# A Predicate Connection Graph Based Logic With Flexible Control

Richard Whitney, Darrel J. VanBuer, Donald P. McKay,
Dan Kogan, Lynette Hirschman, Rebecca Davis

System Development Corporation
Santa Monica, California and Paoli, Pennsylvania

## Abstract

The FDE has been designed to support multiple search strategies for logic programs. This machine represents the knowledge base in a strategy independent fashion as a predicate connection graph which encodes potential unifications between predicates. It facilitates knowledge representation in the language of full first order predicate calculus Immediate developments include implementation of various database access strategies and addition of evaluable predicates and functions to the language. Long-term research will focus on exploration of search strategies, especially for parallel logic machines.

## I. Introduction

The *Flexible Deductive Engine* (FDE), is designed to serve as the deductive core for both logic programming environments and knowledge management systems involving database access. The intent is to create a single module which supports the full functionality of logic programming, as well as alternate forms of deduction appropriate to a knowledge management system [Kellogg, 19821. Support of logic programming applications and deductive querying of databases in a single environment requires flexibility in search strategy through control customized to the application. Such flexibility results in a highly modular search engine, operating on a strategy-independent representation of the search space.

One goal of the FDE is to provide an experimental framework where the issues of control of inference, database access, and function evaluation are treated as part of a general problem of search control. For this reason, we have designed the core of the FDE with a set of well-defined interfaces; the person developing a logic system then specifies the search strategy by choosing a set of control functions, which can be augmented as needed to fit the application. The current repertoire of strategies includes at one extreme Prolog-style depth-first left to-right search and at another Loglisp-style breadth-first search with cost functions for limiting depth-first expansions. Other strategies may combine depth-first and breadth-first expansion at different points in the deduction cycle depending on heuristic choice functions.

The FDE draws upon previous research carried out at SDC over the past several years in the general area of Knowledge Management and specifically in DADM, the Deductively Augmented Data Management project [Kellogg & Travis, 1981]. The FDE preserves the general Knowledge Management architecture underlying DADM I Kellogg, 1982; Kogan, 1984] through decomposition of computational functions into an inference engine, with its associated intensional set of rules, and a search engine, with its associated external extensional DBMSs. This division of relations into those with intensional support and those with extensional support has been a central feature of the DADM architecture [cf. Klahr, 19781 as well as other logic-based systems attempting to deal with external DBMSs [Chang, 1981; Henschen & Naqvi, 19821. The intensional rules are encoded in a predicate connection graph (PCG) which records both the possible resolvents for each rule and the relations which have extensional database support. Our implementation of the PCG is the main topic of this report.

The FDE provides:

—uniform interface to a collection of diverse relational DBMSs with different query languages, data presentation strategies, and functionalities;

—alternative views to the data in extensional DBs by hiding irrelevant fields or defining new relations derived from the stored data via shallow deductions;

—an interface to the extensional DB allowing external relations to act as ground clauses to the logic programming environment.

The core of the FDE consists of four main parts (figure 1

(1) The knowledge base contains a rule base (the collection of procedures of a logic program or the set of definitions of virtual relations for a high-order query language) and a collection of database facts maintained in multiple relational DBs external to the deductive core The predicate connection graph encodes the *rule base* in the deductive core and also identifies the points of contact to the relations represented in the database components. To support the relational DB component of the knowledge base, the FDE maintains its own internal relational DBMS as well as communication links to external intelligent DB systems. As much as possible, the specialized functions of the external DBMSs will be exploited by the deductive core.

(2) The unification engine, central to any logic-based system, supports multiple parallel breadth-first search processes in the FDE. It is based *on* the unification algorithm *of* Loglisp [Robinson & Sibert, 1982], extended to allow for multiple simultaneous contexts, i.e. multiple collections of variable bindings in force for a deduction Each search state has its own associated context to represent the variable bindings in effect in that state [McKay & Travis, 1984]

(3) The search engine explores the search space by constructing an AND/OR tree from the PCG representation of the rule base. A list of search states keeps track of the different search paths concurrently under investigation The search engine proceeds through a deduction cycle which performs reduction on a particular search state. If this search state represents a solution, the deduction cycle returns it and its continuation; otherwise, the deduction cycle continues exploring the search space.

(4) The search control strategy is invoked by the search engine to control its deduction cycle through a suite of functions which stipulate how to choose and prune states from the search space. It is this set of functions which allows the control strategy to vary with the application.

The following sections describe our implementation of the predicate connection graph and its representation of first order predicate calculus. The FDE is implemented in Interlisp-Don Xerox 1100-series Lisp Machines.

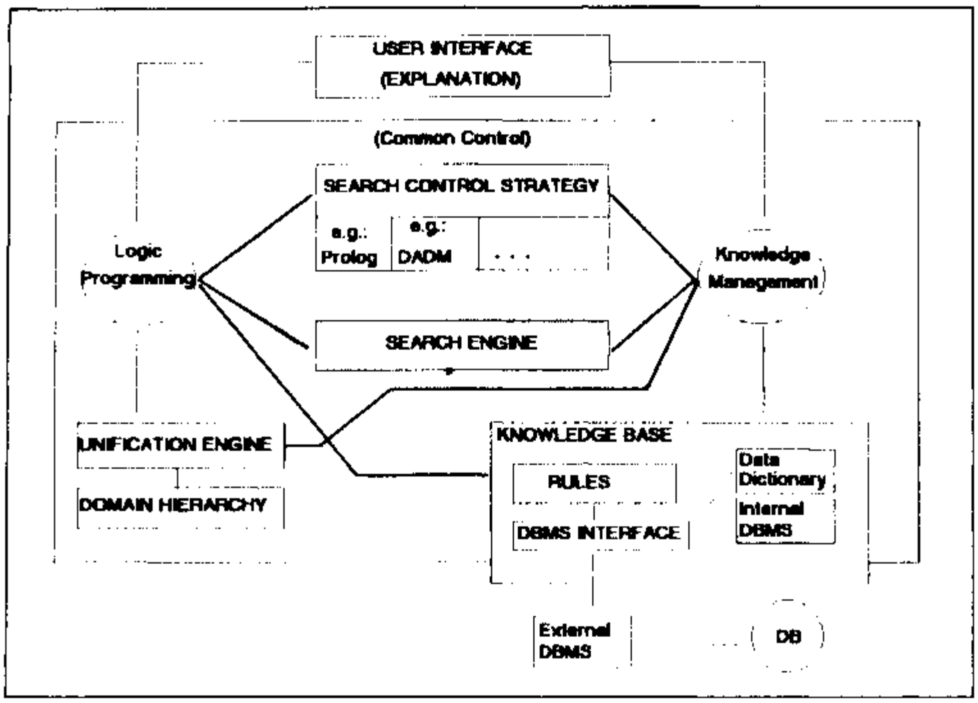


Figure 1. Organization of the FDE

## II. The predicate connection graph in the FDE

The purpose of any PCG is to encode the potential unifications between the predicates (literals) as used in rules [Kowalski, 1975; Sickel, 1976]. As the internal specification in the FDE of the logic program, it represents an extended form of indexing The rules submitted to the FDE are normalized and encoded in the PCG; growth of a proof tree through the deductive cycle is determined by the deductive interactions contained in the PCG

We chose a predicate connection graph to represent the rules of a logic program primarily because such a formulation lends itself to the central design consideration of flexibility in search strategy. Also, we have substantial prior experience with such structures in database question answering; the present implementation of the predicate connection graph form of rule representation is a refinement of the PCG of DADM.

Our PCG facilitates a language with the expressive power of first order predicate calculus Its inner representation of a rule set is a Skolem normal form of conjuncts which in turn comprise disjunctions of literals or negations of literals Simple syntactic transformations reduce a given formula to an equivalent set of normalized PCG entries. The availablility of full first order predicate calculus allows convenient expression of complicated rules and submission of queries involving hypothetical or counterfactual antecedents. Moreover, divorcing the language from a limited canonical form is important when a range of search strategies is available: Horn clauses may be appropriate for Prolog's depth-first left to right strategy, but such forms should not prejudice search control in the general case.

One example of the kind of problem that the FDE simplifies is a general treatment of negation. As a database question answering system, the FDE allows for both *open* and *closed* worlds [Reiter, 19781. Domains where the open world assumption is in effect require that we express and use negated literals in our rule base, since the Prolog technique of negation as failure is inadequate and logically incorrect in such cases. But even in closed worlds, where negation as failure yields desired results, a more robust language provides additional benefits. Consider the definition *of bachelor as unmarried male.* A straightforward Prolog interpretation of this definition would be:

bachelor(X):-not(married(X)),male(X).

With ground clauses *marriedlb)* and *male(a),* however, the particular goal *bacheloria)*[2] will succeed while the general goal *bacheloriX)?* will fail, due to the combination of Prolog's search strategy and the handling of negation as failure In order to compute the relative complement of a database relation, the FDE will first determine which individuals satisfy the positive relations (in this case, which are males). Then it will determine which individuals satisfy the negated relations in a positive sense (i.e. which are married). The relative complement of the first set with respect to the second (members of the set of males which are not members of the married set) then determines the answer. The delayed negation is a feature of the database operations, unlike the more general solution of MU-Prolog [Naish, 1983], where it is a function of search control.

The solution in Prolog is to reorder the defining literals:

bachelor(X):-male(X),not(married(X)).

Now, with the ground clause *male(a),* the goal *bacheloriX)?* will succeed for individual *a* However, asserting *bachelor(a)* does not allow any conclusion to be drawn about a's marital status. In the FDE, we can express the definition in biconditional form,

$$\mathbf{bachelor(X) \Leftrightarrow male(X) \land not(married(X))}$$

and infer, from the assertion *bacheloiia),* the conclusion *not(married(a)).* And this is achieved within a depth-first left-to-right search strategy simply through the flexibility of the language provided by rule and query encoding in the PCG.

## A. The rationale of the PCG

The PCG is composed of two kinds of abstract objects: relations and occurrences. Relations correspond to symbols used as the name of a relation, e.g. *P* in *P(a).* Occurrences represent instances of literals in rules; they also represent instances of literals in queries. Since a single relation may occur many times in a collection of rules, we distinguish the properties pertinent to the relation itself from those pertinent to the occurrences of it

Relations are objects with two properties, namely, ordered sets of the uses of this relation in literals in (1) a positive sense and (2) a negative sense. The ordering reflects the order in which rules are made known to the system, to preserve the sequential nature needed for Prolog-style programming For some alternate modes of deduction, it will be possible to express the measure of confidence in each rule by means of a plausibility factor This factor can then be used to control deduction by ordering the rule set or the uses of rules according to the strength of the plausibility factors.

Literal occurrences are objects with a number of properties. These properties are:

SIGN: whether usage is (P - ) or (NOW --)).

LITERAL: the literal exclusive of any negation, e.g. (P --),

ORMATES: links of this occurrence to other parts of the rule containing this occurrence,

UNIFIERS: links to occurrences which represent potential resolvents when this occurrence is a goal

The internal representation of a rule set as a conjunctive normal form expects each rule to be convertible to a disjunction of literals and negated literals. Pure Horn clause rules, which consist of a conjunction of literals as antecedent and a single literal as conclusion, meet this expectation by using the transformation:

$$A_1\wedge\ldots\wedge A_n \rightarrow B \leftrightarrow \neg(A_1\wedge\ldots\wedge A_n)\vee B$$

De Morgan's law then converts the conjunctive antecedent to a disjunction:

$$\neg A_1\vee\ldots\vee\neg A_n\vee B$$

Other logical forms are transformed by the generation of new, unique literals to replace complex forms in the disjunction and the creation of new rules connecting the new literals with the forms replaced. Such a situation is encountered with the PCG representation of the rule defining bachelorhood previously mentioned (figure 2):

$$\text{bachelor(X)} \leftrightarrow \text{male(X)} \wedge \text{not(married(X))}$$

In the *only-if* («-) direction of the biconditional, the Horn clause equivalent of the Prolog definition is simply transformed as represented in the upper branch of the tree from BACHELOR (figure 2). Distribution of negation by De Morgan's law causes the reversal of sign in the components of the original conjunct, so the equivalent disjunctive form would read: x is a bachelor *or* x is not male *or* x is married.

In the *only-if* (<■) direction of the biconditional, the Horn clause equivalent of the Prolog definition is simply transformed as represented in the upper branch of the tree from BACHELOR (figure 2). Distribution of negation by De Morgan's law causes the reversal of sign in the components of the original conjunct, so the equivalent disjunctive form would read: x is a bachelor *or* x is not male *or* x is married.

In the case of the if (■>) direction of the biconditional, a new literal, *not(GPred0(X)),* is generated to represent the conjunction in the consequent of the rule, so we first rewrite the original as two rules:

bachelor(X) -> not(GPred()(X))

and

not(GPred0(X)) -> male(X) A not (married(X))

(Negation of generated literals is introduced to minimize negated terms in the ultimate normal forms ) The first of these is equivalent to the normal form:

$$\text{not(bachelor(X))} \vee \text{not(GPred0(X))}$$

The second is rewritten into two more rules,

$$\text{not(GPred0(X))} \Rightarrow \text{male(X)}$$

and

$$\text{not(GPred0(X))} \Rightarrow \text{not(married(X))}$$

which find equivalent normal forms in

GPredO(X) v male(X)

and

GPredO(X) V not(married(X)).

This situation can now be read off from the lower branch of the tree from BACHELOR. (Since the UNIFIERS field of a given occurrence represents potential unifications for the purpose of resolution, there is a difference of sign between occurrences so connected).

Queries are also treated as occurrences of relations which are connected to the PCG. Because the FDE uses resolution as its main technique and resolution is a form of proof by contradiction, the negation of a query is actually entered into the PCG For queries, resolution is done primarily by backchaining, and exclusively so when emulating Prolog, so unifications for queries are actually connected in only one direction (from query occurrence to PCG occurrence) instead in the bidirectional form of the PCG proper The main reason for using the unidirectional links is robustness of the implementation. Because there are no links from the PCG to the query, abnormal termination of query processing *never* leaves any unwanted connections. Another reason for avoiding reverse links is to avoid begging the question, since queries are not rules, so they should not be backchaining resolvents

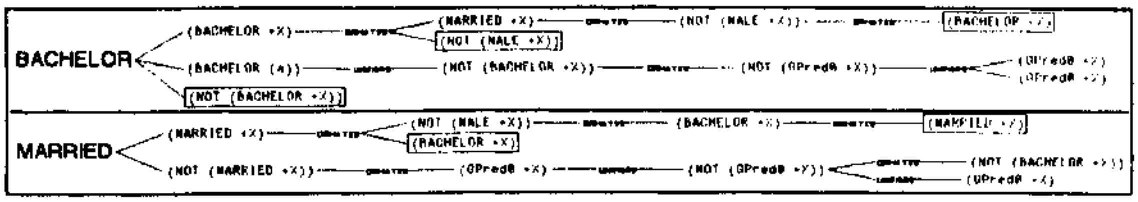


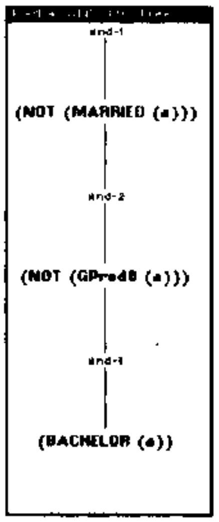
Figure 2. The PCG from **BACHELOR** and **MARRIED**

## B. The PCG and the deduction cycle

The search engine of the FDE uses an AND/OR tree to represent progress through the knowledge base. Growth of this tree is governed by the suite of functions which encode the control strategy The tree contains alternating levels of AND nodes and OR nodes. An OR node represents a literal to be resolved. Each OR node contains a list of alternate potential unifiers obtained from the PCG, the AND-node children of the OR node represent the resolvents of the goal literal obtained from the alternates

Submitting a query causes it to be conjoined to the PCG and a root AND-node initialized with its children OR nodes. Resolution of an OR-node causes a new branch to be added to the tree if the alternative unified with has ORMATES. In that case, the resolved OR acquires a child AND-node representing each alternative. If there are no ORMATES, then the OR node is solved

A trace of the deduction cycle and the AND/OR tree for the sample query *NOT(MARRIED( <—X))<* are shown in figure 3 and illustrate some of the details of this process, and can be compared with the PCG structure of figure 2 The goal under And 1 in figure 3 corresponds to the PCG node (NOT(MARRIEl) <-X)) connected to the root of the PCG display for MARRIED (fig 2), and is the first chosen goal node. The immediate subgoal is found by following the ORMATES arc to the single goal (GPredO <-X) The goal under And-2 is one of the UNIFIERS (in this case, the only) of this node, (NOTfGPredO *-X)). From here, we again follow the ORMATES arc of (NOT (GPredO <-X)) to its immediate subgoal (NOT(BACHELOR «-X)). The goal under And 3 (fig. 3) is the only unifier of the node (BACHELOR a) Since this node has no ORMATES, it is resolved. This represents deducing NOT(MARRIED(a)) from BACHELOR(a)



```
-> NOT(MARRIED(←X))?

CHOSEN GOAL NODE:
NOT(MARRIED ←X)
CHOSEN ALTERNATIVE:
NOT(MARRIED ←X)<-NOT(GPred0 ←X)
** SUCCESS **

  CHOSEN GOAL NODE:
  NOT(GPred0 ←X)
  CHOSEN ALTERNATIVE:
  NOT(GPred0 ←X)<-(BACHELOR ←X)
  ** SUCCESS **

    CHOSEN GOAL NODE:
    (BACHELOR ←X)
    CHOSEN ALTERNATIVE:
    (BACHELOR a)<-
    ** SUCCESS **

---------------------------
            BACHELOR(a)
        NOT(GPred0(a))
NOT(MARRIED(a))
```

Figure 3. The AND/OR tree and trace of the deduction cycle *for NOT(MARRIED(*-X))?*

## III. Work in progress

One of the primary targets of the current development is an effective solution to problems *of* aggregation and the related problems associated with the *setof* and *bagof* predicates of Prolog, as well as a general treatment of evaluable predicates and functions. One case concerns the problem of determining a value for, e.g., the *number* of bachelors in our database. In line with the overall adaptability of the FDE, we intend to allow different strategies for different predicates: a Prolog style for computing this number would abort the inference procedure if an evaluable predicate were called with not all of its arguments instantiated. The DA DM strategy would be to check the status of the arguments routinely during the course of deduction until all are instantiated, and then perform the computation.

We are also developing a more general treatment of database access than exists in DADM. The FDE will allow for a relation to be defined in both the intensional rule base and in the extensional database. A special choice function will control the point at which database access is performed: eagerly, as in Prolog, or in batched, deferred mode as in the current DADM

## IV. Bibliography

Chang, C L., "On evaluation of queries containing derived relations in a relational data base " *Advances in Data Base Theory, vol. 1,* H. Gallaire, J. Minker, and J Nicolas (eds.), Plenum, New York, 1981

Henschen, L. J. and Naqvi, S., "Representing infinite sequences of resolvents in recursive first order horn databases " *6th Conference on Automated Deduction,* G. Goos and J Hartmanis (eds), New York, 1982.

Kellogg, C. and Travis, L., "Reasoning with Data in a Deductively Augmented Data Management System " *Advances in Data Base Theory, vol. 1,* H. Gallaire, J Minker and J. M Nicholas (eds.), Plenum, New York, 1981

Kellogg, C, "Knowledge Management: a Practical Amalgam of Knowledge and Data Base Technology." *Proceedings of the Second National Conference on Artificial Intelligence,* 1982

Klahr, P., "Planning Techniques for Rule Selection in Deductive Question-Answering," *Pattern-Directed Inference Systems,* D A Waterman and F. Hayes-Roth (eds.), Academic Press, New York, 1978.

Kogan, D., "The Manager's Assistant: an Application of Knowledge Mangement." In *IEEE International Conference on Data Engineering Proceedings,* Los Angeles, April 1984.

Kowalski, R , "A Proof Procedure Using Connection Graphs," *J ACM,* 22:4, October, 1975, pp. 572-595

McKay, DP. and Travis, L., *Unification Engine,* LBS Technical Memo, System Development Corporation, Paoli, PA, April, 1984,

Naish, L., *An Introduction to MU-Prolog,* Technical Report 82/2, Department of Computer Science, University of Melbourne, January, 1983.

Reiter, R., "Deductive question answering on relational data bases." *Logic and Data Bases,* H. Gallaire and J. Minker (eds.), 1978.

Robinson, J. A. and Sibert, E. E., "LOGLISP: Motivation, design and implementation." *Logic Programming,* K.L. Clark and S-A Tarnlund (eds), Academic Press, New York, 1982.

Sickel, S., "A Search Technique for Clause Interconnectivity Graphs." *IEEE Transactions on Computers* C-25:8, August, 1976.