COOPERATIVE ANSWERS IN DATABASE SYSTEMS *

Terry Gaasterland, Parke Godfrey, Jack Minker, and Lev Novik
Department of Computer Science
A.V. Williams Building
University of Maryland
College Park, MD 20742

1 Introduction

A major concern of researchers who seek to improve human-computer communication involves how to move beyond literal interpretations of queries to a level of responsiveness that takes the user's misconceptions, expectations, desires, and interests into consideration. At Maryland, we are investigating how to better meet a user's needs within the framework of the cooperative answering system of Gal and Minker [25]. We have been exploring how to use semantic information about the database to formulate coherent and informative answers. The work has two main thrusts: 1) the construction of a logic formula which embodies the content of a cooperative answer; and 2) the presentation of the logic formula to the user in a natural language form. The information that is available in a deductive database system for building cooperative answers includes integrity constraints, user constraints, the search tree for answers to the query, and false presuppositions that are present in the query. The basic cooperative answering theory of Gal and Minker [15, 25] forms the foundation of a cooperative answering system that integrates the new construction and presentation methods.

This paper provides an overview of the cooperative answering strategies used in the CARMIN cooperative answering system, an ongoing research effort at Maryland. Section 2 gives some useful background definitions. Section 3 describes techniques for collecting cooperative logical formulae. Section 4 discusses which natural language generation techniques are useful for presenting the logic formula in natural language text. Section 5 presents a diagram of the system.

2 Some Definitions

Deductive databases are comprised of syntactic information and semantic information. The syntactic information consists of the intensional database (IDB) which is the set of clauses of the form $A \leftarrow B_1, \ldots, B_n$, $n > 0$, where A and each B_i is an atom, and the extensional database (EDB) which is the set of clauses of the form $A \leftarrow$. IDB clauses are also called rules. EDB clauses are also called facts.

*This paper describes research done at the Computer Science Department at the University of Maryland under Air Force Office of Scientific Research grant AFOSR-91-0350 and NSF grant IRI-89-16059.

Direct answers to database queries, which are clauses of the form $\leftarrow B_1, \ldots, B_n$ are found by using SLD-resolution on the query, IDB, and EDB clause to produce a search tree. The root node of the search tree is the query clause; each node in the tree is produced by applying an IDB rule to the node above. A successful leaf node is one which matches with EDB facts; a failed leaf node is one which has no match in the EDB [16, 20].

The semantic information about the database consists of a set of integrity constraints (IC). Semantic information about users of the database consists of a set of user constraints (UC). The constraints considered in this paper have the form $\leftarrow C_1, \ldots, C_n, E_1, \ldots, E_m$, where each C_i is an atom whose predicate appears in an EDB fact or the head of an IDB rule and each E_i is an evaluable expression.

An integrity constraint restricts the states that a database can take. For example, the integrity constraint, *"No person can be both male and female,"* $\leftarrow$ *person(X),male(X),female(X)*, restricts people in a database from having both properties. Because the constraints on a database do not enable the deduction of new answers but rather give information about existing knowledge and answers, they are considered semantic information rather than syntactic information.

A user constraint reflects restrictions that the user places on the database. For example, a user who detests Kennedy airport may impose a restriction on a database about traveling that he does not want to travel into JFK at any point in a trip: $\leftarrow$ *travel(X,JFK)*. In practice, user constraints are labeled as applicable to some particular user.

3 Constructing the Logic Formula

The cooperative answering system of Gal and Minker [25], the precursor to the CARMIN system, uses two basic strategies to identify extra information which can be included in an answer: 1) it checks whether a query is restricted by or violates any of the database's integrity constraints; and, 2) for failed queries, it searches for any false presuppositions in the query. Integrity constraints can help identify a user's misconceptions about the database that are made evident by the user's query. When an integrity constraint interacts with a query and shows that part of its search space must fail, the interaction indicates that the user does not know about some of the information embodied in the constraint. A description of the constraint to the user can be used to correct such misconceptions.

Some queries can have successful answers only if the database is in a state which conflicts with some integrity constraint. Since the database will never take such a state, the query will never succeed. The failure of such a query can be explained through an integrity constraint with which it conflicts. Such a constraint represents a misconception that the user had about the database. For example, if a query asks for someone who is both a mother and a father, that is, *"who is a parent and female and a parent and male?"*, it conflicts with the constraint that no one can be both male and female. Integrity constraints are also useful for identifying queries that contain redundancies, like *"who are all the males who are not female?"*. A response which includes paraphrases of the involved constraints can help correct a user's misconceptions about the database.

The presuppositions in a query are statements that must be true for either the query or its negation to be true [19]. For example, *"list the mothers in the database."* presupposes that mothers exist in the database. A query with a false presupposition has no answer, positive or negative, and a cooperative response would identify the false presupposition to the user.

3.1 Using User Constraints

In an extension to the cooperative answering system, user constraints are used to take into account a user's restrictions on the world when answering questions. A user's intentions and needs can be modeled as a set of constraints. User constraints reflect the semantics that a particular user imposes on the database. They are provided by individual users and need not be consistent with the database. When answering queries posed by a user, they are used to modify a user's query so that the search space of the original query is limited to find only answers that are amenable to the user's needs. The semantic compilation method of Chakravarthy et al. [3] which Gal and Minker use to incorporate integrity constraints into a query can also be applied to user constraints.

Consider a query to a database *"Which airline can I use to travel from Washington DC to Paris' Charles de Gaulle airport?"*:

Q: $\leftarrow$ *travel(Airline,Airport,CDG,Time)*$\wedge$ *near(Airport,Washington).*

and a user constraint *"I refuse to travel through JFK"*:

U: *Loc1*$\neq$ *JFK* $\leftarrow$ *flight(Airline,Number,Loc1,Loc2,Time).*

With database information that includes facts about flights from Washington National to JFK, JFK to CDG, and Washington Dulles to CDG, the query can be answer with *"Take PanAm to JFK and then to CDG."* However, such an answer is useless to the user - it violates the user constraint. An answer which satisfies the user's constraint provides the alternative which does not go through JFK, *"Take AirFrance from Dulles to CDG."*

When a set of answers has been restricted by a user constraint, it would be judicious to assume that the user knows about the user constraint – after all, the user supplied it. So, whereas selected integrity constraints are presented to the user whenever they apply to the query, user constraints are presented more sparingly.

3.2 The Search Tree

The search tree for the set of direct answers to a query contains abundant information both about the answers that are derived for the query and also about the failure paths. Each branch of the tree ends either in failure or success. Some of the failure branches may be labeled by integrity constraints; some may be labelled by user constraints; and some, by both. Some branches may be labelled by false presuppositions.

The cooperative answering system collects the intensional portion of the proof tree for a query as the IDB rules are applied to the query. This intensional proof tree, which we call the IDB-tree, includes all information necessary to connect the original query with any EDB level query literals which conflict with a constraint. It also includes extra information. We have developed an algorithm for gleaning from the IDB-tree the minimal information for making the connection. We then add the IDB-tree information to the cooperative response.

To illustrate the need for search tree information, consider an example about flights that serve meals. The following IDB rule defines the relation *meal_flight* for *breakfast*:

I: *meal_flight*(A,N,*breakfast*) $\leftarrow$

flight_times(A,N,T1,T2),
T1 <09:00, T2>10:00,
¬ *service_type*(A,N,*express*).

The rule says that a flight serves breakfast if the departure time is before 9:00am and the arrival time is after 10:00am and if the flight is not an express flight. Suppose the user poses a constraint that s/he does not want to take flights longer than one hour on United:

U: ← *flight_times*(A,N,T1,T2),A=*united*, Diff is T1 - T2, Diff > 01:00.

Suppose the user then asks a query about which flights serve breakfast between DC National and Chicago O'Hare:

Q: ← *meal_flight*(A,N,*breakfast*),*flight_airports*(A,N,*dcu*,*ohare*).

The original user constraint and the query have no common predicates, thus, the constraint does not affect the query directly. However, when the IDB rule is applied to derive the following subquery Q′, the constraint U partially subsumes Q′:

Q′: ← *flight_times*(A,N,T1,T2), T1 <09:00, T2>10:00,
flight_airports(A,N,*dcu*,*ohare*).

The residue, { ← A=*united* }, restricts the query variable A from taking the value *united*. Thus, all answers to the user will leave out flights on United Airlines.

Since the interaction between the constraint and the query occurred within the search tree, the user may not be aware of the interaction, and the user constraint should be included in the answer to the query. However, an answer that consists only of answer substitutions for the original query and the user constraint may be confusing to the user. The user must infer the connection between the constraint and the original query. For example, consider the following logic response, in which the components are labelled A for answer substitution and U for user constraint information:

A(*meal_flight*(*american*,101,*breakfast*) ∧ *flight_airports*(*american*,101,*dcu*,*ohare*)) ∧
U(← *flight_times*(A,N,T1,T2),
Diff is T1 - T2, Diff > 01:00,
A=*united*).

The response can be paraphrased as: *"American Airlines flight 101 flies from DC National to Chicago O'Hare and serves breakfast. You do not want to know about United flights that are longer than an hour."* The user must infer that a breakfast flight is longer than an hour. For a user who is not aware of this information, a better answer would include the following elaboration:

meal_flight(A,N,*breakfast*) ← *flight_times*(A,N,T1,T2), T1 <09:00, T2>10:00.

This may be paraphrased as *"a flight serves breakfast only if the flight starts before 9:00am and ends after 10:00am."* Notice that the elaboration does not contain the literal in the rule I about *ser-*

vice_type. The algorithm for selecting search tree information described in [10] adds the elaboration to the logical response.

3.3 Relaxation

As noted by many researchers, including [1, 4, 6, 18, 21, 27, 28, 30], an alternative form of cooperative behavior involves providing associated information which is relevant to a query. Generalizing a query in order to capture neighboring information is one means to obtain possibly relevant information.

Gaasterland, Godfrey and Minker have defined a method to *relax* a query in order to find neighboring information [12]. A query can be relaxed in at least three ways: 1) rewriting a predicate with a more general predicate; 2) rewriting a constant (term) with a more general constant (term); and 3) breaking a join dependency across literals in the query. The first two relaxations are achieved in a general manner using *taxonomy clauses* which define hierarchical type relationships between predicates and constants in the database language. For example, the following clauses define relationships between the predicates *communicate*, *call*, *mail*, and *say*:

T1: *communicate*(P_1, P_2, *Msg*) $\leftarrow$ *say*(P_1, P_2, *Msg*).
T2: *communicate*(P_1, P_2, *Msg*) $\leftarrow$ *lives_at*(*Addr*, P_2), *mail*(p_1, *Addr*, *Msg*).

A query containing a request to mail an invitation to the address *"34 Cherry Lane,"* upon failing, can be relaxed to produce alternative queries:

Q: $\leftarrow$ *mail(Terry, "34 Cherry Lane", invitation).*
Relaxed Q: $\leftarrow$ *communicate(Terry, P, invitation), lives_at("34 Cherry Lane", P).*

After the relaxation step, SLD resolution can be used to find related answers. [12] discusses search strategies for relaxation. The method has been incorporated into the cooperative answering system.

4 On Presenting the Cooperative Response in Natural Language

The cooperative answering strategies discussed in the previous section produce potentially large and complex logic formulae which are composed of logical expressions with a variety of origins, as seen in the breakfast flight example of Section 3.2. Even if the formula is presented in small portions, it still may be difficult for the user to read and understand the logic. The generation of natural language is a promising alternative method of presentation [2, 14].

Mapping each individual atom in a logical formula into an expression that emulates natural language is a straightforward process with the use of templates for each predicate. Unfortunately, the coherence and organization of the resulting text is limited to the organization that is accidentally present in the ordering of rules in the database and in the ordering of literals within the rules. In the direction of connecting individual pieces of text into a coherent whole, we have developed a series of linguistically motivated logic transformations that identify potential sites of coordination, subordination, and anaphora in the target text. We also have a method to select temporal

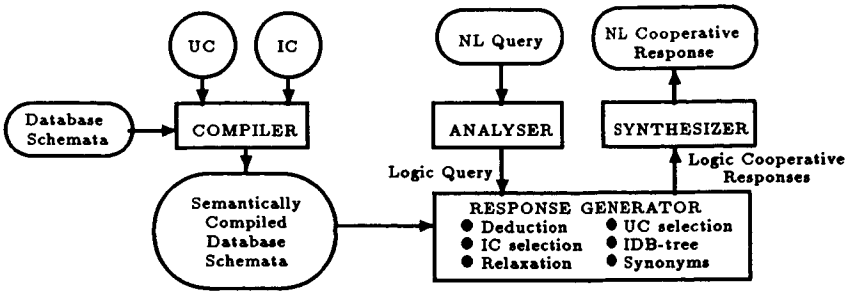

Figure 1: Cooperative Answering System At Maryland

connecting words, tense, and aspect [8]. To reduce redundancy across sentences, an additional transformation incorporates synonymous and generalizing substitutions into the target text.

The transformations take advantage of dependencies across literals, type information, and historical knowledge about how the formula was gathered. The coordination transformation uses a compact representation technique called π-notation [23] to help organize clauses. The computational theory of coordination used in the transformation builds on the work of [7, 9]. The selection of tense and aspect depends on temporal information available in the temporal database model of [31] and on temporal interval relationships [1]. This information enables the selection of allowable tenses using Hornstein's theory of tense [17], selection of aspectual feature values [5, 26] and selection of temporal connecting words such as "before," "while," "when," and "after." The incorporation of synonyms into the target text uses the taxonomy clauses and the relaxation method mentioned in the previous section to identify more concise representations of concepts in the logic formula.

Each transformation on the logic formula adds cohesion to or removes redundancy from the natural language text that is generated to describe the formula. The transformed logic formula is used as input to a natural language phrase generator to produce output text. The language generation work draws from and builds on work that has been done in the area of generating language from database information (e.g. [22]) and in the area of generating language from logic-based knowledge representations (e.g. [24, 32]). The language generation techniques used for cooperative answering with CARMIN are described in detail in [10].

5 Summary

The techniques discussed in this paper extend the cooperative answering system of Gal and Minker [25]. The user constraint, relaxation and search tree selection techniques complement the explanatory information provided by integrity constraints. They give information about how a user might construct new queries that will better serve the user's needs. CARMIN incorporates each of these techniques into a uniform user interface written in Prolog for relational and deductive databases. CARMIN provides a testbed for studying these and future cooperative techniques.

Figure 1 describes the structure of the cooperative answering system described here. Semantic query optimization techniques are used by a semantic compiler to integrate ICs and UCs with the database schema. For limited input language, a natural language analyzer borrowed from [29] parses natural language queries into queries in the language of the database — *logic queries*. The response generator processes a logic query with both the semantically compiled database schema

and the database itself to perform deduction, detection of IC and UC violations, restriction of the query with ICs and UCs, relaxation, and selection of information from the query's search tree. The response generator produces a cooperative response in the language of the database. The synthesizer then produces a natural language description of the logic response.

Semantic information in a deductive database is a relatively untapped source of information that can be used in responses to users' queries. Cooperative answers that use semantic information such as integrity constraints and user constraints can provide the user with information about the database's organization and the world modeled by the database. When queries fail or when a user wants additional answers, semantic information that describes taxonomy relationships between database predicates and constants can be used to find new queries that return answers related to the original query. For a background survey and comparison of approaches to cooperative answering, the reader is referred to [11].

References

[1] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–160, 1984.

[2] P. Amsili, A. Gal, F. Molines, P. Saint-Dizier, and E. Viegas. Some linguistic aspects of the generation of cooperative responses. In *ICO '91*, Montreal, Canada, 1991.

[3] U. Chakravarthy, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–274. Morgan-Kaufmann, 1987.

[4] W. W. Chu, Q. Chen, and R. Lee. Cooperative query answering via type abstraction hierarchy. Technical report, University of California at Los Angeles, October 1990.

[5] Bernard Comrie. *Aspect.* Cambridge University Press, Cambridge, England, 1976.

[6] F. Cuppens and R. Demolombe. How to Recognize Interesting Topics to Provide Cooperative Answering. *Information Systems*, 14(2):163–173, 1989.

[7] V. Dahl and M. McCord. Treating coordination in logic grammar. *American Journal of Computational Linguistics*, 9(2):69–91, 1983.

[8] Bonnie J. Dorr and Terry Gaasterland. Selecting tense, aspect, and connecting words: Generating from a temporal database. In *30th Annual Conference of the Association for Computational Linguistics*, 1992, submitted.

[9] Gazdar et al. *Linguistic Inquiry*, 13(4), 1982.

[10] T. Gaasterland. *Cooperative Answers for Database Queries.* PhD thesis, University of Maryland, Department of Computer Science, College Park, 1992. Dissertation work in progress.

[11] T. Gaasterland, P. Godfrey, and J. Minker. An Overview of Cooperative Answering. *Journal of Intelligent Information Systems*, 1992. To appear.

[12] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a Platform for Cooperative Answering. *Journal of Intelligent Information Systems*, 1992. To appear.

[13] T. Gaasterland, P. Godfrey, J. Minker, and L. Novik. CARMIN: A Cooperative AnsweRing MetaINterpreter. Technical Report to appear, University of Maryland, Department of Computer Science, College Park, Maryland 20742.

[14] T. Gaasterland and J. Minker. User needs and language generation issues in a cooperative answering system. In *ICLP Workshop on Advanced Logic Programming Tools and Formalisms for Language Processing*, Paris, France, June, 1991.

[15] A. Gal. *Cooperative Responses in Deductive Databases.* PhD thesis, University of Maryland, 1988.

[16] H. Gallaire and J. Minker, editors. *Logic and Databases.* Plenum Press, New York, April 1978.

[17] Norbert Hornstein. *As Time Goes By.* MIT Press, Cambridge, MA, 1990.

[18] J. M. Janas. On the feasiblity of informative answers. In H. Gallaire, J. Minker, and J.M. Nicolas, editors, *Advances In Database Theory: Volume 1*, pages 397–414. Plenum Press, 1981.

[19] S.J. Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19:165–187, 1982.

[20] Robert Kowalski. *Logic For Problem Solving.* Elsevier Science Publishing Co, Inc, 1979.

[21] K. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, 14:52–63, September 1988.

[22] K. McKeown. Generating natural language text in response to questions about database queries, 1982. Ph.D. Dissertation. University of Pennsylvania.

[23] J.R. McSkimin and J. Minker. Predicate Calculus Based Semantic Network for Question-Answering Systems, 1979.

[24] C. Mellish. Generating natural language explanations from plans. In L. Sterling, editor, *The Practice of Prolog*, chapter 6, pages 181–223. MIT Press, 1990.

[25] J. Minker and A. Gal. Producing cooperative answers in deductive databases. In P. Saint-Dizier and S. Szpakowics, editors, *Logic and Logic Grammar for Language Processing.* L.S. Horward, Ltd., to appear, 1990.

[26] Marc Moens and Mark Steedman. Temporal ontology and temporal reference. *Computational Linguistics*, 14(2):15–28, 1988.

[27] A. Motro. SEAVE: A Mechanism for Verifying User Presuppositions in Query Systems. *ACM Transactions on Office Information Systems*, 4(4), October 1986.

[28] M. Pollack. Generating expert answers through goal inference. Technical report, SRI International, Stanford,CA, October 1983.

[29] P. Saint-Dizier. Etude et realisation de esope. Technical report, Universite de Rennes, France, 1983.

[30] C. Shum and R. Muntz. Implicit Representation for Extensional Answers. In Larry Kershberg, editor, *Expert Database Systems*, Tysons Corner, VA, 1987.

[31] R. Snodgrass. Research concerning time in databases: Project summaries. *ACM SIGMOD Record*, 15(4):19–39, 1986.

[32] W. Swartout and J. Moore. Explanation in expert systems: A survey. Technical Report ISI/RR-88-228, Information Sciences Institutute, USC, Marina del Ray, California, December 1988.