short. Car one is an engine. Car two has an open trapezoidal shape, car three a jagged top shape, car four an open rectangle shape and car five has an open

trapezoidal shape. Car one has no load, all the other cars each have one load. Car one has a circleload, cars two and three both have a rectangle load and car four has a circle load.

Output:

Eastbound Trains:
```
[ncar(t1)≠5]
```

Paraphrase:
The number of cars of the train is not five.

Westbound Trains:
```
[ncar(t1)≠4]
```

Paraphrase:
The number of cars of the train is not four.

Step 2:

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=hexagon][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod][lshaple(lod3)=trianglod]
=>
[dir=east]
```

Paraphrase:
This eastbound train has four cars one, two, three and four each in front of the next. They are each in their respective locations. Cars one, two and three have two wheels, while car four has three wheels. The lengths of cars one and four are long, cars two and three are short. Car one has an engine shape, car two an open rectangle shape, car three is shaped like a hexagon and car four has a closed rectangle shape. the number of places where car one has a load is zero while all the other cars each have one load. Car two has a circle load, cars three and four both have triangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=ushaped]
[cshape(car4)=jaggedtop]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=0]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod]
=>
[dir=west]
```

Paraphrase:
The train contains four cars each in front of the other in their respective locations. Each car has two wheels. The

length of cars one and four are long while the other cars are short. Car one is shaped like an engine, car two has a double open rectangle shape, car three is u-shaped and car four has a jagged top. Cars two and three each have one load, the other cars have no loads. Car two has a circle load and car three has a triangle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=circlelod][lshaple(lod3)=circlelod]
[lshape(lod4)=trianglod]
=>
[dir=west]
```

Paraphrase:
This train has three cars. The first one is an engine and has no load. The second car looks like a closed rectangle and has three loads, all circle loads. The third car looks like an open rectangle and has a triangle load. All three cars have two wheels. Only the third car is short, the others are long. This train is traveling west.

Output:

Eastbound Trains:
```
[nwhl(car1)≠2]
```

Paraphrase:
The number of wheels of a car is not two.

Westbound Trains:
```
[cshape(car1)=ushaped] or [ncar(t1)≠4]
```

Paraphrase:
There is a car that is u-shaped or there is a train that does not have four cars.

Step 3:

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=3][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=ushaped]
[npl(car1)=0][npl(car2)=1][npl(car3)=1]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)= rectanglod][lshape(lod2)= circlelod]
=>
[dir=west]
```

Paraphrase:
There is a train with three cars each in front of the other. The first and third cars have two wheels while the second car has three wheels. Cars one and two are long and car three is short. The shape of car one is an engine, car two looks like a closed rectangle and car three is u-shaped. Cars two and three each have one load while the first car has no loads. The load of the second car is a rectangle load and the load of the third car is a circle load. The train is

traveling west.

Output:

Eastbound Trains:
```
[loc(car1)≠2][nwhl(car1)≠2]
```

Paraphrase:
There is a car that isn't in the second position which does not have two wheels.

Westbound Trains:
```
[cshape(car1)=opentop][loc(car1)≠2]
```

Paraphrase:
Thereis a car with an open top that is not in position two.

Step 4:

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short][ln(car4)=long][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=slopetop]
[cshape(car4)=openrect][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)][lcont(car4,lod5)]
[lcont(car5,lod6)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=trianglod][lshape(lod5)=hexagonlod][lshape(lod6)=circlelod]
=>
[dir=east]
```

Paraphrase:
There is a train with five cars each in front of the other and in positions one, two, three, four and five. All the cars but the fourth have two wheels, the fourth car has three. The length of cars one, two and four are long, the others are short. The first car looks like an engine, the second, fourth and fifth cars have an open rectangle shape, and the third car has a sloped top. The third, fourth and fifth cars have one load apiece. The second car has three loads and the first car has no loads. The second car's loads are all rectangle loads. The third car has a triangle load, the fourth car has a hexagon load and the fifth car has a circle load. This train is traveling east.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=opentrap][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car4,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshape(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:

This eastbound train has four cars each in front of the other and in positions one, two, three and four. They all have two wheels. Car one is long and all the others are short. Car one looks like an engine, car two is u-shaped, car three has an open trapezoidal shape and car four has a closed rectangle shape. Cars two and three each carry one load while car four has two loads. Car one has no loads. Cars two and three have triangle loads. Car four has two rectangle loads.

Output:

Eastbound Trains:

```
[cshape(car1)=closedtop][ln(car1)=short] or [npl(car1)=2]
```

Paraphrase:

There is a short car with a closed top or there is a car that is carrying two loads.

Westbound Trains:

```
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]
```

Paraphrase:

There is a car with a jagged top shape, or the train does not have four or five cars.

Step 5:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=dblopnrect]
[cshape(car4)=ellipse][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:

There is an eastbound train with five cars each in front of the other and in locations one, two, three, four and five. They all have two wheels. Only the first car is long, all the others are short. The shape of car one is engine shape. Car two looks like an open trapezoid, car three looks like a double open rectangle, car four looks like an ellipse and car five looks like an open rectangle. Each of the cars except the first car have one load, the first car doesn't have any loads. Cars two and three have triangle loads and cars four and five have rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car3,lod3)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
=>
[dir=west]
```

Paraphrase:
There is a westbound train with three cars in locations one, two and three. They all have two wheels. Cars one and three are long and car two is short. Car one is engine shaped, car two is u-shaped and car three looks like an open rectangle. Car two has one load and car three has two loads. Car one has no loads. All the loads are rectangle loads.
Output:

Eastbound Trains:
```
[cshape(car1)=closedtop][ln(car1)=short]
```

Paraphrase:
There is a short car with a closed top.

Westbound Trains:
```
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]
```

Paraphrase:
There is a car with a jagged top or the number of cars in the train is not four or five.

Part b. (Incremental Run)

Step 1:

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=3][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=closedrect]
[cshape(car4)=closedrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=triangle][lshape(lod2)=rectanglod][lshaple(lod3)=circlelod]
=>
[dir=east]
```

Paraphrase:
There is an eastbound train containg four cars; one, two, three and four. Car one is in front of car two which is in front of car three, in front of car four. Car one is in location one, car two in location two and so forth. Car one, two, three, and four have two, two, three, and two wheels respectively. The shape of car one is engine shape, car two is double open rectangle shape, car three is closed rectangle shape and car four is also closed rectangle shape. The number of places where car one has a load is zero, car two has one as with car three and car four. Car two contains a triangle load, car three a rectangle load and car four a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short][ln(car5)=short]
```

```
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=jaggedtop]
[cshape(car4)=openrect][cshape(car5)=opentrap]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=circlelod]
=>
[dir=west]
```

Paraphrase:

There is a westbound train that contains five cars; one, two, three, four and five. Car one is in front of car two and so on down the line. The location of car one is in position one, car two in position two, etc. Each car has two wheels. The length of cars one and three is long, the other cars are short. Car one is an engine. Car two has an open trapezoidal shape, car three a jagged top shape, car four an open rectangle shape and car five has an open trapezoidal shape. Car one has no load, all the other cars each have one load. Car one has a circleload, cars two and three both have a rectangle load and car four has a circle load.

Output:

Eastbound Trains:

```
[ncar(t1)≠5]
```

Paraphrase:

The number of cars of the train is not five.

Westbound Trains:

```
[ncar(t1)≠4]
```

Paraphrase:

The number of cars of the train is not four.

Step 2:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=hexagon][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod][lshaple(lod3)=trianglod]
=>
[dir=east]
```

Paraphrase:

This eastbound train has four cars one, two, three and four each in front of the next. They are each in their respective locations. Cars one, two and three have two wheels, while car four has three wheels. The lengths of cars one and four are long, cars two and three are short. Car one has an engine shape, car two an open rectangle shape, car three is shaped like a hexagon and car four has a closed rectangle shape. the number of places where car one has a load is zero while all the other cars each have one load. Car two has a circle load, cars three and four both have triangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
```

```
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=ushaped]
[cshape(car4)=jaggedtop]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=0]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod]
=>
[dir=west]
```

Paraphrase:
The train contains four cars each in front of the other in their respective locations. Each car has two wheels. The length of cars one and four are long while the other cars are short. Car one is shaped like an engine, car two has a double open rectangle shape, car three is u-shaped and car four has a jagged top. Cars two and three each have one load, the other cars have no loads. Car two has a circle load and car three has a triangle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=circlelod][lshaple(lod3)=circlelod]
[lshape(lod4)=trianglod]
=>
[dir=west]
```

Paraphrase:
This train has three cars. The first one is an engine and has no load. The second car looks like a closed rectangle and has three loads, all circle loads. The third car looks like an open rectangle and has a triangle load. All three cars have two wheels. Only the third car is short, the others are long. This train is traveling west.

Output:

Eastbound Trains:
```
[nwhl(car1)≠2]
```

Paraphrase:
There is a car that does not have two wheels.

Westbound Trains:
```
[cshape(car1)=ushaped] or [ncar(t1)≠4]
```

Paraphrase:
There is a car that is ushaped or the train does not have four wheels.

Step 3:

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=3][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
```

```
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=ushaped]
[npl(car1)=0][npl(car2)=1][npl(car3)=1]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)= rectanglod][lshape(lod2)= circlelod]
=>
[dir=west]
```

Paraphrase:

There is a train with three cars each in front of the other. The first and third cars have two wheels while the second car has three wheels. Cars one and two are long and car three is short. The shape of car one is an engine, car two looks like a closed rectangle and car three is u-shaped. Cars two and three each have one load while the first car has no loads. The load of the second car is a rectangle load and the load of the third car is a circle load. The train is traveling west.

Output:

Eastbound Trains:

```
[loc(car1)≠2][nwhls(car1)≠2]
```

Paraphrase:

There is a car that is not in the second position that does not have two wheels.

Westbound Trains:

```
[cshape(car1)=ushaped] or [ncar(t1)≠4]
```

Paraphrase:

There is a car that is ushaped or the train does not have four wheels.

Step 4:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short][ln(car4)=long][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=slopetop]
[cshape(car4)=openrect][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)][lcont(car4,lod5)]
[lcont(car5,lod6)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=trianglod][lshape(lod5)=hexagonlod][lshape(lod6)=circlelod]
=>
[dir=east]
```

Paraphrase:

There is a train with five cars each in front of the other and in positions one, two, three, four and five. All the cars but the fourth have two wheels, the fourth car has three. The length of cars one, two and four are long, the others are short. The first car looks like an engine, the second, fourth and fifth cars have an open rectangle shape, and the third car has a sloped top. The third, fourth and fifth cars have one load apiece. The second car has three loads and the first car has no loads. The second car's loads are all rectangle loads. The third car has a triangle load, the fourth car has a hexagon load and the fifth car has a circle load. This train is traveling east.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
```

```
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=opentrap][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car4,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshape(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:
This eastbound train has four cars each in front of the other and in positions one, two, three and four. They all have two wheels. Car one is long and all the others are short. Car one looks like an engine, car two is u-shaped, car three has an open trapezoidal shape and car four has a closed rectangle shape. Cars two and three each carry one load while car four has two loads. Car one has no loads. Cars two and three have triangle loads. Car four has two rectangle loads.

Output:

Eastbound Trains:

```
[npl(car1)=2] or [loc(car1)≠2][nwhl(car1)≠2]
```

Paraphrase:
There is a car that has two loads or there is a car not in the second position that does not have two wheels.

Westbound Trains:
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]

Paraphrase:
There is a car with a jagged top or there is a train that doesn't have four or five cars.

Step 5:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=dblopnrect]
[cshape(car4)=ellipse][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:
There is an eastbound train with five cars each in front of the other and in locations one, two, three, four and five. They all have two wheels. Only the first car is long, all the others are short. The shape of car one is engine shape. Car two looks like an open trapezoid, car three looks like a double open rectangle, car four looks like an ellipse and car five looks like an open rectangle. Each of the cars except the first car have one load, the first car doesn't have any loads. Cars two and three have triangle loads and cars four and five have rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car3,lod3)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
=>
[dir=west]
```

Paraphrase:
There is a westbound train with three cars in locations one, two and three. They all have two wheels. Cars one and three are long and car two is short. Car one is engine shaped, car two is u-shaped and car three looks like an open rectangle. Car two has one load and car three has two loads. Car one has no loads. All the loads are rectangle loads.

Output:

Eastbound Trains:
```
[cshape(car1)=closedtop][ln(car1)=short]
```

Paraphrase:
There is a short car with a closed top.

Westbound Trains:
```
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]
```

Paraphrase:
There is a car with a jagged top or the number of cars in the train is not four or five.

End of transcript.

The following edited transcript shows the example input and output for the 1-step method of covering the train examples. This example is not a good example in the sense that the incremental version does not perform any better than it's non-incremental counterpart. In fact the incremental version runs about one second slower for this example. Part a. is the non-incremental run while part b. is the incremental run. Each step is a set of input examples followed by the output as produced by *INDUCE 4*.

Part a. (Non-incremental run)

Step 1:

Input:

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=3][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=closedrect]
[cshape(car4)=closedrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=triangle][lshape(lod2)=rectanglod][lshaple(lod3)=circlelod]
=>
[dir=east]
```

Paraphrase:

There is an eastbound train containg four cars; one, two, three and four. Car one is in front of car two which is in front of car three, in front of car four. Car one is in location one, car two in location two and so forth. Car one, two, three, and four have two, two, three, and two wheels respectively. The shape of car one is engine shape, car two is double open rectangle shape, car three is closed rectangle shape and car four is also closed rectangle shape. The number of places where car one has a load is zero, car two has one as with car three and car four. Car two contains a triangle load, car three a rectangle load and car four a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=jaggedtop]
[cshape(car4)=openrect][cshape(car5)=opentrap]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=circlelod]
=>
[dir=west]
```

Paraphrase:

There is a westbound train that contains five cars; one, two, three, four and five. Car one is in front of car two and so on down the line. The location of car one is in position one, car two in position two, etc. Each car has two wheels. The length of cars one and three is long, the other cars are short. Car one is an engine. Car two has an open trapezoidal shape, car three a jagged top shape, car four an open rectangle shape and car five has an open trapezoidal shape. Car one has no load, all the other cars each have one load. Car one has a circleload, cars two and three both have a rectangle load and car four has a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=hexagon][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod][lshaple(lod3)=trianglod]
=>
[dir=east]
```

Paraphrase:

This eastbound train has four cars one, two, three and four each in front of the next. They are each in their respective locations. Cars one, two and three have two wheels, while car four has three wheels. The lengths of cars one and

four are long, cars two and three are short. Car one has an engine shape, car two an open rectangle shape, car three is shaped like a hexagon and car four has a closed rectangle shape. the number of places where car one has a load is zero while all the other cars each have one load. Car two has a circle load, cars three and four both have triangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=ushaped]
[cshape(car4)=jaggedtop]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=0]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod]
=>
[dir=west]
```

Paraphrase:

The train contains four cars each in front of the other in their respective locations. Each car has two wheels. The length of cars one and four are long while the other cars are short. Car one is shaped like an engine, car two has a double open rectangle shape, car three is u-shaped and car four has a jagged top. Cars two and three each have one load, the other cars have no loads. Car two has a circle load and car three has a triangle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=circlelod][lshaple(lod3)=circlelod]
[lshape(lod4)=trianglod]
=>
[dir=west]
```

Paraphrase:

This train has three cars. The first one is an engine and has no load. The second car looks like a closed rectangle and has three loads, all circle loads. The third car looks like an open rectangle and has a triangle load. All three cars have two wheels. Only the third car is short, the others are long. This train is traveling west.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=3][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=ushaped]
[npl(car1)=0][npl(car2)=1][npl(car3)=1]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)= rectanglod][lshape(lod2)= circlelod]
=>
[dir=west]
```

Paraphrase:

There is a train with three cars each in front of the other. The first and third cars have two wheels while the second car has three wheels. Cars one and two are long and car three is short. The shape of car one is an engine, car two looks like a closed rectangle and car three is u-shaped. Cars two and three each have one load while the first car has no loads. The load of the second car is a rectangle load and the load of the third car is a circle load. The train is traveling west.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short][ln(car4)=long][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=slopetop]
[cshape(car4)=openrect][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)][lcont(car4,lod5)]
[lcont(car5,lod6)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=trianglod][lshape(lod5)=hexagonlod][lshape(lod6)=circlelod]
=>
[dir=east]
```

Paraphrase:

There is a train with five cars each in front of the other and in positions one, two, three, four and five. All the cars but the fourth have two wheels, the fourth car has three. The length of cars one, two and four are long, the others are short. The first car looks like an engine, the second, fourth and fifth cars have an open rectangle shape, and the third car has a sloped top. The third, fourth and fifth cars have one load apiece. The second car has three loads and the first car has no loads. The second car's loads are all rectangle loads. The third car has a triangle load, the fourth car has a hexagon load and the fifth car has a circle load. This train is traveling east.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=opentrap][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car4,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshape(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:

This eastbound train has four cars each in front of the other and in positions one, two, three and four. They all have two wheels. Car one is long and all the others are short. Car one looks like an engine, car two is u-shaped, car three has an open trapezoidal shape and car four has a closed rectangle shape. Cars two and three each carry one load while car four has two loads. Car one has no loads. Cars two and three have triangle loads. Car four has two rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=dblopnrect]
[cshape(car4)=ellipse][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:

There is an eastbound train with five cars each in front of the other and in locations one, two, three, four and five. They all have two wheels. Only the first car is long, all the others are short. The shape of car one is engine shape.

Car two looks like an open trapezoid, car three looks like a double open rectangle, car four looks like an ellipse and car five looks like an open rectangle. Each of the cars except the first car have one load, the first car doesn't have any loads. Cars two and three have triangle loads and cars four and five have rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car3,lod3)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
=>
[dir=west]
```

Paraphrase:
There is a westbound train with three cars in locations one, two and three. They all have two wheels. Cars one and three are long and car two is short. Car one is engine shaped, car two is u-shaped and car three looks like an open rectangle. Car two has one load and car three has two loads. Car one has no loads. All the loads are rectangle loads.

Output:

Eastbound Trains:
```
[cshape(car1)=closedtop][ln(car1)=short]
```

Paraphrase:
There is a short car with a closed top.

Westbound Trains:
```
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]
```

Paraphrase:
There is a car with a jagged top or the number of cars in the train is not four or five.

Part b. (Incremental run)

Input:
```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=3][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=closedrect]
[cshape(car4)=closedrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=triangle][lshape(lod2)=rectanglod][lshaple(lod3)=circlelod]
=>
[dir=east]
```

Paraphrase:

There is an eastbound train containg four cars; one, two, three and four. Car one is in front of car two which is in front of car three, in front of car four. Car one is in location one, car two in location two and so forth. Car one, two, three, and four have two, two, three, and two wheels respectively. The shape of car one is engine shape, car two is double open rectangle shape, car three is closed rectangle shape and car four is also closed rectangle shape. The number of places where car one has a load is zero, car two has one as with car three and car four. Car two contains a triangle load, car three a rectangle load and car four a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=jaggedtop]
[cshape(car4)=openrect][cshape(car5)=opentrap]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=circlelod]
=>
[dir=west]
```

Paraphrase:

There is a westbound train that contains five cars; one, two, three, four and five. Car one is in front of car two and so on down the line. The location of car one is in position one, car two in position two, etc. Each car has two wheels. The length of cars one and three is long, the other cars are short. Car one is an engine. Car two has an open trapezoidal shape, car three a jagged top shape, car four an open rectangle shape and car five has an open trapezoidal shape. Car one has no load, all the other cars each have one load. Car one has a circleload, cars two and three both have a rectangle load and car four has a circle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=hexagon][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod][lshaple(lod3)=trianglod]
=>
[dir=east]
```

Paraphrase:

This eastbound train has four cars one, two, three and four each in front of the next. They are each in their respective locations. Cars one, two and three have two wheels, while car four has three wheels. The lengths of cars one and four are long, cars two and three are short. Car one has an engine shape, car two an open rectangle shape, car three is shaped like a hexagon and car four has a closed rectangle shape. the number of places where car one has a load is zero while all the other cars each have one load. Car two has a circle load, cars three and four both have triangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=long]
[cshape(car1)=engine][cshape(car2)=dblopnrect][cshape(car3)=ushaped]
[cshape(car4)=jaggedtop]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=0]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)=circlelod][lshape(lod2)=trianglod]
```

```
=>
[dir=west]
```

Paraphrase:

The train contains four cars each in front of the other in their respective locations. Each car has two wheels. The length of cars one and four are long while the other cars are short. Car one is shaped like an engine, car two has a double open rectangle shape, car three is u-shaped and car four has a jagged top. Cars two and three each have one load, the other cars have no loads. Car two has a circle load and car three has a triangle load.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)]
[lshape(lod1)=circlelod][lshape(lod2)=circlelod][lshaple(lod3)=circlelod]
[lshape(lod4)=trianglod]
=>
[dir=west]
```

Paraphrase:

This train has three cars. The first one is an engine and has no load. The second car looks like a closed rectangle and has three loads, all circle loads. The third car looks like an open rectangle and has a triangle load. All three cars have two wheels. Only the third car is short, the others are long. This train is traveling west.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=3][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short]
[cshape(car1)=engine][cshape(car2)=closedrect][cshape(car3)=ushaped]
[npl(car1)=0][npl(car2)=1][npl(car3)=1]
[lcont(car2,lod1)][lcont(car3,lod2)]
[lshape(lod1)= rectanglod][lshape(lod2)= circlelod]
=>
[dir=west]
```

Paraphrase:

There is a train with three cars each in front of the other. The first and third cars have two wheels while the second car has three wheels. Cars one and two are long and car three is short. The shape of car one is an engine, car two looks like a closed rectangle and car three is u-shaped. Cars two and three each have one load while the first car has no loads. The load of the second car is a rectangle load and the load of the third car is a circle load. The train is traveling west.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=3][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=long][ln(car3)=short][ln(car4)=long][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=openrect][cshape(car3)=slopetop]
[cshape(car4)=openrect][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=3][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car2,lod2)][lcont(car2,lod3)][lcont(car3,lod4)][lcont(car4,lod5)]
[lcont(car5,lod6)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=trianglod][lshape(lod5)=hexagonlod][lshape(lod6)=circlelod]
=>
[dir=east]
```

Paraphrase:
There is a train with five cars each in front of the other and in positions one, two, three, four and five. All the cars but the fourth have two wheels, the fourth car has three. The length of cars one, two and four are long, the others are short. The first car looks like an engine, the second, fourth and fifth cars have an open rectangle shape, and the third car has a sloped top. The third, fourth and fifth cars have one load apiece. The second car has three loads and the first car has no loads. The second car's loads are all rectangle loads. The third car has a triangle load, the fourth car has a hexagon load and the fifth car has a circle load. This train is traveling east.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)]
[ncar(t1)=4]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=opentrap][cshape(car4)=closedrect
]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car4,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod][lshape(lod4)=rec
tanglod]
=>
[dir=east]
```

Paraphrase:
This eastbound train has four cars each in front of the other and in positions one, two, three and four. They all have two wheels. Car one is long and all the others are short. Car one looks like an engine, car two is u-shaped, car three has an open trapezoidal shape and car four has a closed rectangle shape. Cars two and three each carry one load while car four has two loads. Car one has no loads. Cars two and three have triangle loads. Car four has two rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)][ccont(t1,car4)][ccont(t1,car5)]
[ncar(t1)=5]
[infront(car1,car2)][infront(car2,car3)][infront(car3,car4)][infront(car4,car5)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3][loc(car4)=4][loc(car5)=5]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2][nwhl(car4)=2][nwhl(car5)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=short][ln(car4)=short][ln(car5)=short]
[cshape(car1)=engine][cshape(car2)=opentrap][cshape(car3)=dblopnrect]
[cshape(car4)=ellipse][cshape(car5)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=1][npl(car4)=1][npl(car5)=1]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car4,lod3)][lcont(car5,lod4)]
[lshape(lod1)=trianglod][lshape(lod2)=trianglod][lshape(lod3)=rectanglod]
[lshaple(lod4)=rectanglod]
=>
[dir=east]
```

Paraphrase:
There is an eastbound train with five cars each in front of the other and in locations one, two, three, four and five. They all have two wheels. Only the first car is long, all the others are short. The shape of car one is engine shape. Car two looks like an open trapezoid, car three looks like a double open rectangle, car four looks like an ellipse and car five looks like an open rectangle. Each of the cars except the first car have one load, the first car doesn't have any loads. Cars two and three have triangle loads and cars four and five have rectangle loads.

```
[ccont(t1,car1)][ccont(t1,car2)][ccont(t1,car3)]
[ncar(t1)=3]
[infront(car1,car2)][infront(car2,car3)]
[loc(car1)=1][loc(car2)=2][loc(car3)=3]
[nwhl(car1)=2][nwhl(car2)=2][nwhl(car3)=2]
[ln(car1)=long][ln(car2)=short][ln(car3)=long]
[cshape(car1)=engine][cshape(car2)=ushaped][cshape(car3)=openrect]
[npl(car1)=0][npl(car2)=1][npl(car3)=2]
[lcont(car2,lod1)][lcont(car3,lod2)][lcont(car3,lod3)]
[lshape(lod1)=rectanglod][lshape(lod2)=rectanglod][lshape(lod3)=rectanglod]
=>
```

```
[dir=west]
```

Paraphrase:
There is a westbound train with three cars in locations one, two and three. They all have two wheels. Cars one and three are long and car two is short. Car one is engine shaped, car two is u-shaped and car three looks like an open rectangle. Car two has one load and car three has two loads. Car one has no loads. All the loads are rectangle loads.

Output:

Eastbound Trains:
```
[cshape(car1)=closedtop][ln(car1)=short]
```

Paraphrase:
There is a short car with a closed top.

Westbound Trains:
```
[cshape(car1)=jaggedtop] or [ncar(t1)≠4,5]
```

Paraphrase:
There is a car with a jagged top or the number of cars in the train is not four or five.

End of transcript.

To summarize our test results, the two-step method seems to be the best for learning incrementally. It has also been shown that when events are selected in a good order, two-step covering in incremental mode may even be faster than one-step covering. It seems that in those cases in which all events are not initially available, *INDUCE 4*, with its incremental learning capability, results in significantly less resource usage while producing rules of similar or better comprehensibility values than the non-incremental *INDUCE 3*.

5. CONCLUSION

Incremental learning of structural descriptions from examples has been shown to be successful. In our tests, *INDUCE 4* required less time to modify hypotheses than it would to generate them in non-incremental mode. The incremental algorithm generated hypotheses that were of comparable complexity to those generated in batch mode, but with a considerable savings in resources. The best performance was achieved when a small number of the most representative examples were given to form the initial hypotheses. *INDUCE 4* is a superior system to use when not all of the events are initially known.

6. REFERENCES

Cohen, Paul P., and Fiegenbaum, E. A. (eds.), *The Handbook of Artificial Intelligence*, William Kaufman, Inc., 1982.

Dietterich, T. G., *Description of Inductive Program INDUCE 1.1*, Department of Computer Science, Internal Report, University of Illinois, Urbana-Champaign, 1978.

Dietterich, T. G., and Michalski, R. S., "Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods", *Artificial Intelligence Journal*, Vol. 16, No. 3, 1981.

Dietterich, T.G., Chapter 14 of *The Handbook of Artificial Intelligence,* William Kaufman, Inc., 1982.

Hoff, W., Michalski, R.S., Stepp, R., *Induce 2: A Program for Learning Structural Descriptions from Examples*, ISG-83-4, UIUCDCS-F-83-904, Department of Computer Science, University of Illinois, Urbana, Il., 1983.

Larson, J., *INDUCE-1: An Interactive Inductive Inference Program in VL_{21} Logic System,* DCS Tech. Report UIUCDCS-R-77-876, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1977.

Michalski, R. S., *Learning by Inductive Inference,* NATO Advanced Study Institute on Computer Oriented Learning Process, France, 1974.

Michalski, R.S., "Pattern Recognition as Rule-Guided Inductive Inference.", *IEEE Transactions on Pattern Anaylsis and Machine Intelligence*, Vol. PAMI-2, No. 4, July 1980.

Michalski, R. S. , "A Theory and Methodology of Inductive Learning", *Machine Learning, An Artificial Intelligence Approach,* (eds. Michalski, R. S., Carbonell, J. B., Mitchell, T.), Tioga Publishing Company, 1983.

Michalski, R. S., *Knowledge Repair Mechanisms: Evolution vs. Revolution,* Report UIUCDCS-F-85-946, Computer Science Department, University of Illinois at Urbana-Champaign, 1985.

Michalski, R. S., Chilausky, R. L., "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnose", *International Journal of Policy Analysis and Information Systems*, Vol. 4, No.2, 1980.

Michalski, R. S., Stepp, R. E., *INDUCE 3: A Program for Learning Structural Descriptions from Examples,* Report of the Dept. of Computer Science, University of Illinois at Urbana-Champaign, in preparation.

Michalski, R. S., Larson, J. B., *Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11*, Report No. 867, Department of Computer Science, University of Illinois, Urbana, Il, May 1978.

Mitchell, T. M.,*Version Spaces: An Approach to Concept Learning*, Report STAN-CS-78-711, PhD Dissertation, Computer Science Department, Stanford University, 1978.

Mozetic, I., *NEWGEM: Program for Learning from Examples; Program Documentation and User's Guide*, Report of the Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1985.

Reinke, R.E., *Knowledge Acquisition and Refinement Tools for the ADVISE META-Expert System*, M.S. Thesis, ISG-84-4, UIUCDCS-F-84-921, Department of Computer Science, University of Illinois, Urbana, July 1984.

Reinke, R.E., and Michalski, R. S., Incremental Learning of Concept Descriptions: A Method and Experimental Results, *Machine Intelligence 11* (ed. Donald Michie), 1985.

Rumelhart, D. E., Norman, D. A., Accretion, Tuning, and Restructuring: Three Modes of Learning, *Semantic Factors in Cognition* , (eds. Cotton, J. W., Klatzky, R.), Lawrence Erlbaum Publisher, 1977.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-87-958	2.	3. Recipient's Accession No.
4. Title and Subtitle INDUCE 4: A Program For Incrementally Learning Structural Descriptions from Examples			5. Report Date February, 1987 6.
7. Author(s) John A. Bentrup, Gary J. Mehler, Joel D. Riedesel			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF DCR 84-06801 ONR N00014-82-K-0186
12. Sponsoring Organization Name and Address National Science Foundation, Washington, DC Office of Naval Research, Arlington, VA			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

The program *INDUCE 4* is a general-purpose *incremental inductive learning* program that transforms symbolic descriptions of real-world events into more general and more useful descriptions of these events. These events may be specified in terms of attribute as well as *structural* descriptors. The program produces such descriptions by performing various generalizing, simplifying and *constructive* transformations on the input descriptions, under the guidance of background knowledge specified by the user. As new events are made available, *INDUCE 4* can incrementally modify what has been proviously generated to reflect this new knowledge.

17. Key Words and Document Analysis. 17a. Descriptors

Machine learning, Concept learning, Inductive inference, Learning from examples, Incremental learning, Structural descriptors, Constructive induction.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 54
	20. Security Class (This Page) UNCLASSIFIED	22. Price