A Representation Scheme for Constructive Induction

P. Vaidya

January 1983

File No. UIUCDCS-F-83-907

ISG 83-7

A REPRESENTATION SCHEME FOR CONSTRUCTIVE INDUCTION

by

Pravin Vaidya

University of Illinois at Urbana-Champaign

Abstract - This report presents a scheme for representing rules for guiding constructive induction. The scheme utilises only the notion of unifying substitutions between logical expressions for interpretation of these rules.

January 1983

ISG 83-7

File No. UIUCDCS-F-83-907

This work was supported by NSF MCS 82-05896 and in part by NSF MCS 82-05166.

1- INTRODUCTION

As constructive induction plays a vital role in hypothesis formation there is a need for a clear formalism to express the construction of new descriptors that express relationships between old descriptors. What is required is a way to declaratively express the kind of constructive induction that has to be done along with a simple interpreter which, given the declarative definition of the new vocabulary to be introduced, will proceed with the actual construction. The main criteria must be the simplicity of expression along with ease of interpretation.

The encoding of new name generation for relationships into procedures has the following drawbacks:-
1- it makes the program difficult to comprehend and therefore still more difficult to modify.
2- it does not allow ease of evaluation of heuristic algorithms which are to some degree domain independent and therefore affects their transfer from one domain to another.
3- it does not allow good abstraction of the induction under progress.
4- it is not suitable for developing tools which will allow programming of constructive induction using high level primitives.

2- REPRESENTATION IN THE FORM OF LOGICAL RULES-

One hopes that the following language for expression of formation of new vocabulary for relationships will solve some of the problems pointed out above.

The constructive induction to be done is expressed in the form of rules which form part of the knowledge base. Moreover a program which has only the ability of finding matching substitutions between logical expressions can act as an interpreter for these rules.

Rules are defined below:-
A rule consists of four parts-
1-a logical expression which is matched against some event,
2-a condition part (which is a boolean-value expression) which is evaluated to see if the rule is applicable,
3-a consequent which is either a list of expressions or a logical expression,
4-a substitution for the free variables in the rule.
In other words a rule is a list of expressions with alternating logical and boolean-value expressions along with a substitution for some or all the free variables in the rule.

A logical expression is a list of atomic logical expressions. It is to be interpreted as the conjunction of its constituents.
An atomic logical expression is (loosely) a selector.

eg. (x v1) is an atomic logical expression and it could be interpreted to stand for x=v1.
eg. (type x 1) means type of x is 1 and is another atomic logical expression.

eg. ((x v1) (type x 1)) is a logical expression.

A boolean-value expression is a list of atomic boolean-value expressions. Before such an expression may be evaluated, each free variable in it must be bound to a value. Such an expression is interpreted as the conjunction of its constituents.

An atomic boolean-value expression is comprised of a relational symbol as its head and a list of n terms as its tail where n is the arity of the relational symbol.

A term is a function symbol followed by a list of n terms where n is the arity of the function symbol.

A term can also be a variable.

eg. (> v1 v2) is an atomic boolean-value expression and stands for v1 greater-than v2.

eg. (= (mod v3 v4) v5) is also an atomic boolean-value expression.

eg. ((> v1 v2) (= (mod v3 v4) v5)) is a boolean-value expression.

A boolean-value expression evaluates to true or false. It is true if each of its consitutents evaluates to true.

The purpose of a logical expression is to match with an event, whereas a boolean-value expression is evaluated once the logical expression part of the rule has matched with some event.

EXAMPLES OF RULES-

```
eg. ((((time v6) (headache v1) (temp normal))     #logical exp. part

     ((not(equal v1 absent)))                        #condition part

     (((time (+ v6 1)) (temp v2) (throat v3))       #consequent part
     (())
     ((follows-headache->temp&throat v1 v2 v3)) )

    () )                                          #substitution part
```

The above rule might introduce in a patient's history of disease development a relationship indicating what kind of temperature and throat conditions followed a state where the patient had a headache.

```
eg. ((((x v1) (y v2) (z v3))          #logical exp. part

     ((> v1 v2))                      #condition part

     ((fun (+ v1 v2))) )              #consequent part

    ((v3 7)) )                        #substitution part
```

The above rule introduces the function "fun"= x + y when x is greater than y and z is equal to 7.

3- RULE INTERPRETATION-

A rule may intrepreted as follows:-
Let the rule be as follows-
L-logical expression, B-condition part, C-consequent,
S-substitution.
1-the substitution S is first applied to L to give another logical expression LNEW,
2-the logical expression LNEW is then matched against some event E,
3-if there is no match then no further action is taken,
4-if there is a match and SMATCH is the matching substitution then the following is done-
Let SPRIME be the substitution "S union SMATCH".
a-the substitution SPRIME is applied to the condition part B to obtain a new boolean-value expression BNEW.
b-BNEW is then evaluated,
c-if BNEW evaluates to false then no further action is taken,
d-if BNEW evaluates to true then the following is done-
e-if the consequent C is a logical expression then SPRIME is applied to C to obtain a new logical expression CPRIME. CPRIME may now be added to the event E in question..
f-if the consequent is a list of expressions LEXP then LEXP along with the substitution SPRIME is a new rule.

EXAMPLE OF RULE INTERPRETATION-

```
((((x v1) (y v2) (z v3))          #logical exp.  part

  ((> v3 4))              #condition part

  ((x v4) (y v5) (z (+ v3 2)))            #consequent part
  (())
  (((dx+y (- (+ v1 v2) (+ v4 v5)) ) )

 () )            #substitution part
```

This rule starts out with a nil substitution.

```
event e1: ((x 2) (y 4) (z 7))
event e2: ((x 4) (y -1) (z 9))
```

A matching with event e1 produces the substitution ((v1 2) (v2 4) (v3 7))
and the rule

```
((((x v4) (y v5) (z (+ v3 2)))      #logical exp.  part

  (())            #condition part

  ((dx+y (- (+ v1 v2) (+ v4 v5))) )   #consequent part

((v1 2) (v2 4) (v3 7)) )  #substitution part
```

The new rule may then be reduced by substituting for the free variables as dictated by the substitution part and then

reducing terms containing no free variables to values. A reduction leads to the rule below-

```
((((x v4) (y v5) (z 9))    #logical exp. part

  (())          #condition part

  ((dx+y (- 6 (+ v4 v5)))) )    #consequent part

 () )        #substitution
```

Note that the substitution part may be reduced to nil once the substution as dictated by the substitution part has been carried out.

A matching of the reduced rule with event e2 would generate the substitution ((v4 4) (v5 -1)) and the logical expression ((dx+y 3)) which could be added to the event e2.

4- ILUSTRATIONS OF REPRESENTATIONS-

The following is a description of a number of examples requiring constructive induction and rules are given which accurately describe the action to be taken.

Example 1-

One is given a set of events each event describing the financial situation of a company along with factors supposed to affect the company's condition.

One has the following descriptors-

productl product2 salesl sales2 capitalexpense r&dexpense fixedcost economicgrowth newemployees organisationstructure year labourrates

The problem is to characterise the profitable years and the loss years. Now the profitabilty of a year not only depends on the conditions in that year but also on the conditions of previous years. For example it would depend on the "r&dexpenses a couple of years earlier" and the "capitalexpense the previous year".

The problem would be handled by doing constructive induction to introduce in every event "r&dexpense a couple of years" earlier and the "capitalexpense of the previous year" and the like and then partitioning into profitable and non-profitable years and generalising.

The following rules express the profit computation and previous year dependence.

```
((((salesl v1) (sales2 v2) (r&dexpense v3) (fixedcost v4)
   (capitalexpense v5))
  (())
  ((profit (- (+ v1 v2) (+ (+ v3 v4) v5)))) )
 () )

((((year v1) (r&dexpense v2))
```

```
  (())
  ((year (+ v1 2)))
  (())
  ((r&d-two-years-ago v2)) )
 () )

((((year v1) (capitalexpense v2))
  (())
  ((year (+ v1 1))
  (())
  ((last-years-capital v2)) )
 () )
```

These rules introduce into an event the capital expenses of the previous year and the r&dexpenses two years earlier in the form of new variables "last-years-capital" and "r&d-two-years-ago".

Example 2-

The given set of events describes the progression of symptoms for a group of persons suffering form the same disease. One has the following descriptors-

person time temp bloodpressure pulse headache throat tongue swellings

Thus each event describes what was the state of a person with respect to variables like temperature,bloodpressure etc. at some given time. The object is to develop a description of the symptoms of the disease for disease identification.

Many a time how the symptoms progressed from one state to another is crucial to the diagnosis of the disease he or she has. Or in other words compound symptoms like "slight headache followed by sore throat and high temperature","yellow tongue rash followed by high pulse and neck swellings" play an important role in disease identification. The rules described above present a simple vehicle for expressing this kind of compound symptom generation via constructive induction.

Below are shown two rules for the sample symptoms discussed above-

headache->temp&throat :- this describes what kind of a headache is followed by what kind of temperature and throat states.

tongue->pulse&swellings :- this indicates the pulse and the kind of swellings that may follow some state of the tongue.

```
((((person v10) (time v1) (headache v2))
  (())
  ((person v10) (time (+ v1 1)) ((temp v3) (throat v4))
  (())
  ((person v10))
  (())
  ((headache->temp&throat v2 v3 v4)) )
 () )
```

```
((((person v10) (time v1) (tongue v2))
  (())
  ((person v10) (time (+ v1 1)) (pulse v3) (swellings v4))
  (())
  ((person v10)
  (())
  ((tongue->pulse&swellings v2 v3 v4)) )
 () )
```

Example 3-

Each event has the descriptors-

time input-x output-y

One wants to find a relationship between the differential output and the differential input. This may be accomplished by first introducing descriptors "deltax" and "deltay" into each event (deltax,deltay standing for the differentials of input-x,output-y) and then generalising (if need be after additional constructive induction).

The following rule accomplishes the construction of the differentials deltax and deltay-

```
((((x v1) (y v2) (t v3))
  (())
  ((x v4) (y v5) (t (+ v3 1)))
  (())
  ((deltax (- v4 v1)) (deltay (- v5 v2))) )
 () )
```

Example 4-

One is given a sequence of words of fixed size k0 over some alphabet A. The problem is to discover some rule describing this sequence. One has the following descriptors-

time position1 position2 . ... position-k0

Each event describes a word in the sequence.

The value of position-i in any event is the character in the i-th position of the word described by the event. A variety of new descriptors may be generated depending on the model schemas being used to generate descriptions. For example say one wishes to introduce the following descriptors-

follows-1--3-4 :- this describes a "follows" relationship between the 3-rd character in the (i-1)-th word and the 4-th character in the i-th word.

diff-2--5-5 :- this is the difference between the 5-th character value in the i-th word and the (i-2)-th word.

The following rules would accomplish the task-

```
((((time v1) (position3 v3) )
  (())
  ((time (+ v1 1)) (position4 v4) )
  (())
  ((follows-1--3-4 v3 v4)) )
 () )

((((time v1) (position5 v2))
   (())
  ((time (+ v1 2)) (position5 v3))
   (())
   ((diff-2--5-5 (- v3 v2)) )
  () )
```

Example 5-

This example illustrates the usage of the condition part of a rule.

Given an event set with the following descriptors-

x1 x2 x3 x4 x5

In the process of trying to dicover algebraic relationships say one wants to introduce the following functions-

fx1x2 :- this describes the function
if (x1 LT x2) then x1 - x2
else x1 + x2

max-x3-x4 :- maximum(x3,x4)

This is accomplished by the four rules below-

```
((((x1 v1) (x2 v2))
  ((< v1 v2))
  ((fx1x2 (- v1 v2))) )
 () )

((((x1 v1) (x2 v2))
  ((not(< v1 v2)))
  ((fx1x2 (+ v1 v2))) )
 () )

((((x3 v1) (x4 v2))
  ((> v1 v2))
  ((max-x3-x4 v1)) )
 () )

((((x3 v1) (x4 v2))
  ((> v2 v1))
  ((max-x3-x4 v2)) )
 () )
```

5- SAMPLE RUNS-

This section lists a set of examples that were run on a preliminary implementation for constructing derived events given a set of events together with a set of rules describing new descriptors to be introduced.

eventlist

((sales 100) (capitalexpense 20) (r&dexpense 11) (fixedcost 13) (economicgrowth 2) (newemployees 150) (year 1976))

((sales 120) (capitalexpense 14) (r&dexpense 15) (fixedcost 12) (economicgrowth 3) (newemployees 40) (year 1977))

((sales 150) (capitalexpense 10) (r&dexpense 19) (fixedcost 17) (economicgrowth 5) (newemployees 200) (year 1978))

((sales 130) (capitalexpense 20) (r&dexpense 15) (fixedcost 15) (economicgrowth 4) (newemployees 20) (year 1979))

((sales 160) (capitalexpense 16) (r&dexpense 17) (fixedcost 17) (economicgrowth 4.6) (newemployees 100) (year 1980))

rulelist

((((year v1) (capitalexpense v2)) (nil) ((year (+ v1 1))) (nil) ((last-years-capital v2))) nil)

((((year v1) (r&dexpense v2)) (nil) ((year (+ v1 2))) (nil) ((r&d-two-years-ago v2))) nil)

((((sales v1) (r&dexpense v3) (fixedcost v4) (capitalexpense v5)) (nil) ((profit (- v1 (+ (+ v3 v4) v5))))) nil)

derived- eventlist

((sales 100) (capitalexpense 20) (r&dexpense 11) (fixedcost 13) (economicgrowth 2) (newemployees 150) (year 1976) (profit 56))

((sales 120) (capitalexpense 14) (r&dexpense 15) (fixedcost 12) (economicgrowth 3) (newemployees 40) (year 1977) (profit 79) (last-years-capital 20))

((sales 150) (capitalexpense 10) (r&dexpense 19) (fixedcost 17) (economicgrowth 5) (newemployees 200) (year 1978) (profit 104) (last-years-capital 14) (r&d-two-years-ago 11))

((sales 130) (capitalexpense 20) (r&dexpense 15) (fixedcost

```
15) (economicgrowth 4) (newemployees 20) (year 1979) (profit
80) (last-years-capital 10) (r&d-two-years-ago 15))

 ((sales 160) (capitalexpense 16) (r&dexpense 17) (fixedcost
17) (economicgrowth 4.6) (newemployees 100) (year 1980)
(profit 110) (last-years-capital 20) (r&d-two-years-ago 19))
```

```
 eventlist

 ((person 1) (time 1) (temp 98) (pulse 74) (headache slight)
(tongue normal) (throat normal) (swellings absent))

 ((person 1) (time 2) (temp 102) (pulse 95) (headache
intense) (tongue yellow) (throat sore) (swellings absent))

 ((person 2) (time 1) (temp 97) (pulse 69) (headache slight)
(tongue white) (throat normal) (swellings absent))

 ((person 2) (time 2) (temp 101) (headache intense) (tongue
yellow) (throat sore) (swellings slight))

 ((person 3) (time 1) (temp 97) (headache intense) (tongue
normal) (throat normal) (swellings absent))

 ((person 3) (time 2) (temp 102) (headache slight) (tongue
white) (throat sore) (swellings slight))
```

```
 rulelist

 ((((person v10) (time v1) (tongue v2)) (nil) ((person v10)
(time (+ v1 1)) (pulse v3) (swellings v4)) (nil) ((person
v10)) (nil) ((tongue->pulse&swellings v2 v3 v4))) nil)

 ((((person v10) (time v1) (headache v2)) (nil) ((person
v10) (time (+ v1 1)) (temp v3) (throat v4)) (nil) ((person
v10)) (nil) ((headache->temp&throat v2 v3 v4))) nil)
```

```
 derived- eventlist

 ((person 1) (time 1) (temp 98) (pulse 74) (headache slight)
(tongue normal) (throat normal) (swellings absent) (tongue-
>pulse&swellings normal 95 absent) (headache->temp&throat
slight 102 sore))

 ((person 1) (time 2) (temp 102) (pulse 95) (headache
intense) (tongue yellow) (throat sore) (swellings absent)
(tongue->pulse&swellings normal 95 absent) (headache-
```

```
>temp&throat slight 102 sore))

 ((person 2) (time 1) (temp 97) (pulse 69) (headache slight)
(tongue white) (throat normal) (swellings absent)
(headache->temp&throat slight 101 sore))

 ((person 3) (time 1) (temp 97) (headache intense) (tongue
normal) (throat normal) (swellings absent) (headache-
>temp&throat intense 102 sore))

 ((person 3) (time 2) (temp 102) (headache slight) (tongue
white) (throat sore) (swellings slight) (headache-
>temp&throat intense 102 sore))
```

```
eventlist

((time 1) (input 4) (output 7))

((time 2) (input 5) (output 9))

((time 3) (input 7) (output 13))

((time 4) (input 10) (output 19))

((time 5) (input 14) (output 27))


rulelist

((((time v1) (input v2) (output v3)) (nil) ((time (+ v1 1))
(input v4) (output v5)) (nil) ((deltain (- v4 v2)) (deltaout
(- v5 v3)))) nil)


derived- eventlist

((time 1) (input 4) (output 7))

((time 2) (input 5) (output 9) (deltain 1) (deltaout 2))

((time 3) (input 7) (output 13) (deltain 2) (deltaout 4))

((time 4) (input 10) (output 19) (deltain 3) (deltaout 6))

((time 5) (input 14) (output 27) (deltain 4) (deltaout 8))


eventlist

((x1 7) (x2 3) (x3 15) (x4 11) (x5 1))

((x1 -1) (x2 -6) (x3 0) (x4 5) (x5 7))

((x1 -4) (x2 -3) (x3 16) (x4 8) (x5 17))

((x1 4) (x2 8) (x3 9) (x4 14) (x5 9))

((x1 4) (x2 6) (x3 11) (x4 15) (x5 6))


rulelist

((((x3 v1) (x4 v2)) ((> v2 v1)) ((max-x3-x4 v2))) nil)

((((x3 v1) (x4 v2)) ((> v1 v2)) ((max-x3-x4 v1))) nil)
```

```
((((x1 v1) (x2 v2)) ((> v2 v1)) ((fx1x2 (- v2 v1)))) nil)

((((x1 v1) (x2 v2)) ((> v1 v2)) ((fx1x2 (- v1 v2)))) nil)


derived- eventlist

((x1 7) (x2 3) (x3 15) (x4 11) (x5 1) (max-x3-x4 15) (fx1x2
4))

((x1 -1) (x2 -6) (x3 0) (x4 5) (x5 7) (max-x3-x4 5)  (fx1x2
5))

((x1 -4) (x2 -3) (x3 16) (x4  8)  (x5  17)  (max-x3-x4  16)
(fx1x2 1))

((x1 4) (x2 8) (x3 9) (x4 14) (x5 9) (max-x3-x4 14)  (fx1x2
4))

((x1 4) (x2 6) (x3 11) (x4 15) (x5 6) (max-x3-x4 15) (fx1x2
2))
```

ACKNOWLEDGEMENTS

The author gratefully acknowledges the financial support he obtained from the Department of Computer Science of the University of Illinois at Urbana-Champaign. He also expresses his thanks to W. Hoff for fruitful discussions concerning the material in this report.

REFERENCES

1. R. S. Michalski, "A Theory and Methodology of Inductive Learning," Chapter in the book MACHINE LEARNING, R. S. Michalski, J. Carbonell, and T. Mitchell (Eds.), In press, 1982.

2. R. S. Michalski, "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, July 1980, pp. 349-361.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-83-907	2.	3. Recipient's Accession No.

4. Title and Subtitle

A Representation Scheme for Constructive Induction

5. Report Date: January 1983

6.

7. Author(s)

P. Vaidya

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, IL

10. Project/Task/Work Unit No.

11. Contract/Grant No.

MCS 82-05896
MCS 82-05166

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, DC

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This report presents a scheme for representing rules for guiding constructive induction. The scheme utilizes only the notion of unifying substitutions between logical expressions for interpretation of these rules.

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

Machine Learning
Induction
Constructive Induction
Knowledge Representation
Expert Systems

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 14
	20. Security Class (This Page) UNCLASSIFIED	22. Price