NASA Technical Memorandum 101031

A Technique for Solving Constraint Satisfaction Problems Using Prolog's Definite Clause Grammars

Philip R. Nachtsheim

October 1988

National Aeronautics and Space Administration

NASA Technical Memorandum 101031

A Technique for Solving Constraint Satisfaction Problems Using Prolog's Definite Clause Grammars

Philip R. Nachtsheim, Ames Research Center, Moffett Field, California

October 1988

NASA

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

SUMMARY

This paper presents a new technique for solving constraint satisfaction problems using Prolog's definite clause grammars. It exploits the fact that the grammar rule notation can be viewed as a "state change notation." The novel feature of the technique is that it can perform informed search as well as blind search. It provides the Prolog programmer with a new technique for application to a wide range of design, scheduling, and planning problems.

INTRODUCTION

Constraint satisfaction problems (CSP) require the assignment of values to variables subject to a set of constraints. In logic programming constraint models have wide application in design, scheduling, and planning. This paper describes a technique that solves CSP using Prolog's definite clause grammars, and exploits the fact that the grammar rule notation can be viewed as a "state change notation." As will be shown, such notation facilitates the development of a dynamic representation that can perform informed search as well as blind search. The remainder of the paper illustrates the technique by solving the four queens problem, points out how a priori information is incorporated into the search, and solves a more complex scheduling problem that exploits the full capabilities of the grammar rules. The precise solution is presented in an appendix.

Richard O'Keefe provided valuable advice on many aspects of this paper. I am also indebted to Frank Cady and Don Ferguson for valuable discussions of the material.

ILLUSTRATION OF THE TECHNIQUE

First, the four queens problem illustrates the technique. This problem consists of placing four queens on a square board with four positions on a side subject to the constraints that no queen occupies the same row, column, diagonal, or cross diagonal as any other queen.

The key observation in understanding the technique is that the grammar rule notation of Prolog can be viewed as state change notation. We can understand this key by comparing the four queens problem to the more general problem of parsing an English sentence.

In the four queens problem, the initial state can be viewed as an empty board, and the placement of a single queen subject to the constraints can be viewed as the next state. Each subsequent state entails placement of additional queens subject to additional constraints until the board contains four queens, the final state.

Similarly, parsing a sentence involves a series of transitions from state to state, which we normally think of in such terms as subject, verb, and object. In the general parsing problem as described in reference 1, the goal of parsing a sentence can be expressed in the grammar rule notation using the infix operator "-->" as follows:

```
sentence --> noun_phrase,verb_phrase.
```

When read in, the above grammar rule expands to the following Prolog procedure, which states the initial state is represented as a list, S0, whose elements constitute a sentence:

```
sentence(S0,S) :-
     noun_phrase(S0,S1),
     verb_phrase(S1,S).
```

The elements of the list S0 are then removed according to a set of rules. For example:

```
noun_phrase(S0,S1) is true if
  there is a noun phrase at the beginning of S0
  and the part of the list left after the noun phrase is S1.
```

In the current technique for solving CSP, a predicate, adjoin/3 is introduced (the notation adjoin/3 indicates that adjoin is a predicate with three arguments):

```
adjoin(Goal,Node,Next_Node) is true if
  the Goal is consistent with the set of facts in Node
  and Goal is added to Node to yield Next_Node, provided it is not
    already a member of the set of facts.
```

Let the set of facts in Node be represented as a list. Elements can be added to this list in moving to the next node according to a set of rules, the constraints. In the parsing problem the list which originally contained the sentence is being depleted according to a set of rules. Both cases exploit the grammar rule notation as a state change notation. Note that because we never use the grammar rule terminals, we are not compelled to represent the set of facts as a list; but it is convenient to do so.

The real advantage of this dynamic representation of the nodes in a search tree comes about when an informed search is performed. We discuss this advantage after the technique is illustrated for a blind search in the case of the four queens problem.

For the four queens problem, let the possible positions on the board be denoted by the predicate, pos(Row,Column). The fact that a square is occupied by a queen is indicated by the predicate, queen/2 (e.g., queen(1,3)).

The positions on the board are established by asserting into the data base the following facts:

```
pos(1,1). pos(1,2). pos(1,3). pos(1,4).
pos(2,1). pos(2,2). pos(2,3). pos(2,4).
pos(3,1). pos(3,2). pos(3,3). pos(3,4).
pos(4,1). pos(4,2). pos(4,3). pos(4,4).
```

The adjoin/3 procedure updates the state of the search as represented by the set of facts at a node in the search tree

```
adjoin(queen(Row,Column),Node,Node) :-
        member(queen(Row,Column),Node).

adjoin(queen(Row,Column),Node,[queen(Row,Column)|Node]) :-
        pos(Row,Column),
        not member(queen(Row,Column),Node),
        not inconsistent(queen(Row,Column),Node).
```

The first clause checks whether the goal is already a member of the list Node. In addition, the first clause can also instantiate goals. For example, the call to member/2 may be used to enumerate existing members of Node. (This situation does not occur in this problem, but it does occur in the problem treated in the appendix.) The second clause attempts to add the goal to the list Node to yield Next_Node. This is successful if the ground goal satisfies the constraints, i.e., it is not inconsistent with the current state, as expressed by the contents of the list Node. The second clause fits the general paradigm

```
find :- generate,test.
```

The goal of the program is expressed using the Prolog grammar rule notation which employs the operator "-->".

```
queens -->
     adjoin(queen(1,_)),
     adjoin(queen(2,_)),
     adjoin(queen(3,_)),
     adjoin(queen(4,_)).
```

Note, the above grammar rule expands to the Prolog procedure shown below.

```
queens(Initial_State,Final_State) :-
     adjoin(queen(1,_),Initial_State,Node1),
     adjoin(queen(2,_),Node1,Node2),
     adjoin(queen(3,_),Node2,Node3),
     adjoin(queen(4,_),Node3,Final_State).
```

The comparison with the general problem of parsing a sentence is clear. Queens(Initial_State,Final_State) plays the role of sentence(S0,S1) and adjoin(queen(1,_),Initial_State,Node1) plays the role of noun_phrase(S0,S1). Adjoin has been given an extra argument, namely queen(1,_). The extra argument is necessary because we are building a list. The list grows as the solution proceeds. In the parsing problem the list empties as the solution proceeds.

Now we can understand how the state change notation aspect of the grammar rules facilitates the formation of dynamic data structures which represent the nodes in a search tree. Because Prolog has no nonlocal variables, operationally the variables generated when the grammar rules are expanded can be viewed as placeholders for the current node list and the next node list as the computation proceeds. Initial_State and Node1 are placeholders in the first subgoal above. The expanded subgoal can be regarded as a dynamic data structure linking together the nodes of the search tree in which the current node is con-

nected by the current goal to the next node. This feature enables the technique for solving CSP to work from the bottom up in the sense that what has been accumulated so far on the current node is accessible as the solution proceeds.

For example in the query, queens(Initial_State,Final_State), the list Initial_State will be the input and the list Final_State will be the output. Suppose the Initial_State is the empty list. If the first subgoal, adjoin(queen(1,_),[],Node1), is successful, queen(1,_) would be instantiated to some value, say queen(1,1), and Node1 would become the list [queen(1,1)]. This represents the current state at the next node and an attempt would be made to satisfy the next subgoal. Satisfaction of the final subgoal returns Final_State, the sought-for solution.

The member/2 procedure is used to search the current state

```
member(E,[E|_]).
member(E,[_|R]) :-
member(E,R).
```

The no-attack constraint is met by the following consistency rule which states that a queen in row R, column C, diagonal, and cross diagonal can be attacked by another queen occupying these coordinates.

```
inconsistent(queen(R,C),W) :-
        member(queen(R1,C1),W),
        (R =:= R1 ;                 % same row, or
         C =:= C1 ;                 % same column, or
         R + C =:= R1 + C1 ;        % same diagonal, or
         R - C =:= R1 - C1).        % same cross diagonal
```

The query queens([],Final_State) leads to an answer. Additional answers can be obtained by forcing backtracking.

It is interesting to view the dynamic data base as the computation proceeds. The query above leads to the trace shown below.

```
[]
[queen(1,1)]
[queen(2,3),queen(1,1)]                     % subsequently failed to place
                                            %   a queen in row three
[queen(2,4),queen(1,1)]                     % shallow backtracking
[queen(3,2),queen(2,4),queen(1,1)]          % subsequently failed to place
                                            %   a queen in row four
[queen(1,2)]                                % deep backtracking
[queen(2,4),queen(1,2)]
[queen(3,1),queen(2,4),queen(1,2)]
[queen(4,3),queen(3,1),queen(2,4),queen(1,2)]   % solution
```

In the trace above, the first line indicates that we start with an empty list, and each successive line represents a different list. These lists represent nodes in the search tree. The row number for each queen in the first position of each list is the level of the node in the search tree. As we descend in the search tree, this row number increases.

In the trace, as we go to a new line with the same row number, the system is performing shallow backtracking, i.e., old values are rejected and the search continues at the same level. Shallow backtracking occurs above in the first attempt to place a queen in row 3. As we go to a new line with a lower row number, the system is performing deep backtracking, i.e., old values are rejected and the search continues at a previous level. Deep backtracking occurs above in the first attempt to place a queen in row 4.

Clearly the dynamic data base is exhibiting nonmonotonic behavior in that it grows and shrinks as the computation proceeds. In some cases, what was true earlier in the computation may no longer be true later in the computation. For example, queen(1,1) the first element added to the list, is no longer a member of the list at the end of the computation.

Goals in the bodies of the "inconsistent" clauses are treated differently than goals appearing in the rules stated in the grammar rule notation, so it is necessary to make some remarks about when it is appropriate to use the Prolog grammar rules and when it is appropriate to use ordinary Prolog rules.

The "inconsistent" clauses which represent the constraints should always be ordinary Prolog rules. The current node list is always input to these rules and the subgoals of these rules are satisfied by searching the current node list using member/2.

The relations that express the goal of the program that attempt to "prove" goals may be expressed in the grammar rule notation. Actually, what must be provided are placeholders in the expanded goals to represent the current node and the next node. The grammar rule notation just makes it easier to formulate these goals. The goals appearing in the grammar rules are the ones that will be adjoined to the lists at the nodes in search tree.

INFORMED SEARCH

In view of the fact that this new technique for solving CSP can traverse the search tree and generate the next node starting from a current node, there is no reason why the initial node has to be empty. In principle, the initial node could be anything. For example, suppose the initial node, instead of an empty board, in the four queens problem, is the list [queen(1,3)]. The query queens([queen(1,3)],Final_State) will lead to an answer directly. As another example, suppose the user is only interested in a solution to the four queens problem in which pos(1,1) is occupied. The query queens([queen(1,1), Final_State) will lead to a negative reply directly. These results illustrate the real advantage of the current technique which represents the nodes in a search tree dynamically.

Traditional techniques for solving the N-queens problem, such as reference 2, can obtain the same answers as above, but only indirectly. For the first example, all the terms that represent solutions starting with an empty board are collected; the solutions that do not include queen(1,3) are discarded. For the second example, all the solutions starting with an empty board would have to be obtained to determine there is no solution with pos(1,1) occupied. The current technique, as opposed to traditional techniques permits the user to specify a priori properties that the solution should possess. The ability to initiate the search with arbitrary states may not be important in all applications; however, an application where it may be very important is treated in the appendix.

Introducing information at a node has an interesting interpretation. The original problem starting at an empty node can be considered as a CSP and the problem starting at a node with the given information can be considered as a new CSP with the information regarded as a constraint (see ref. 3).

However, because the user can specify inconsistent information, a procedure is needed to guard against erroneous input. The procedure consistent/1, which takes a list of goals as an argument performs the task of determining that the user-supplied goals are consistent.

```
consistent([queen(Row,Column)|Rest_of_Queens]):-
        pos(Row,Column),
        not inconsistent(queen(Row,Column),Rest_of_Queens),
        consistent(Rest_of_Queens).
consistent([]).
```

In the example above, the use of consistent/1 is illustrated by the compound query below

```
consistent([queen(1,3)]),queens([queen(1,3)],Final_State).
```

Alternately, consistent/1 can be used as a simple query to check the validity of a complete solution.

CONCLUSIONS

The new technique for solving CSP provides the Prolog programmer with a new technique for application to a wide range of problems. It should be especially useful for those applications where it is desirable to perform an informed search or for checking a possible solution. The definite clause grammars formalism greatly facilitated the implementation of this technique. Because definite clause grammars are widely available in Prolog systems, the new technique should likewise be widely available.

APPENDIX

This appendix is a formulation of the "The Case of the MSAI Program" which appeared in reference 4 using the current technique.

This is a realistic scheduling problem and is an interesting application of the new technique. The goal of the program is to design a schedule of courses for a student in a 1-year college program leading to a master's degree in artificial intelligence. A student must take three courses in each of three quarters. Five courses are required, and some courses have prerequisites.

An interesting feature of this problem is that the goals are coupled. This feature was not present in the four queens problem. Satisfaction of the requirements goal may subsume a full-quarter goal. Another interesting feature of this problem is that its solution requires an application of the grammar rule notation that the four queens problem did not require.

Because one of the main purposes of discussing this problem is to illustrate the application of the new technique to a realistic problem and not to explain the new technique, its formulation is relegated to an appendix. Remarks concerning the solution are to be found after the problem is formulated.

Predicates

msai(P) :- student P satisfies the msai requirements
reqts(P) :- student P's schedule includes all required courses
full(P) :- the schedule is full for student P
fullq(Q,P) :- the schedule is full for quarter Q for student P
course(Q,P,C) :- course C is taken in quarter Q by student P
offered(C,Q) :- the course C is offered in quarter Q
prereq(C,D) :- course C is a prerequisite for course D

Catalog of courses

```
offered(cs206,f).
offered(cs204,f).
offered(cs161,f).
offered(cs156,f).
offered(cs142,f).
offered(cs102,f).

offered(cs226,w).
offered(cs223,w).
offered(cs145,w).
offered(cs143,w).

offered(cs225,s).
offered(cs224,s).
offered(cs222,s).
offered(cs142,s).
offered(cs102,s).

before(f,w).
before(f,s).
before(w,s).

prereq(cs223,cs222).
prereq(cs102,cs224).
prereq(cs142,cs145).
```

The above data base is required for the MSAI Program shown below.

The adjoin procedure updates the state of the search as represented by the set of facts at a node in the search tree

```
adjoin(course(Q,P,C),Node,Node) :-
  member(course(Q,P,C),Node).
```

```
adjoin(course(Q,P,C),Node,[course(Q,P,C)|Node]) :-
 offered(C,Q),
 not member(course(Q,P,C),Node),
 not inconsistent(course(Q,P,C),Node).
```

Goals

```
msai(P) -->
  reqts(P),
  full(P).

reqts(P) -->
  adjoin(course(Q1,P,cs223)),
  adjoin(course(Q2,P,cs222)),
  adjoin(course(Q3,P,cs156)),
  adjoin(course(Q4,P,cs142)),
  adjoin(course(Q5,P,cs161)).

full(P) -->
  fullq(f,P),
  fullq(w,P),
  fullq(s,P).

fullq(Q,P) -->
  adjoin(course(Q,P,C1)),
  adjoin(course(Q,P,C2)), {C1 @> C2},
  adjoin(course(Q,P,C3)), {C2 @> C3}.
```

Constraints

```
inconsistent(course(_,P,C),W) :-    % no repetition of courses
    member(course(_,P,C),W).

inconsistent(course(Q,P,C),W) :-    % no more than three courses per quarter
    bagof(D, member(course(Q,P,D),W), Course_Load),
    length(Course_Load,3).

inconsistent(course(Q2,P,D),W) :- % do not schedule follow on course unless
    prereq(C,D),                  % prerequisite is scheduled
    not member(course(Q1,P,C),W).

inconsistent(course(Q2,P,D),W) :- % do not schedule follow on course
    prereq(C,D),                  % before prerequisite is scheduled
    member(course(Q1,P,C),W),
    (before(Q2,Q1) ; Q2==Q1).
```

Member procedure

```
member(E,[E|_]).
member(E,[_|R]) :-
    member(E,R).
```

Remarks on the program follow:

1. The query msai(prn,[],Final_State) leads to an answer, the program assigns a list of courses to the variable Final_State. Additional answers can be obtained by forcing backtracking.

2. A priori information can be incorporated by invoking, for example, the goal msai(prn,[course(s,prn,cs142)], Final_State). As in the case of the four queens problem, a separate procedure could be written to guard against erroneous input by the user.

3. The operator "@>" did not appear in reference 4. It is used herein merely for convenience to avoid redundant solutions.

4. The curly bracket notation is used in the clause fullq(Q,P) to maintain the integrity of the list of facts at each node.

5. It should be stressed that the first clause in the adjoin/3 procedure is supposed to be able to instantiate goals, not just to test whether an already-existing goal is a member of the list Node. In order to illustrate this, one of the solutions to the query msai(prn,[],Final_State) is presented below. Incidentally, there are nine nonredundant solutions to this query.

```
Final_State =
[course(s,prn,cs102),course(s,prn,cs225),course(w,prn,cs143),
 course(w,prn,cs145),course(f,prn,cs161),course(f,prn,cs142),
 course(f,prn,cs156),course(s,prn,cs222),course(w,prn,cs223)]
```

In this list the goals are in reverse order to the order in which they were obtained. The last five courses of the list satisfied the first subgoal of the program, reqts(prn). Note, three of these courses are in the fall quarter. The next subgoal of the program, fullq(f,prn) is satisfied by repeated application of the first clause in the adjoin/3 procedure. The calls to member/2 enumerated the existing members of the list Node for the fall quarter.

6. Finally, this program provides an opportunity to point out an application when this technique for solving CSP would be better than traditional approaches. Suppose there were 10 courses offered in the fall quarter instead of 6, but a new student knew exactly which 3 courses he or she would take in the fall quarter. Clearly, there would be an advantage in obtaining a schedule directly by performing an informed search by inserting the desired courses a priori into the schedule as opposed to obtaining all potential schedules by traditional approaches and then discarding those schedules that did not contain the desired courses.

REFERENCES

1. Clocksin, W. F.; and Mellish, C. S.: Programming in Prolog, Third Edition, Springer-Verlag, New York, N.Y., 1987.

2. Sterling, L.; and Shapiro, E.: The Art of Prolog, The MIT Press, Cambridge, Mass. 1986.

3. Finger, J. J.: Exploiting Constraints in Design Synthesis, Doctoral Dissertation, Stanford University, Stanford, CA, 1987.

4. Genesereth, M. R.: The MRS Casebook, HPP-83-26, Stanford University Heuristic Programming Project, Stanford, CA, 1983.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-101031		

4. Title and Subtitle	5. Report Date
A Technique for Solving Constraint Satisfaction Problems Using Prolog's Definite Clause Grammars	October 1988
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Philip R. Nachtsheim	A-88286
	10. Work Unit No. 506-66-11

9. Performing Organization Name and Address	11. Contract or Grant No.
Ames Research Center Moffett Field, CA 94035	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, D.C. 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

Point of contact: Philip R. Nachtsheim, Ames Research Center, MS 244-17, Moffett Field, CA 94035
(415) 694-4387 or FTS 464-4387

16. Abstract

This paper presents a new technique for solving constraint satisfaction problems using Prolog's definite clause grammars. It exploits the fact that the grammar rule notation can be viewed as a "state change notation." The novel feature of the technique is that it can perform informed search as well as blind search. It provides the Prolog programmer with a new technique for application to a wide range of design, scheduling, and planning problems.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Constraint satisfaction Prolog Search	Unlimited–Unclassified Subject category: 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	12	A02

For sale by the National Technical Information Service, Springfield, Virginia 22161