# A SYSTEM FOR GENERATING EXPLANATIONS OF PROLOG PROGRAM RESULTS

Ephraim Paz

1985

Cognitive Studies Research Paper
Serial No. CSRP. 048

The University of Sussex,
Cognitive Studies Programme,

# A SYSTEM FOR GENERATING EXPLANATIONS OF PROLOG PROGRAM RESULTS.

Submitted to the Third International Conference on Logic Programming

Ephraim Paz,
Cognitive Studies programme,
University of Sussex

ABSTRACT:

The better explanation a system can give for its results, the more useful it is. Results of PROLOG programs may appear quite cryptic to naive users, and even experienced programmer may be puzzled by the result, especially if some time has passed since they wrote the program.

This paper outlines a suggestion for a system that will generate explanations for results of PROLOG programs, which will have many advantages over existing explanation mechanisms. The system will use different verbs for describing actions done by the interpreter, and different explanations for different rule types. It will intelligently prun some steps in the execution trace of the program, and will employ rhetorical rules for creating English explanations.

The general approach will include three major elements: extending the expressive power of the language, creating a raw explanation by parsing the execution trace according to a grammar of traces, and dynamic generation of English explanations from this raw explanation.

SECTION 1. INTRODUCTION
1.1 RESTRICTIONS ON KNOWLEDGE REPRESENTATION IN PROLOG.
1.2 USING A SIMPLE TRACE AS AN EXPLANATION.
SECTION 2. THE SYSTEM.
2.1 AN EXAMPLE.
2.2 FEATURES OF THE EXPLANATION
2.3 AN OVERVIEW OF THE SYSTEM.
2.4 TARGET USER.
SECTION 3. ADDITIONAL INFORMATION AND ITS USE
3.1 PROCEDURE LEVEL
3.2 RULES.
3.3 60ALS.
SECTION 4. CONCLUSION

## SECTION 1. INTRODUCTION

Strictly speaking, a PROLOG system, regarded as a theorem prover, produces very succinct answers to queries posed to it: either YES or NO. This, naturally, is not very illuminating, and every PROLOG interpreter produces also the values with which the variables were unified in order to reach a positive answer.

In order to make the system's answer even more plausible, it is relatively easy to devise a mechanism for supplying the execution trace of a program, and to present it to the user.

This paper outlines a suggestion for a system that will generate explanations for results of PROLOG programs, which will have many advantages over existing explanation mechanisms.

The first section describes some problems of representing knowledge in PROLOG, and goes on to explain the shortcomings of using the execution trace of a program as an explanation of its results. The second section outlines the proposed system, and in the third I discuss additional information it requires, how this information will be obtained, and how it will be utilised for the generation of better explanations.

### 1.1 LIMITATIONS OF KNOWLEDGE REPRESENTATION IN PROLOG.

PROLOG is a highly parsimonious language. Its syntax is very simple, uniform and restricted, as is the inference mechanism. The elegance and power of PROLOG lies in these characteristics of the language, but they impose also some limitation on representing knowledge in PROLOG programs. In the process that starts with representing knowledge as a PROLOG program and ends with the final solution or proof, essential information is either lost or becomes implicit. This section describes some of the reasons for this situation.

Representing a problem by using just one uniform knowledge representation construct poses some limitations on the resulting representation. In the case of PROLOG, all knowledge is represented either as "rules" or as "facts". DAVIS [DAVIS,80] describes the general problems of knowledge - representation systems that use rules as the basic uniform representation unit. One of these problem is this: Knowing the mechanism that will be used for invoking the relations or state of affairs, and unless there are some supplementary annotations, the original meaning is quite hard to reconstruct. So, it is sometimes the case that literals in a rule's body are not simply conjoined, but they actually represent another logical structure (e.g. implication).

Having just one uniform syntactic way of representing relations between predicates (the rule construct), the exact relations between the head and the body of rules are implicit and obscure.

## 1.2 USING A SIMPLE TRACE AS AN EXPLANATION.

The trace of the successful unifications done by the PROLOG interpreter may be considered as a good first approximation for an explanation of the reasoning done by a PROLOG program. Existing explanation mechanisms like [HAMMOND,82], [CLARK,82], and [WALKER,83] are in most cases just little more than the execution trace. In this section I will point at some of the reasons why this explanation is far from being satisfactory.

A trace that includes all the steps taken by the interpreter can be much too laborious as it includes also the cases where unification failed, and this can be after many successful steps. If, on the other hand, we give only the steps that have succeeded, some useful information is lost (e.g. when a case really succeeded just because the earlier ones failed as in having a last, default case in a procedure).

The role a predicate plays in the reasoning may vary, according to its Instantiation Status (which arguments are instantiated when the predicate is activated). A simple trace gives us only the final bindings of the predicate, and this information is lost. From the declarative point of view, some literals in the body of a rule may be more essential in establishing the truth of the head, a fact which is not taken into consideration in existing explanation mechanisms.

The trace might include many parts which are not interesting, like steps taken for "book-keeping" (e.g. incrementing an index during a recursive call), or the specific way some system predicates are operating. In other cases, the information may, as a rule , interest only a certain type of user - like in the case of "normalisation" rules which change, let's say, the order of the arguments in a predicate, and may be interesting to a programmer but not to someone whose main interest is in the reasoning process. The solution taken by Walker in [WALKER,83] is to select only certain predicates to be included in the explanation, but this solution is not adequate in this case, because different occurrences of the predicate may have different roles and importance.

A PROLOG program may have two interpretations: declarative and procedural. Having one uniform explanation form may result in poor intelligibility - emphasising always just one aspect of the program.

## SECTION 2. THE SYSTEM.

In this section I give an example of the kind of explanation I expect to get from my system, and some ways in which this explanation is better then explanations one may expect to get from existing explanation mechanisms. This is followed by an overview of the proposed system and its target user.

## 2.1 AN EXAMPLE.

Consider the following simple program:

```
same_den(Country, Other):-
       density(Country, Density1),
       density(Other, Density2),
       Other = Country,
       similar(Density1, Density2).

density(Country, Density):-
       pop(Country, Pop),
       area(Country, Area),
       Density is Pop/Area.

similar(N1,N2):- N1 <N2, similar(N2,N1).
similar(N1,N2):- M =:= N1 - N2,
       N =:= M/N1,
       N < 0.05.

pop(china, 825). pop(belgium, 582). pop(japan, 58).
area(china, 3380). area(belgium, 1139). area(japan, 114).
```

A GOAL:

```
    ?-same_den(japan, X).
```

THE ANSWER:

```
    yes. X=belgium.
```

THE TRACE:

```
same_den(japan, belgium).__________________(1)
       density(japan, 0.508)_______________(2)
           pop(japan, 58)._________________(3)
           area(japan, 114)._______________(4)
           0.508 is 58/114.________________(5)
       density(belgium, 0.514).____________(6)
           pop(belgium, 586).______________(7)
           area(belgium, 1139).____________(8)
           0.514 is 586 / 1139_____________(9)
       japan=belgium______________________(10)
       similar(0.508, 0.514)______________(11)
           0.508<0.514____________________(12)
           similar(0.514, 0.508)__________(13)
               0.006 is 0.514 - 0.508.____(14)
               0.011 is 0.006/0.514.______(15)
               0.011<0.05.________________(16)
```

SAMPLE EXPLANATIONS:

To the question HOW? (i.e. HOW was the answer deriv
following explanation will be given:

EXAMPLE I.

EXAMPLE II.
user: HOW 6b?
system: The density of belgium was calculated as 0.514.
This was done in the following way:
6b.1) The population of Belgium (582) was retrieved from the database.
6b.2) The area of Belgium (1139) was retrieved from the database.
6b.3) The density was calculated as 582/1139 (=0.514).

EXAMPLE III.
user: WHY 2
system: (i.e. WHY is it true that the population density of japan is 0.508)
The population density of japan is 0.508 because
density is defined as population/area
and the population of japan is 58
and the area of japan is 114.

## 2.2 FEATURES OF THE EXPLANATION

(Numbers in brackets refer to lines of the trace)

The user may control the level of details by asking to go deeper into one statement, and there are two flavours of explanation - HOW prompts a procedural explanation, and WHY a declarative one.

The first explanation is built of 4 phrases, but they do not correspond to the goals in the top same_density rule. The second goal in the rule was expanded into two phrases, while the third goal, (japan=belgium), was omitted, because the role of this goal is of secondary importance, namely checking for exceptions. The last phrase does not correspond to line (11), but to the "normalised" version of the "similar" predicate, which had been executed is lines (13)-(16). (This can be recognised by the reverse order of the numbers 0.514 and 0.508 in phrase 3 of explanation I).

In the procedural explanations (I & II), different verbs are used to describe the way goals are satisfied - verbs like "calculate", "suggest" and "compare". The verbs are chosen according to the goal type which was computed according to information about the instantiation status of the arguments and the predicate's keys.

In the third, declarative, explanation, the fact that the "density" rule is a definition helps in explaining the relationship between its body and the head. The order of the phrases does not follow the order of the goals in the rule, but is determined by their relative importance.

## 2.3 AN OVERVIEW OF THE SYSTEM.

The general approach will include three major elements: extending the expressive power of the language, creating a raw explanation by parsing the execution trace according to a grammar of traces, and dynamic generation of English explanations from this raw explanation.

classifications. The grammar will reflect the acceptable ways of combining elements in each level. A. Bundy [BUNDY,84] mentions some common schemas that appear in PROLOG programs. One goal of the system is to identify such schemas, (either by getting the schema as an annotation in the source or as a result of parsing the trace) , and use this information to direct the explanation process.

The trace tree of a program execution will be parsed according to the grammar, and parts of the trace that will be identified will be handed over to the appropriate explanation routine.

So, for example, there will be a certain way of explaining the results of a "case analysis" procedure, or a specialised routine for handling the trace of a recursive rule. On a lower level, rhetorical rules will direct the building of sentences out of groups of goals, and the system will use different verbs to describe different uses of terminal goals.

## 2.4 TARGET USER.

The target user of the system may come from one of two classes - end users and programmers. End users may use the system for two reasons - for getting better understanding of the way the program functions, and for getting deeper insight into the knowledge that lies underneath it. Programmers will usually use it as a debugging aid - and can also benefit from the annotation system, using it for documentation purposes. No specific assumptions are taken regarding the expertise of the user, and the fact that different levels of explanation are available enables the user to decide for himself about the exact level of details he gets.

## SECTION 3. ADDITIONAL INFORMATION AND ITS USE

### 3.1 PROCEDURE LEVEL

The highest level of annotation is the procedure level. In addition to giving names to procedures, it is sometimes important to state the roles of the different occurrences of the predicate. (It may be argued that a PROLOG program should be written so that rules are independent and order should not play a role in their meaning, but many times this ideal is not achieved, and this system is trying to explain programs as they are written).

For example, a procedure may be built as a number of cases, and possibly end with a "catch all" predicate. If, for example, the third occurrence of the predicate succeeded, the system will add to its explanation the fact that the preceding cases failed. If the "catch-all" case succeeds, the dynamic approach and additional information regarding the procedure are even more vital, because the isolated fact that succeeded may be virtually meaningless.

### 3.2 RULES.

In standard PROLOG syntax all rules are treated equally. Additional knowledge about the type of rule can help in creating better explanations. Some rule types describe the declarative

Sometimes the type indicates the syntactic structure of a rule, as in different types of RECURSIVE RULEs. The type of the rule will control the template through which it will be explained (in the example in sec 2.1 there are templates for a definition and for action rules). In the case of recursive rule, a special routine should be called and the recursant part will be omitted. Other rule types like normalisation rules can be simply omitted from the explanation.

## 3.3 GOALS

### 3.3.1 GOAL TYPES.

A terminal of the Raw Explanation represents the trace of a goal that has no sub goals. Such a goal can be one of three types: A special case in a procedure, a system predicate, or a fact from the database.

Some terminals can be better explained not on the isolated predicate level, but in the wider context of the procedure of which they form a component, like the TERMINATION CLAUSE in a recursion, (which can usually be dropped), or a CATCH ALL case in a case-analysis, as was mentioned in the preceding section.

System predicates can usually be omitted from the explanation, with the exception of arithmetics and comparison predicates - in many cases they may be instrumental in the explanation, and we give them the types CALCULATE and COMPARE. If there are more than one or two CALCULATE predicates in one rule, it may be advisable to use an explanation that gives the formula rather than show the actual calculation.

The last class of goals are facts. Sometimes a goal is used to generate a possible solution, some goals are called partly instantiated and are used to retrieve values of one or more variables, and sometimes the goal is already fully instantiated when called, and it is activated just in order to confirm the truth of a predicate.

I have developed a way of finding this "action_type" of the goal by combining knowledge of the "key[11] of the predicate, (the argument position the value of which determines a unique occurrence of the predicate in the database) and its Instantiation Status (which arguments were instantiated when the predicate was activated).

This classification can give us in many cases the best verb to describe the action done by the interpreter and the types of the terminals may even bubble up and give types to the goals that evoked them. When, for instance, we have only calculations as sub goals, we can assume that the top goal is also "calculate".

### 3.3.2 DECLARATIVE IMPORTANCE

Looking at a use of a rule as a declarative statement, that is as a true statement with all the variables instantiated, not all the elements in the body bear the same importance in relation to the head. Some of the literals in the body may be only conditions, others can appear as a means of screening exceptions, while just one or two literals are the main part of the rule. Take, for example the rule :

```
 same_den(Country, Other):-
```

because it is just a means for rejecting unwanted answers for "Other". (1) and (2) serve to retrieve information for the main part, and thus they do not deserve to appear as equal constituents of the explanation, but rather in an auxiliary role. This will help in choosing the best rhetorical strategy for building the sentence that will explain the proof.

As the final phase of the explanation generation will be the generation of English sentences from PROLOG predicates, the programmer should also provide a template for each predicate - the way he wishes it to be expressed in English. For example, the predicate

fly(From, TO, Airline, Flight_Number) might have the following template: "One can fly from From to To, using Airline flight number Flight_number".

## SECTION 4. CONCLUSION

The better explanation a system can give for its results, the more useful it is. Results of PROLOG programs may appear quite cryptic to naive users, and even experienced programmer may be puzzled by the result, especially if some time has passed since they wrote the program.

This paper has described the outlines of a system for generating explanations of results of PROLOG programs. The system will provide explanations which have several benefits over existing explanation mechanism, like using different verbs for describing actions done by the interpreter, different explanations for different rule types, inclusion of additional parts of a procedure, intelligent pruning of some steps in the execution of the program, and employing rhetorical rules for creating English explanations.

This is achieved by adding information as annotation to the program, general knowledge the system has about PROLOG, and dynamic generation of the final explanation.

The execution trace is parsed according to a grammar based on classification of goals, rules and procedures, and the parsed trace directs routines that generate an English explanation.

## REFERENCES

BUNDY,84 A.BUNDY
Simple PROLOG Prototypes
University of Edinburgh
Code Note 6, August 84.

CLARK,82 K.L. Clark & F.G. McCabe
'Prolog:A Language for Implementing Expert Systems'
MACINE INTELLIGENCE v.10 ed. J.E.HAYES, MICHIE,D &
Y.PAO John Wiley & Sons, 1982

DAVIS,80 R.DAVIS
'Meta-Rules: Reasoning About Control'
Artificial Intelligence V.15 pp.179-222

HAMMOND,82 P.Hammond
Logic Programming for Expert Systems
technical report: DOC 82/4 Imperial College.

WALKER,83 A. Walker
Automatic Generation of Explanations of Results
From Knowledge Bases
RJ 3481 (41238) IBM Lab San Jose Cal.

WALLIS,82 J.W.Wallis, E.H.Shortliffe
'Explanatory Power for medical Expert Systems:
Studies in the Representation of Causal Relationships
for Clinical Consultation' Methods of Information
in Medicine V.21, 127-136