International Journal of Computational Engineering Research||Vol, 04||Issue, 1||

Slicing Algorithm for Controlling Backtracking In Prolog

Divanshi PriyadarshniWangoo
CSE & IT Department
ITM University, Gurgaon, Haryana, India

ABSTRACT:
This paper focuses on building an efficient slicing algorithm for controlling backtracking in prolog. Prolog has a built-in backtracking mechanism which alleviates the programmer of explicit backtracking. The automatic backtracking mechanism is often veiled from the user of the system. Uncontrolled backtracking causes inefficiency in a program and may lead to interrupted execution. Slicing technique renders the structural relationship through the ordering of clauses and goals. It facilitates the backtracking prevention mechanisms through relative structuring of parent goal and the relative cut in the program. The algorithm is preferential for the events where the cut and negation mechanisms break down under certain conditions. The slicing algorithm is called BTrack_Slicer and it takes as input the Backtrack Dependency Graph (BDG) which comprises of all dependencies arising out of backtracking. Employing the algorithm for use in various application programming structures would lessen the vast execution time expended as a consequence of indefinite backtracking.

KEYWORDS: *Backtracking, Backtrack Dependency Graph (BDG),BTrack_Slicer, Dynamic System Dependence Graph (DSDG) System Dependence Graph (SDG),Prolog.*

I. INTRODUCTION

Prolog is a programming language for non-numeric and symbolic computations describing relationships between objects. It comprises of a set of basic mechanisms including pattern matching, tree based data structuring and automatic backtracking. Spatial relationships between objects are an easy mechanism in prolog.Relations between objects can be defined by two approaches- either defining relations by facts or defining relations by rules. A prolog program basically consists of clauses followed by one or more goals. Prolog clauses are of three types-facts, rules and questions. Facts declare things that are unconditionally true, rules hold things that are true depending on a given condition and the user can ask the program what things are true through questions [1]. Recursive rules help in determining the predecessor relation in a family hierarchy program. The programming flow in prolog is defined in the form of structural relations and processing queries to those relations. The process of determining that whether an object satisfies a query by the means of questions is often a complicated procedure involving logical inferences, exploring alternatives and backtracking. The cognitive process of backtracking in prolog system is automatic and is often shrouded from the user. Uncontrolled backtracking leads to errant execution of prolog clauses and goals. Therefore there is a need of some strategy involving backtracking that would work efficiently in the situations where the basic backtracking mechanisms fail under certain conditions. One such structural strategy is slicing which relates the co generic association between the prolog clauses and goals. This paper is centralized on designing of backtracking algorithm using dynamic slicing technique. The algorithm is called BTrack_Slicer which takes as input the Backtrack Dependency Graph (BDG). The BDG is a dependency graph which would relate the clauses in a prolog program with their respective goals. The BTrack_Slicer algorithm is a novel backup approach for the prevention of backtracking. It works effectively specially in the cases where the cut and negation process fails in determining the valuable correspondence between the declarative and procedural meaning of the programs.

II. PAPER ORGANIZATION

The remaining part of the paper is organized as follows. In Section three, the different slicing techniques are discussed. Section four states the method of defining the prolog's recursive rules through slicing. The designing of the backtracking algorithm with the help of slicing is discussed in section five of the paper. Section six discusses the results of the BTrack_Slicer algorithm. The last section concludes the paper.

III. SLICING TECHNIQUES

Slicing technique is associated with the construction of program slices which are computed with the help of slicing criterion. The slicing criterion <v, S> defines those part of the program statements that are actually affected by the slicing criterion, where v is the variable name defined at the program statement number S [2]. Program slicing is done with the construction of the System Dependence Graph (SDG) which is a directed dependency graph that relates the dependencies occurring between the statements in a program [2]. Researchers have defined various types of slicing depending on the type of programming applications, the basic types are static slicing and dynamic slicing. Static slicing determines all the parts of a program that might have an effect on the slicing criterion and certainly do not make any assumptions regarding the input [3]. Dynamic slicing defines the dependencies that occur in a specific execution of a program and has an advantage in the run time execution of the program [3,4]. Dynamic slicing is a more powerful slicing technique as it takes the run time execution race of the program variables and exactly states the actual affected statements. The dynamic slicing criterion takes as input the program variable and the statement number. The statements which are actually affected by the slicing criterion are determined as compared to static slicing which states all the possible input statements. This results in the construction of precise slices at run time and eliminates unnecessary statements. Thus, dynamic slicing proves to be the highly efficient in determining run time execution traces of a program. The backtracking algorithm designed in this paper would use the dynamic slicing strategy that will help in setting the breakpoints at run time execution of the prolog program. The goals would be satisfied for their related clauses based on the dynamic slicing criterion at run time of the program.

The main aim of using dynamic slicing in the backtracking algorithm is its precise computation of slices which would be built graphically in the form of Backtrack Dependence Graph (BDG). The BDG would relate the dependency between the rules of the clauses. The mutual dependency between the rules of the clauses would be depicted in the form of arcs defined through dependency edges. The breakpoints where the goals fail would determine the backtracking path in the BDG. The BDG would be taken as input in the BTrack_Slicer algorithm that works efficiently where the cut and negation fails under the closed world assumption.

IV. STATING PROLOG'S RECURSIVE RULES THROUGH SLICING

Prolog's recursive rules helps in determining the predecessor relationship that exists in a set of clauses. The predecessor relation can be defined with the help of two rules- direct and indirect. The former aims at determining the immediate predecessors and the second determine the indirect predecessors [1]. The slicing criterion that would be taken here for the parent relationship example is dynamic slicing. The Dynamic System Dependence Graph (DSDG) is specially designed for the recursive procedure rules in prolog. The DSDG would determine the direct as well as indirect predecessors by taking the slice points in the graph. The solid directed lines represent direct control dependency and dashed lines represent indirect control dependency between the nodes. The family relation program is defined in Fig.1 and its corresponding DSDG is defined in Fig2 as follows.

```
F1   offspring (Y, X) :-
F2   parent (X, Y).
F3   mother (X, Y):-
F4   parent (X, Y),
F5   female (X).
F6   grandparent (X, Z):-
F7   parent (X, Y),
F8   parent (Y, Z).
F9   sister(X, Y):-
F10  parent (Z, X),
F11  parent (Z, Y),
F12  female (X),
F13  different (X, Y).
F14  predecessor (X, Z):-
F15  parent(X, Z).
F16  predecessor (X, Z):-
F17  parent (X, Y),
F18  predecessor (Y, Z).
```

Fig.1 The family hierarchy program in prolog

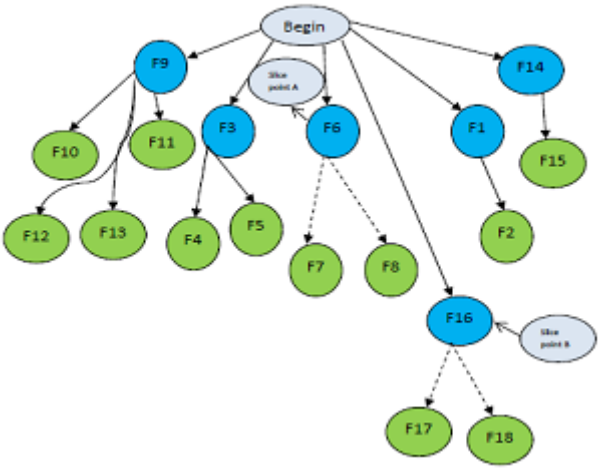

Fig.2 The DSDG of Fig.1

The conventions used for the DSDG of Fig.2 are as follows. The blue color nodes represent the head clauses and the green color nodes represent the body clauses of the program in Fig.1. The DSGD is useful in getting the indirect relationships between the clauses defined through recursive rules. The slice point A and slice point B are the points where the predecessor indirect relationship occurs in the program.

V. BACKTRACKING ALGORITHM USING DYNAMIC SLICING

Backtracking in prolog is an automatic process and it executes under the necessary constraints of satisfying a goal. There are situations which lead to uncontrolled and indefinite backtracking process which leads to inefficiency in the program execution. Although, prolog has in built mechanisms to prevent and control backtracking, but these mechanisms prove to be disadvantageous as there are certain reservations against their use. One of the reservations for the use of backtracking prevention mechanism like cut comes from the fact that by using cut in a prolog program, it can lose the valuable correspondence between the declarative and procedural meaning of the programs. Basically there are two levels of meaning of prolog programs- the declarative meaning and the procedural meaning. The former is concerned only with the relations defined by the program whereas the latter specifies how the prolog system actually evaluates the relation. Change in the order of the clauses in the presence or absence of cuts can lead to two different outputs which would further lead to ambiguous results. In such cases there is need of a backtracking algorithm that would be using the dynamic slicing technique for the prevention of backtracking in the absence of cuts. The backtracking algorithm using dynamic slicing is called BTrack_Slicer that sets the breakpoints in the program that would lead to backtracking. The BTrack_Slicer algorithm takes as input the Backtrack Dependency Graph (BDG) and marks the backtrack breakpoints through the computation of slice points. These slice points are called BTrack slice points and would work as an alternative to the cut mechanism. Thus, the algorithm turns out to work more efficiently in the absence of prolog's in built backtracking prevention mechanisms.

Backtrack Dependency Graph (BDG)

The Backtrack Dependency Graph (BDG) is a directed dependency graph where the nodes of the graph represent the clauses and the arcs or the edges between the nodes determine the dependencies existing between the structural clauses. The BDG is basically designed to represent the relationships between the objects. It is used as an input to the BTrack_Slicer algorithm for the processing of the BTrack slice points and sets up the breakpoints which would trail the backtracking path. The example program that would be used for the construction of the BDG is to find the largest of the three numbers. The program and its corresponding BDG are represented in Fig.3 and Fig.4 respectively.

```
L1  max (X, Y, Z, Max)
L2  max (X, Y, Z, X):-
L3  X>=Y,
L4  X>=Z.
L5  max (X, Y, Z, Y):-
L6  X<Y,
L7  Z<Y.
```

```
L8   max (X, Y, Z):-
L9   X<Z,
L10  Y<Z.
```

Fig.3 Example program of backtracking to find the largest of the three numbers

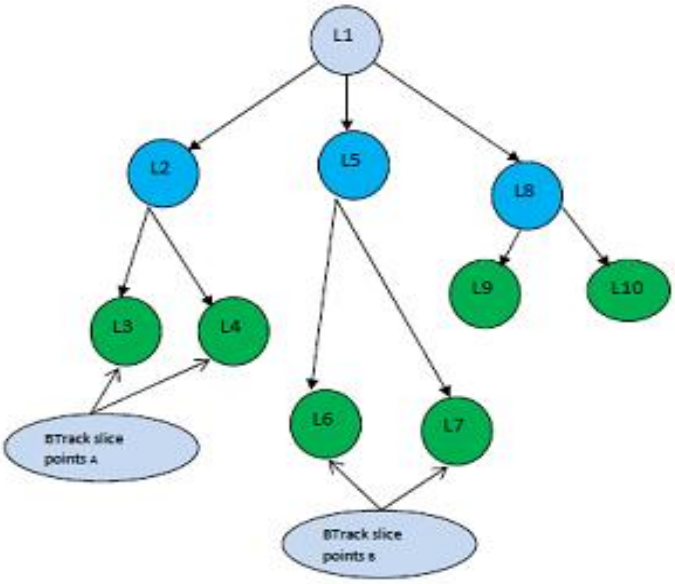

Fig.4 BDG of Fig.3

The program in Fig.3 computes the maximum of three numbers. The blue color nodes represent the head clauses and the conclusion part and the green color nodes represent the body clauses and the condition part. Its corresponding BDG is illustrated in Fig.4 which computes the slice points at the nodes where breakpoints are to be set. The BTrack slice points A are computed at L3 and L4 nodes and the BTrack slice points B are processed at nodes L6 and L7. These slice points work as a substitute to cut points which tells the prolog system not to backtrack beyond the point marked by the cut. If the condition nodes L3 and L4 of the body clause at the BTrack slice points A succeed then the prolog system will prevent futile backtracking that would lead to falsifying of all the remaining goals and the conclusion part of the corresponding head clause will be processed. Thus, multiple results for the same clauses will not come in the output. Similarly, if the condition nodes L6 and L7 of the body clause at the BTrack slice points B succeed then no further backtracking will take place and the corresponding conclusion part of the head clause will be processed. Thus, the execution time will be saved which would otherwise have been exhausted with the unavailing backtracking mechanism.

BTrack_Slicer

The BTrack_Slicer algorithm is a backtracking prevention algorithm which would help in the minimization of execution time at a faster rate. It takes as input the BDG and the backtrack slicing criterion <S, c> where S is the statement number and c is the condition part of the body clause at that particular statement. This algorithm would assist in the decision process of processing BTrack slice points. The condition nodes at the slice points would act as decision nodes that would tell the system whether to backtrack or not.

BTrack_Slicer Algorithm:-

Input: BDG and backtrack slicing criterion <S, c>. H represents the head clause and C represents the condition clause. B_A and B_B represents the Btrack slice points A &B.

Output: Backtrack Slice points A & B

Step 1. Construct the BDG of the prolog program.

Step 2. Compute the slicing criterion <S, c> with some variable inputs and repeat the following.

(a) For all i=1 to i<a where a= number of condition clauses in body part, repeat until i=0.

(b) if (c1==true)

(c) if(c2==true)
(d) then stop backtracking and print the corresponding H, B_A & B_B.
(e) elseif (c1==false)
(f) elseif (c2==false)
(g) then proceed backtracking
(h) else backtrack(c1, c2)
(i) else backtrack c(i)
(j) result print false goals
(k) else print output H, B_A &B_B
(l) stop

Step 3. End

VI. RESULTS AND ANALYSIS OF BTrack_Slicer ALGORITHM

The BTrack_Slicer algorithm takes the dynamic slicing criterion execution trace <S, c> for the condition part of the body clauses and gets the slice point as output which will locate the dependency between the node clauses and prevent the backtracking at that point. The Table 1 below gives an accurate analysis of the BTrack_Slicer algorithm and locates the actual number of statements affected and which would be the execution path in the case where backtracking prevails. The algorithm is accurate under all conditions and is the best alternative solution to the inbuilt backtracking prevention mechanism.

Table 1

BTrack_Slicer Algorithm	**Execution traces**	**Backtracking points in BTrack_Slicer**	**Statements affected**	**Accuracy attained**
BTrack_Slicer	<L3, X>	BTrack slice points A	L2, L1	Yes
	<L4, X>	BTrack slice points A	L2, L1	Yes
BTrack_Slicer	<L6, Y>	BTrack slice points B	L5, L1	Yes
	<L7, Y>	BTrack slice points B	L5, L1	Yes

VII. CONCLUSION

The backtracking controlling algorithm BTrack_Slicer works effectively in the cases where the backtrack prevention mechanisms fail. The unexpected outcomes and errant executions resulting out of indefinite backtracking are reduced by employing the BTrack_Slicer algorithm. Also, the execution time of the program increases rapidly at a faster rate. The utility of the algorithm can be best carried through instances of pattern matching and categorization grounded problems.

REFERENCES

[1] Ivan Bratko, Prolog Programming for Artificial Intelligence, Pearson Education Asia, 3rd edition, 2001
[2] Donglin Liang and Mary Jean Harrold , *"Slicing Objects using System Dependence Graphs,"*International Conference on Software Maintenance,Washington,D.C,pp.358-67,November 1998.
[3] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, Sept. 1995.
[4] N.Sasirekha,A.Edwin Robert and Dr.M.Hemalatha ," Program Slicing Techniques and its Applications ", International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3,pp 50- 64, July 2011