Journal of Intelligent Information Systems, 1, 123–157 (1992)

An Overview of Cooperative Answering

TERRY GAASTERLAND
Department of Computer Science, University of Maryland, College Park, Maryland 20742

PARKE GODFREY
Department of Computer Science, University of Maryland, College Park, Maryland 20742

JACK MINKER
Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742

Abstract. Databases and information systems are often hard to use because they do not explicitly attempt to cooperate with their users. Direct answers to database and knowledge base queries may not always be the best answers. Instead, an answer with extra or alternative information may be more useful and less misleading to a user. This paper surveys foundational work that has been done toward endowing intelligent information systems with the ability to exhibit cooperative behavior. Grice's maxims of cooperative conversation, which provided a starting point for the field of cooperative answering, are presented along with relevant work in natural language dialogue systems, database query answering systems, and logic programming and deductive databases. The paper gives a detailed account of cooperative techniques that have been developed for considering users' beliefs and expectations, presuppositions, and misconceptions. Also, work in intensional answering and generalizing queries and answers is covered. Finally, the COOPERATIVE ANSWERING SYSTEM at Maryland, which is intended to be a general, portable platform for supporting a wide spectrum of cooperative answering techniques, is described.

Keywords: cooperative answering, query/answer systems, deductive databases, human/machine interaction

1. Introduction

Whenever we ask a question, we expect to receive a worthwhile answer in reply. This is certainly true in conversation, and this expectation remains whenever we interact with information systems. While people have always built databases and knowledge bases with this intention in mind, they have not necessarily succeeded.

Databases and information systems are often hard to use because they do not explicitly attempt to cooperate with their users. Unlike people, they answer literally the queries posed to them. Answers will be correct, but can often be misleading. Responses to queries may not contain the information the user really wants. A user may need more information, or might even need different information, than the query requests. The user may have misconceptions about the database and what the database knows.

If we are to build information systems that meet users' expectations, it is first necessary to determine what it means for an answer to be appropriate.

Grice (1975) proposes a number of maxims that characterize a *cooperative answer.* According to his maxims, a cooperative answer should be a correct, nonmisleading, and useful answer to a query. As we seek to endow information systems with cooperative behavior, the maxims can be used to serve as a measure of the cooperative performance of our information systems. A number of researchers have used Grice's maxims as guidelines in their work. The maxims describe fundamental properties of cooperative behavior. They describe what people do in conversation.

The *maxim of quality* requires that each contribution to a conversation be valid. One should only say things that one believes to be *true*, and one should not say anything for which one has inadequate evidence. In other words, one should not lie or overstate what one knows. Joshi (1982) extends this maxim to say that one should never say anything which may imply to the listener something that oneself believes to be *++false*. Given this cooperative maxim, both Joshi and Kaplan (1981, 1982) consider how to apply it to information systems and databases. In particular, a system should never give an answer which might mislead the user. If a literal answer to a query would lead the user to *false* assumptions, the system must provide further information to prevent this.

The *maxim of quantity* states that a contribution to a conversation should be as informative as required. However, it should not be more informative, or more detailed, than necessary. As databases become more complex and very large, an answer to a query can result in a huge table, much larger than is possible to read. In deductive databases, research has been done in *intensional answers*, which are succinct logical formulas that entail the answer. An intensional answer will often be as informative to the user as the table would be, and often will be much easier to understand. Intensional answers also can be more informative than the table itself by informing the user of general facts about the table.

The *maxim of relation* says a contribution should be relevant. In particular, an answer should be relevant to the user who asked the question. Much work (Allen, 1987; Allen and Perrault, 1986; McCoy, 1984, 1988; Pollack, 1983) has been done to model users and to determine a user's goals and intentions. Their work provides a basis for determining whether an answer is relevant.

The *maxim of manner* states one must avoid obtuse expression and avoid ambiguity. Also, one must be brief. When an information system provides an answer, it ought to be presented in a form that is easy for the user to understand. It should not be ambiguous, leaving the user with choices to make about its meaning. Otherwise, the answer could be misleading. The answer should be as succinct as possible without encroaching on any of the other criteria.

In part, Grice's intent in introducing a set of conversational maxims was to show that first-order logic suffices to model cooperative dialogue. The maxims describe what should happen in a natural conversation. They also serve to explain when a conversation fails to be cooperative. One can evaluate whether an answer is cooperative with respect to the maxims and generate cooperative responses by adhering to them.

Others have held the view that more powerful logics, such as logics of intentionality and modality, are needed to handle cooperative discourse. Many of the philosophical considerations expressed in Smith (1987) reflect this concern. Of course, whether or not first-order logic can serve as an adequate ontology for cooperative answering is an open question. Much research is being conducted on nonclassical and higher-order logics. These may play a role in future cooperative answering work.

Once we have established the criteria for an answer to be cooperative, we must determine how to build information systems that exhibit cooperative answering behavior. Databases, and information systems in general, satisfy the standard of providing *correct* answers, in so far as the system contains valid information. However, for answers to be useful to users, they must ask the appropriate queries in order to arrive at the desired information. To do so, the user must know something about the database's schema, and just what it is that the database knows. When a user's suppositions about the system are inaccurate, most information systems do not do much in the way of correcting the user. Once users do ask the queries that comply with the database organization, the answers returned may still be misleading, less informative than desired, or ambiguous.

To address these problems, specific cooperative techniques have been developed to identify and correct *false presuppositions* and *misconceptions* apparent in the question. Other techniques ensure that answers are *relevant* with respect to the user's intentions. When an answer consists of several parts, it is important to try to make the answer *cohesive* across these parts, and *coherent* as a whole. Finally, if the answer is presented in natural language, the response should exhibit an appropriate conversational organization.

Our focus in this survey is on the nascent field of cooperative answering for query/answer systems. Some of the key topics involved are

- Maxims and criteria for cooperative behavior (as seen above)
- Techniques to evoke cooperative behavior in query/answer systems
- Issues of computational cost of such methods and containing that cost
- Knowledge representation needs for constructing cooperative answers

In the next section, we review foundational efforts in cooperative answering. In Section 3 we look at specific cooperative answering strategies and systems. These strategies are universal in that any system ought to support them and all users require them. In Section 4 we consider when an answer ought to be tailored for a particular user or class of users. This is necessary to attend properly to a user's particular needs and requirements. In Section 5 we review the work in cooperative answering at the University of Maryland at College Park. In finishing, we attempt to outline the directions in which the field of cooperative answering is evolving, which direction we should go, and further problems to be solved.

2. Query/answer systems

Research in cooperative answering has grown out of three areas in which question-and-answer discourses arise:

1. Natural language interfaces and dialogue systems built for the purpose of studying natural language exchanges between users and computers
2. Databases
3. Logic programming and deductive databases

In this section, we discuss each in turn.

2.1. Natural language interfaces

Much of the initial work in cooperative answering focused on natural language dialogue systems (Joshi, 1982; Joshi et al., 1982; Lehnert, 1981; Pollack, 1982). Researchers sought to endow their information systems with the same normative cooperative behavior as described by Grice as occurring in human conversation. Many of the systems we discuss in this survey employ natural language interfaces or are tightly integrated with natural language facilities. The focus on natural language within the nascent field of cooperative answering can be considered both distracting and beneficial in different ways.

Some researchers are interested in modeling human conversation. Others hope to use human discourse as a normative standard for information systems' behavior. Many believe natural language dialogue systems ought to be subject to the same cooperative conversational maxims as human discourse. Such systems are ideal settings for evaluating a systems' behavior against the conversational maxims.

Natural language interfaces along with more fully integrated natural language query/answer systems are important in another way too: for information systems to be accessible by lay users requires an easily understood interface language. Many hope that natural language, or a subset thereof, will fulfill this need. As information systems become ever larger and more complex, we all become lay users, and the need for natural interfaces becomes even more vital. One cannot expect all users to understand a complex database's schema. Natural language conversation techniques are also for answering queries in an interactive fashion when answers become long and complex. One cannot expect users to understand overly complex answers expressed in arcane formalisms or expressed all at once.

While the research in natural language systems is important in its own right, it risks confounding the issues faced in cooperative answering. The tenets of cooperative answering ought to be universal to queries themselves, regardless of whether the queries are cast in natural language, in logic, or in *SQL*. (Of course, there are many other issues particular to natural language.) The same maxims

for cooperation discussed above ought to apply to all information systems.

This paper covers specific topics in natural language research when they relate directly to cooperative answering systems. There are many more topics indirectly related to cooperative answering with extensive literatures that we do not cover. We look at a number of natural language systems of importance to the cooperative answering field in Section 3.1. However, our focus is on the cooperative issues they tackle. We attempt to clarify the relation between natural language and cooperation in such systems as best we can throughout the survey. We do not consider the particulars of the natural language components in cooperative systems.

2.2. *Databases*

A number of researchers in the area of databases (Chu, et al., 1990, 1992, Kaplan, 1982; Motro, 1986) have recognized the practical need for cooperative answering behavior in standard, widely available information systems. This need has led people to consider how to adapt and develop cooperative techniques specifically for databases.

Kaplan (1982) built a cooperative system on top of *CODASYL* (a network-model database) (CODASYL, 1971; Ullman, 1988). He pays special attention to making his system *portable*: the cooperative answering behavior is independent of the information system's knowledge and domain. His system was portable not only between databases using the network model, but also between other types of information systems too, such as relational databases and expert systems. Mays (1980) builds on Kaplan's work and considers how to use a relational database's schema to correct *false presuppositions* in a user's query.

Motro (1986) considers modifications to the relational model which would be easy to incorporate into a relational database system to allow for certain cooperative behaviors. In Motro (1990) he considers a user interface for relational database systems that would allow users to interact with the database in more cooperative ways. The database can correct queries that have apparent mistakes or that return no answers, and yield to the user *corrected* queries with which to proceed. He also explores the notion of returning to a user related queries that may closer match the user's intended question.

Chu, Chen, and Lee (1990, 1992) explore methods to generalize and refine queries to generate new queries related in direct ways to the user's original. Such related queries can find answers outside of the scope of the original query which may be of interest to the user.

2.3. *Deductive databases*

A large portion of the work in cooperative answering has been done within deductive databases. The logic model both subsumes and extends the relational model (Ullman 1988) for databases. The logic model offers a uniform representation for data, knowledge,[1] and queries. Logic also offers a suitable representation for expressing cooperative answers, whereas *SQL* and other database languages must be extended to do this. Finally, the rules and integrity constraints of deductive databases allow for a rich semantics, which plays a vital role in generating cooperative response. In the remainder of this section, we provide basic definitions about deductive databases that will be used throughout this paper.

A deductive database (Gallaire, and Minker, 1978; Gallaire et al., 1984; Minker, 1988) employs the *logic model* instead of the relational model as its data model (Ullman, 1988). A deductive database consists of three parts: *facts, rules* and *integrity constraints.* We denote this by the tuple ⟨*EDB,IDB, IC*⟩. A *rule* is of the form

$$A_0 \leftarrow A_1, \ldots, A_n. \qquad (C_1)$$

A rule is a clause, and C_1 is equivalent logically to $\forall.\ A_0 \vee \neg A_1 \vee \ldots \neg A_n$. The A_i's, $0 \leq i \leq n$, are atoms. For this paper, a clause contains a single positive atom, A_0, and some negative atoms, $A_1, \ldots, A_n$, $n \geq 0$. An *atom* is a predicate instantiated across variables and constants. (*Logic programs* are more general in that they also permit functional terms.) For instance, *student(chris)* and *student(X)* are atoms of the unary predicate *student/1*. The former is over the constant *chris*, the latter over the variable X.[2] Each variable appearing in a clause is considered to be universally quantified across the clause.

The Atom A_0 in C_1 is called the *head*, and $A_1, \ldots, A_n$ the *body*. The atom A_0 is necessarily a logical consequence of the database if each A_i, $1 \leq i \leq n$, is a logical consequence of the database. A *ground* atom is one that contains no variables. A *fact* is a ground clause with an empty body, $n = 0$.

The set of all facts constitute the *extensional* database (*EDB*), the set of rules the *intensional* database (*IDB*). It is assumed that a given predicate either appears only in facts or in the heads of rules. The *EDB* predicates are equivalent to relations from the schema in a relational database, and individual facts to individual tuples. The rules for the *IDB* predicates are analogous to views, but are more powerful than views as they admit recursion. Clauses and facts as we have defined them constitute *DATALOG*, and a finite set of such clauses and facts is a *DATALOG* database (Ullman, 1988).

A relational database is conceptualized as two parts: its *schema* and its *state*. The schema is its relations and views. The state is the database's collection of tuples, the data. Databases are designed with the assumption that the schema seldom changes and remains fairly constant through time, but the state will change rapidly as new data are added and old data are revised. Likewise, in

a deductive database, it is assumed the *IDB* and the $\mathcal{IC}$ are mostly static. The *EDB* will change often as new facts are added and old ones revised.

The type of question permitted in a deductive database is an existentially quantified conjunction of atoms. For instance, if one wanted to know whether there is a student who has passed *CMSC 420*, the logical formula would be

$$\exists X.\ student(X) \wedge passed(X, \text{'CMSC-420'})?$$

It is standard to take the negation of the question, which is then called the *query*, to resolve against the database. A query then is a clause, and has the same form as a rule but without a head. For instance, our questions above is represented as

$$\leftarrow student(X), passed(X, \text{'CMSC-420'}). \qquad (\mathcal{Q}_1)$$

An *answer* to a query is a substitution of the query's variables such that (the negation of) the substituted query is a logical consequence of the database. For instance, X = *chris* is an answer to $\mathcal{Q}_1$ if *student(chris)* and *passed(chris, 'CMSC-420')* follow from the database.

The third component of a deductive database is an associated set of integrity constraints, $\mathcal{IC}$. An *integrity constraint (IC)* is a logical formula that must be *true* of the database.[3] We often limit our concern to *denial* constraints, *IC*s which state facts that cannot be simultaneously *true* in the database. For example, the following *IC* states that no one can be both a student and a professor:

$$\subset professor(X), student(X). \qquad (\mathcal{IC}_1)$$

One may note that (denial) integrity constraints have the same syntactic form as queries,[4] but a different meaning with respect to the database. We may write other *IC*s in the same form as denials, but allowing for negation in the body. For instance,

$$\subset student(X), not\ enrolled(X). \qquad (\mathcal{IC}_2)$$

states that all students must be enrolled. In other words, if someone is a student, then it is not possible that he or she is not enrolled. This *not* is not classical negation,[5] but negation as failure (Shepherdson, 1984; Lloyd, 1987). Thus, *not enrolled(chris),* say, is *true* if *enrolled(chris)* cannot be proven. Such constraints are more expressive and we use such *IC*s in some of our examples.

We now turn to a detailed overview of cooperative techniques that have been developed and explored in each of these three areas over the past two decades.

3. Foundational work in cooperative answering

The cooperative answering techniques that have grown out of research in each

of the areas that were reviewed in the previous section can be separated into five categories. The techniques in each category are differentiated by the following capabilities.

1. Consideration of specific information about a user's state of mind
2. Evaluation of presuppositions in a query
3. Detection and correction of misconceptions in a query (other than false presuppositions)
4. Formulation of intensional answers
5. Generalization of queries and of responses

This section describes each in turn.

3.1. Beliefs and expectations

The genesis of the field of cooperative answering can, for the most part, be traced to work by Joshi and Webber at the University of Pennsylvania. In May of 1978 a workshop entitled *"Computational Aspects of Linguistic Structure and Discourse Setting"* was held, sponsored by the Sloan Foundation at the University of Pennsylvania. Several topics were discussed: question-answering—intensional verses extensional responses, direct and indirect answers; meaning representations—logical representations and their role in natural language processing; and inferences in context. A book ensued, *Elements of Discourse Understanding*, in 1981, edited by Joshi, Webber, and Sag, based upon the articles presented at the workshop (Joshi et al., 1981).

Most of the articles in the book fall under the following topics:

1. Utterance meaning *(understanding the questions and answers)*
2. Discourse models *(following the query/answer discourse)*
3. Models of participants' beliefs *(mutual ground between user and system)*

For query/answer systems, topic 1 concerns whether the system can understand and properly interpret the query. This covers a number of important issues. For instance, natural language queries can be ambiguous and must be disambiguated to be answered. Lehnert (1981) has noticed that a given query might be asked with different intentions in mind. Thus a user's intentions and goals must be taken into account. (See Section 4.) Even with logical queries (not asked in natural language), proper interpretation of the query is necessary. Cuppens and Demolombe (1988) consider rewriting queries to include more information in the answers than would be gathered with the original query in order to match a user's intentions and to provide generally more informative answers. (See Section 3.5.)

Topic 2 concerns whether a system is able to model its dialogue with a user. A system should account for a user's goals changing over time and what the user has learned through previous dialogue with the system (a building of *mutual knowledge*). Also, the system should gain a better understanding of a user's intentions as the dialogue progresses. Webber is interested in how aspects of discourse might shape the mental model that a listener has. Webber (1981) considers how people understand (definite) anaphora. McKeown (1982) and Gaasterland (1992a,b) have both considered such issues as involved in the presentation of answers which are crucial to the user's understanding of the answers.

In general, Webber's work considers user beliefs in order to anticipate user expectations (topic 3). She explores how to provide extra information to prevent misconceptions. For example:

Q: *"Is Sam an associate professor?"*
User believes most associate professors have tenure.
Sam is not tenured. Sam is an associate professor.

A: *"Yes, but he doesn't have tenure."*

Pollack, Hirschberg, and Webber (1982) consider the effects a user's interaction should have in the reasoning of expert systems. Joshi, Webber, and Weischedel (1984) consider how to generate expert responses that match the user's expectations. Webber and Mays (1983) focus on detecting and correcting misconceptions (Section 3.3).

Joshi (1982) considers some of the abstract problems of establishing mutual knowledge in question/answer systems. The paper appears in the 1982 collection *Mutual Knowledge*, edited by N.V. Smith (1982).[6] The book is a philosophical treatise on mutual knowledge: what mutual knowledge is, its ramifications in dialogue, and how it is established. Joshi, in his article, considers the ramifications for cooperation in dialogue.

Lehnert (1981) devised a system to analyze questions within an ongoing discourse between a user and a computer in order to determine the user's intent behind the questions. Her system strives to answer questions in a cooperative manner. Her system, called *QUALM*, parses its English questions into a conceptual dependency representation (Schank, 1975) which represents the question's meaning. It analyzes the question and surrounding context in the conversation to decide the question's intent. The analysis helps determine the type of answer the system should provide.

One might respond to the question below with any given one of the following answers, depending on what intent is behind the question.

Q:	*"How did John take the exam?"*	(intent?)
A_1:	*"He crammed the night before."*	(enablement)
A_2:	*"He took it with a pen."*	(instrumental/procedural)
A_3:	*"He took it badly."*	(emotional)[7]

Lehnert points out that her system is more than a natural language front-end. The conceptual dependency representation also serves as the core knowledge representation, and for her system there is necessarily no clear distinction between translating the query and answering it.

Many of Webber's and Joshi's students have worked in this area, and established many cooperative answering techniques. Mays (1980) considers how to employ the database's schema to detect user misconceptions (as discussed in Section 2.2). Kaplan (1981, 1982) detects presuppositions in queries, and Hirschberg (1983) accounts for implicatures in answers. (See next section.) McKeown (1982) considers natural language response to database queries. Pollack (1983) studies how users' questions fit with their plans. (See Section 4.)

3.2. Presuppositions

A student asks an appropriate university database

"Who passed CMSC 420 in the fall semester of 1991?"

The database returns with the answer *"No one,"* leaving the student to think, possibly, that *CMSC 420* is a very hard course. The student then asks

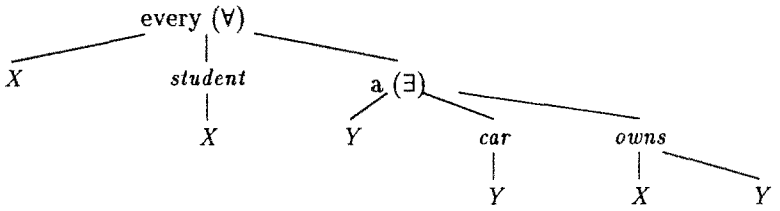
"Who failed CMSC 420 in the fall semester of 1991?"

Again, the database returns with the answer *"No one."* Finally, the student is suspicious and asks

"Who taught CMSC 420 in the falls semester of 1991?"

The database answers again *"No one"* (Kaplan, 1982).

Kaplan calls this behavior *stonewalling*. If the initial question had been asked to a person instead, he or she would have probably answered immediately, with a reply such as *"Oh, there was no such course taught last semester."* Databases stonewall. They will answer a *yes/no* question with a *yes* or a *no* regardless of whether the answer is misleading. As Grice said though, correct answers are not enough. Clearly a cooperative answer would be an explanation of why the query fails (why the answer is *no*), exposing any false presuppositions. Kaplan states that one should consider the presuppositions in a query to be statements that must be *true* for the query to have an answer (Kaplan, 1981, 1982). If any presuppositions are *false*, the query is nonsensical.



"Every student owns a car."

$$\forall X.[student(X) \longrightarrow \exists Y.[car(Y) \wedge owns(X, Y)]]$$

Fig. 1. Quantifier tree representation.

Kaplan built a system called *CO-OP* (A Cooperative Query System), a natural language query system that operates with a *CODASYL* database management system (Codasyl, 1971; Ullman, 1988) *CO-OP* provides cooperative responses to simple natural language questions, requesting the relevant data from the database. The system runs as two processes: a *FORTRAN* process of the *SEED* database system (*CODASYL*) with a query processor and a *LISP* process of the natural language front-end with a report writer. The system has been tested over a real database from the National Center for Atmospheric Research over a domain on users and programmers.

Kaplan devises and employs an intermediate representation for queries he calls the *Meta Query Language (MQL)*. A query is represented as a graph structure in *MQL*, consisting of sets at the nodes and binary relations between sets (the arcs). The graph is a semantic network, and the query is re-expressed in a binary notation. The query answering system checks that each connected subgraph is non-empty. If any is empty, this is a failed presupposition that ought to be reported.

Presumptions are distinguished from *presuppositions*. All presuppositions of a query must hold for the query to make sense. However, the query may contain presumptions too. If a presumption does not hold, then the query will have only one answer (usually *no*). For example,

"What day does Mary have her lesson?"

presupposes that Mary has a (weekly) lesson. If she does not, the query cannot be answered adequately. In other words, no day of the week can be given in reply. The reply *"Mary has her lesson on no day."* would not be a correct response since it, too, seems to presuppose that Mary has a lesson. (But when?) The query

"Did Mahler complete an 11^{th} *symphony?"*

presumes that he began one, but it does not *presuppose* that he began one (Kaplan, 1982). We can answer meaningfully and correctly *"No"* even if he never began his 11^{th} symphony, albeit *"No, Mahler never began an 11^{th} symphony."* would be a more *informative* answer.

Presumptions are a wider class than presuppositions. They are also much harder to handle. For instance, the query *"Did it rain yesterday?"* could be answered *"No, there were no clouds yesterday."* This entails a fair amount of reasoning and domain knowledge, and can be considered a type of explanation. A system must be able to conduct such reasoning, and an appropriate explanation for the user must be chosen from many potential explanations. We explore these issues next in Section 3.3. when we discuss the closely related topic of misconceptions.

A query containing a false presupposition can be considered to have no answer, positive or negative, even though the database may be able to deliver a "literal" answer. Colmerauer and Pique noticed this dilemma in their work to translate natural language queries into a logical formalism (Colmerauer and Pigue, 1981). They employed a three-valued logic that allows a sentence to be marked *undefined* when such false presuppositions occurred. (But in their work they do not develop a means to identify false presuppositions to the user.)

In particular, in a natural language sentence, a speaker assumes the existence of the subject and objects (the noun phrases) of the sentence. If any of these do not exist, then the sentence does not have any clear meaning. Colmerauer and Pique translate natural language sentences (and queries) into a logical tree representation called *three branch quantifier trees (3BQs)*. The first branch is a variable introduced in the sentence, the second a translation of the subject (this could be recursive), and the third a translation of the rest of the sentence (also can be recursive). In Figure 1 the statement *"Every student owns a car."* is represented as a *3BQ* tree. Once a sentence is translated, they check that none of the typing relations (for instance, *student/1* and *car/1*, unary predicates) representing the noun phrases are empty. If there are no students, then the sentence from Figure 1 is deemed to be nonsensical.

Usually when one asks a query, one not only presupposes the existence of all the components of the query, but one also presupposes an answer to the query itself. For instance, suppose one asks *"Which employees own red cars?"* (query Q_2, see Figure 2). Colmerauer and Pique are concerned that there be cars and employees known to the database. Otherwise, there would be a false presupposition in Q_2, and so no valid answers.

On asking a query, one assumes there are answers to the query too. If there are no answers, this deserves an explanation. For instance, if there are no employees who own red cars, the database can answer literally *"No one"* to query Q_2. To be more informative, if there is some reason why there are no answers, the database should say so. (We discuss this in Section 3.3.) In the least, the part of the query responsible for failure ought to be identified. There may be no employees owning red cars because no employee owns a car at all. Thus, the subquery $\langle \leftarrow employee(X), owns(X,Y), car(Y). \rangle$ would fail too.[8] Reporting this

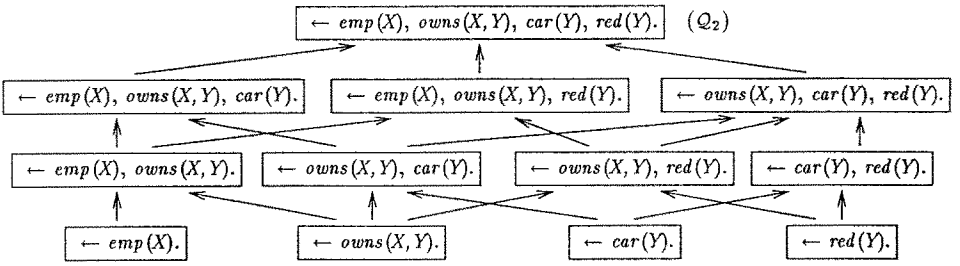


Fig. 2. A lattice of subqueries.

failure is more informative than just reporting the failure of Q_2.

Janas (1981) studies the feasibility of yielding such informative answers. His idea is to report the smallest subqueries that fail. If we consider a conjunctive query as a set of atoms to be satisfied, then the subqueries are the elements of the power set. In all, there are 2^n- 2 subqueries for a conjunctive query with n atoms, disregarding the query itself (which has already been seen to fail) and $\emptyset$, the empty query. Query Q_2 has 4 atoms, so there are 14 subqueries to consider. The naive approach is to test all of them. This incurs exponential cost over the length of the query. More clever algorithms would cut this cost, especially for average case, but cannot reduce the worst case behavior to better than exponential.

Janas shows that for most queries, many of the potential subqueries may be discounted. He defines that two atoms in a query are *joined* if they share a variable. The relation *connected* is then the transitive closure of *joined*. Janas defines a query to be *connected* if every two atoms in the query are connected.[9] It is reasonable to insist that queries be connected. (A user can always ask any disconnected parts as separate queries.) In searching for failed subqueries, it is only necessary to consider connected subqueries. For instance, consider the subquery $\langle \leftarrow employee(X), car(Y), red(Y). \rangle$. This query is not connected. It can only fail if $\langle \leftarrow employee(X) \rangle$ or $\langle \leftarrow car(Y), red(Y) \rangle$ fails. These latter subqueries are connected and will be considered anyway. It is not necessary to properly compute answers for each subquery considered; it is only necessary to compute if it has *an* answer.

Even if the cost of this processing remains high, it may be offset by the benefits that these answers offer. When informed of the part of the query that fails, the user will not waste time asking many follow up questions that

also will necessarily fail while looking for the information that he or she is still trying to find. The overall reduction in cumulative query-answering cost could be significant, especially if the user would have asked many spurious queries (as in Kaplan's stonewalling) to get the desired information. The user can quickly become frustrated and dissuaded from his or her goal.

Kaplan (1982) also considers the necessity of finding the smallest failing subqueries and devises an algorithm very similar to Janas' that operates over a query translated into his *MQL*, but he does not consider the computational issues involved. He points out that his (as does Janas') algorithm to find presumptions (failed subqueries) is independent of domain specific knowledge. These are techniques that are applicable over any domain. Kaplan also introduces the notion of generalizing a query: if a query fails due to a failing subquery, the failing part can be removed, resulting in a new query that will have answers. This can serve as a tool in correcting possible errors in users' queries, and to give the user information *related* to the query asked. (We elaborate on this last idea in Section 3.5.)

The statement *"No employees own red cars."* implies that employees do own cars. Otherwise, a stronger assertion should have been made according to the maxim of quantity. A response should be as informative as necessary, but no more so. In particular, Janas' queries and subqueries lie on a scale: if a subquery has no answers, then the query cannot have any answers either since the subquery logically subsumes the query. Such a tacit implication is called a *scalar implicature.*

Hirschberg (1983) considers how scalar implicatures are involved in obeying the maxim of quantity. Many such implicative scales exist and are employed in conversation. Consider the following piece of dialogue, which takes place between, say, an engineer and an assistant across a remote radio link:

Engineer: *"Did you loosen the hose?"*

Assistant: *"I have detached it."*

If the assistant were to have (truthfully) answered *"Yes"* (he or she did *loosen* the hose in the process of detaching it), this would mislead the engineer, who assumes the whole story has been told, as in *"Yes, I have loosened it (but no more)."* Going from *loosen* to *detached* is a scale, the latter implying the former. It is assumed in conversation that one always give the response *highest* on such a scale that is true, that it implies the most possible and all implied is true.

Q: Engineer: *"Are mushrooms poisonous?"*

A: Assistant: *"Some are."*

Degree of existence is also a scale. To the question above, one can not answer correctly *yes* or *no*. Some mushrooms are edible, others are poisonous. The answer to the universal question *"Are all mushrooms poisonous?"* could be *"No,"*

but this would be misleading. The cooperative response would be *"No, but some are."*

3.3. Misconceptions

A query may be free of any false presuppositions, but still harbor misconceptions. False presuppositions concern the schema of the knowledge base. In contrast, misconceptions concern the domain of the knowledge base, and what is and is not possible within the domain. Misconceptions can mean a query will not have any answers, or that some subquery will not lead to any answers. Misconceptions can lead to logical redundancy in a query, so the query overspecifies the solutions, indicating the possibility that the user is unaware of certain properties of the domain. While false presuppositions usually occur with respect to the database's state and schema, misconceptions usually occur with respect to the database's semantics. Misconceptions arise when the user has a false or unclear understanding of what is necessarily *true* or *false* in the database.

Mays (1980) employs schema information of a relation database to correct false presuppositions. By employing schema information, he introduces aspects of the database's semantics into answers and can detect and correct misconceptions of the user with respect to the database schema. For instance, say that only students *take* courses, whereas professors *teach* courses. Even though the following query is destined to fail having no answers, it is more informative to correct the user's misconceptions of the database by explaining why there are no answers.

Q: *"Which professors take CMSC 620?"*

A: *"None."*
"Professors teach courses."
"Students take courses."

McCoy (1984) uses world (or general) knowledge to correct *object related* misconceptions that a user might have, misconceptions about the properties of a given object or class. For instance, a user might ask:

Q: *"Where are the gills on a whale?"*
The system knows the user probably thinks whales are fish because fish use gills to breathe.

A: *"Whales do not have gills. They breathe through lungs."*

Whenever the user asks a query that *cannot* have an answer, the system infers the probable mismatches between the user's view of the world and the knowledge in the knowledge base. The system then answers with a correction to rectify the mismatch.

Again, Janas (1981) considers the use of *IC*s to eliminate subqueries from consideration. He shows how *IC*s that apply to a query can be used to rule out presuppositions that are certain to be *true*. An *IC* might reduce a query to a subquery. For instance, a constraint

$$\subset car(X), not\ red(X).$$

stating that all cars are red would reduce $\mathcal{Q}_2$ to $\langle \leftarrow emp(x), owns(X, Y), car(Y).\rangle$ If something is a car, it is certain to be red. An *IC* can logically subsume a query, meaning the query cannot have answers. For instance, say

$$\subset emp(X), owns(X, Y), car(Y).$$

were a constraint itself, meaning that employees do not own cars. This is a necessarily failing subquery of $\mathcal{Q}_2$, thus $\mathcal{Q}_2$ itself must fail. The *SEAVE* system of Motro (1986) also uses integrity constraints together with false presuppositions in order to provide additional information about failed queries. Consider the query

$$\leftarrow professor(X), enrolled\_in('CMSC\text{-}420'). \qquad (\mathcal{Q}_3)$$

in a database with the following integrity constraints

$$\subset professor(X), student(X). \qquad (\mathcal{IC}_1)$$
$$\subset enrolled\_in(X, Y), not\ student(X). \qquad (\mathcal{IC}_3)$$

The $\mathcal{IC}_1$ states that professors cannot be students (and the contrary as well) and $\mathcal{IC}_3$ states that all students are enrolled. The query $\mathcal{Q}_3$ violates these *IC*s. Semantic query optimization techniques (Chakravarthy, 1985; Chakravarthy, et al., 1986a,b) apply a set of constraints to a query. If any constraint, or set of constraints, are violated, the query is known to fail before any database search takes place. Not only do the constraints identify failed queries, but they also provide information about why the queries fail. Furthermore, they notify the user that if the same query is asked again in the future, the query will still fail, even if the state of the database has changed.

The realization that failure assured by *IC*s is more meaningful than just exhaustive failure motivated the work in cooperative answering by Gal (1988) and by Gal and Minker (1985, 1988). They used the semantic optimization techniques (Chakravarthy, 1985; Chakravarthy, et al, 1986a,b) to determine when *IC*s applied to a query. The system developed by Gal (1988) identifies the *IC*s that guarantee failure. It then includes the constraints in an answer to the user. For the example above with query $\mathcal{Q}_3$, a cooperative response could say

"No one is both a professor and a student.
Anyone who is enrolled in a class is a student.
So on one is a professor and enrolled in a class."

Gal's work also addresses some of the difficulties encountered with *IC*s are used for explanation. Sometimes several sets of *IC*s can independently explain the failure. In this case, Gal's system uses heuristics to select the best explanation to provide the user. Sometimes explanations can be quite complex and involved, to the point they would tax a user's attention and interest. The user is allowed to impose a limit on the content of an explanation prior to asking a query.

*IC*s can be used to do more than explain failed queries. Gal and Minker use them to summarize very large answers. When an *IC* indicates that a query must be *true*, the *IC* can be returned as a summary. For example, the query *"Which professors teach classes?"* would be answered by an *IC* that states *"All professors teach classes."* This is a type of *intensional* answer, as will be seen in the next section. In addition, *IC*s can identify logical redundancies in a query or subquery. For example, consider the query

$$\leftarrow \textit{student(susan), enrolled(susan).} \qquad (\mathcal{Q}_4)$$

to a database that has the associated *IC*

$$\subset \textit{student}(X), \textit{not enrolled}(X). \qquad (\mathcal{IC}_2)$$

which states that all students are necessarily enrolled. This means the query is overspecified. Simply asking

$$\leftarrow \textit{student(susan).} \qquad (\mathcal{Q}_5)$$

would be equivalent. If Susan is indeed a student, a cooperative response would say

"Yes, Susan is a student.
By the way, all students are enrolled."

3.4. Intensional answers

Instead of always responding to a query with a substitution for the variables in the query, it is sometimes more appropriate to provide the user with an *intensional* answer. An intensional answer denotes the complete set of answers, or some subset thereof, to a given query. Query $\mathcal{Q}_4$ which stated *"Which students are enrolled?"* was overspecified since all students were necessarily enrolled. Enumerating the answers, say, *chris, terry, josé, carol, ...* is misleading, unless it is believed that the user would know that this enumeration constitutes the set of all students. Thus, the answer $\forall X.\textit{student}(X)$ is better. Such an answer is called a *universal* answer as it is a universally quantified formula.

Motro (1991) draws a distinction between *data* and *knowledge*. In deductive databases, the data are represented in the *EBD*, and *knowledge* in the *IDB* and *IC*. Knowledge is the semantics of the databases, that which must be *true* of

the database's state, and the logical conclusions that must follow from given data. Usually, answers to a query are only in terms of data. Cooperative answers incorporate knowledge in the response. In particular, *IA*s are in terms of knowledge, not data.

Intensional answers (*IA*s) can be more informative than enumerated answers. They can teach users about the structure of the database and of the domain and help to clear up misconceptions. Intensional answers can be more succint than concrete answers; this is an important cooperative behavior when databases contain huge stores of data. Users are often overwhelmed by the size of the responses of their queries. In logic programs, it is possible for a query to have an infinite answer set. *IA*s can be used to characterize answer sets of infinite cardinality (Chomicki and Imielinski, 1989).[10] Intensional answers can be provided in lieu of concrete answers when the database system is under time and computational constraints (Imielinski, 1988).

Let us formalize the notion of *IA*s. Let the query be an existential query (Section 2.3), $\exists \vec{v}.\mathcal{Q}$. ($\mathcal{Q}$ here is not the negated formula for refutation.) Let $\vec{v}$ be the vector of all the variables of $\mathcal{Q}$. Let $\mathcal{F}$ be a formula over $\vec{v}$ and $\vec{u}$ where $\vec{u}$ are the variables that occur in $\mathcal{F}$ but not in $\mathcal{Q}$. The vector $\vec{u}$ can be empty. There are three definitions considered: $\forall\vec{v}\exists\vec{u}.\mathcal{F}$ is an intensional answer of the query $\mathcal{Q}$ *iff* **I, II,** or **III** alternatively:

I. $\forall\ \vec{v}\ \exists\ \vec{u}.\ \mathcal{F} \iff \mathcal{Q}$ (equivalence)
II. $\forall\ \vec{v}\ \exists\ \vec{u}.\ \mathcal{F} \implies \mathcal{Q}$ (sufficiency)
III. $\forall\ \vec{v}\ \exists\ \vec{u}.\ \mathcal{F} \impliedby \mathcal{Q}$ (necessity)

First, we must consider how *IA*s can be computed. The number of potential *IA*s for a query is infinite since an *IA* is a formula over the (nonempty) theory of the database, and there are an infinite number of tautologies. We want to find only the interesting ones. One notion of *IA*s is that they are partial evaluations of the query. In a resolution based refutation proof procedure such as *SLD-resolution* (Lloyd, 1982), the query is *rewritten* at each step (in a deductive, sound manner) by a clause from the database until the empty clause remains. (A goal atom is removed from the goal list when it unifies with a fact.) The resulting substitution is an answer.

Rules: $B \leftarrow C_1,\ C_2.$
$\vdots$

G_0 : $\leftarrow$ $A_1,\ A_2,\ \ldots,\ A_m.$
G_1 : $\leftarrow$ $(C_1,\ C_2,\ A_2, \ldots, A_m)\theta_1.$ $\quad A_1\theta_1 = B\ \theta_1$
$\vdots$
G_n: $\leftarrow$ $\square$ Answer $= \theta_1 \ldots \theta_{n-1}$

The query is $\mathbf{G}_0$ above. The substitution $\theta_1 \ldots \theta_{n-1}$ at step $\mathbf{G}_n$ is an answer to the query. Once the definition for answer is revised to be any formula (meeting criterion **II** above) rather than simply being a substitution, any of the partial solutions above, $\mathbf{G}_1, \ldots, \mathbf{G}_{n-1}$, may be considered as *IA*s. These are intermediate answers in between the concrete answers found at the leaves of the solution tree and the query at the root. These *IA*s are dependent on the search (proof) tree. Another notion of intensional answer is that it characterizes a set of answers (substitutions) in some natural way, independent of the search tree.

Imielinski (1988) started the work in intensional answers. He defined an intensional answer to be a rewrite of the query that preserves its semantics. This satisfies the strict equivalence definition (**I**) for *IA*s. In the degenerate case, a query is an intensional answer to itself. The rewritten formula, the *IA*, denotes the same answers as the query. For an *IA* to be an adequate answer, it must satisfy the user as being a sufficient, understandable characterization of the ground answers to the query. Obviously, the query itself is an *IA* by Imielinski's definition; however, it is not an adequate answer, since otherwise the user would not have asked the query.

EDB:	*prerequisite('MATH-300', 'MATH-350')*	
	prerequisite('MATH-350', 'MATH-400')	
	teaches(smith, 'MATH-400')	
IDB:	*teaches(X, Y)* $\leftarrow$ *teaches(X, Z), prerequisite(Y, Z).*	
Q:	*Q(A)* $\leftarrow$ *teaches(smith, A).*	($\mathcal{Q}_6$)
IA:	*Q(A)* $\leftarrow$ *prerequsite(A, B).*	($\mathcal{C}_2$)
	Q('MATH-400').	($\mathcal{C}_3$)

The *IA* in response to the query $\mathcal{Q}_6$ is composed of the clause $\mathcal{C}_2$ and the fact $\mathcal{C}_3$. If $\mathcal{C}_2$ and $\mathcal{C}_3$ were added to the database and the query $\langle \leftarrow Q(A). \rangle$ asked, the same answers would result as do with query $\mathcal{Q}_6$.

Cholvy and Demolombe (1990, 1987) consider *IA*s under the sufficiency criterion (**II**). An *IAc* represents sufficient conditions for a query to be *true* if all ground substitutions of the *IA* are correct (ground) answers to the query. They focus on *IA*s that are *invariant*. An *IA* is *invariant* if it implies the query by the sufficiency criteria under *any* state of the database (Motro 1991). If an *IA* is based only on the *IDB* and *IC* (does not include any facts from the *EDB*) and would still remain an answer to the query regardless of any changes to the *EDB*, then the *IA* is an invariant answer to the query. The intensional answer to query $\mathcal{Q}_6$ above is not invariant because it includes extensional information, *Q('MATH- 400')*, derived from a fact in the *EDB*, namely *teaches(smith, 'MATH-400')*. Consider the following database and the query about who receives bonuses:

EDB: *started(michael, 1979).* *commended(carol).*
started(josé, 1985). *commended(chris).*
started(susan, 1987).

IDB: *bonus(P)* ← *started(P, D), today(N),*
elapsed(N, D, Y), Y ≥ 10. (C_4)
bonus(P) ← *commended(P).* (C_5)

Q: ← *bonus(P).* (Q_7)

A's: *"Everyone who has worked here for 10 years or more."* ($\mathcal{A}_1$)
"Everyone who has been commended." ($\mathcal{A}_2$)

The ground answers to query Q_7 are {*michael, carol, chris*}. However, these three do not receive bonuses for the same reason. Carol and Chris get bonuses because they were commended, $\mathcal{A}_2$, while Michael gets a bonus because he has been employed for more than ten years, $\mathcal{A}_1$. Answer $\mathcal{A}_1$ is a paraphrase of rule C_4, answer $\mathcal{A}_2$ of rule C_5. These are invariant answers that will always be true with respect to query Q_7 (until the *IDB* changes). These answers may be better in the sense that they explain the semantics behind who gets a bonus rather than just listing the people.

The *IA*s that Cholvy and Demolombe can find may be less direct than $\mathcal{A}_1$ and $\mathcal{A}_2$ above. This is the simplistic case in which each *IA* derives directly from a rule. Rules and *IC*s may interact to result in new *IA*s. The *IA*s may not be obvious, but they still offer correct, and often insightful, characterizations of answers to the query.

It is necessary to find a small finite set of *interesting IA*s from among the potentials for a given query. One criterion Cholvy and Demolombe use is to find only *IA*s that are not logically subsumed (a syntactic check) by any others. This will result in a finite set. They introduce other criteria as well. Their *IA*s are limited to a vocabulary of interest (predicates and constants) defined for a user. Spurious *IA*s that are not meaningful to the user will not be produced.

Motro (1989), Pirotte and Roelants (1989), and Pirotte, Roelants, and Zimanyi (1990) also consider intensional answers based on the residues as seen in Chakravarthy (1985) and Chakravarthy, Grant, and Minker's work (1986a,b) and Gal (1988) and Gal and Minker (1985, 1988). They use the residues to identify necessary conditions of answers (criterion **III**) and report these conditions along with the answers as a cooperative style. Like Gal, Pirotte, Roelants, and Zimanyi note that the residues from semantic compilation are knowledge in Motro's sense and are a type of intensional answer. Their work is subsumed by (Chakravarthy, 1985; Chakravarthy, et al., 1986a, b, 1990, Gal, 1988; Gal and Minker, 1985).

Another approach is taken by Shum and Muntz (1987) who consider *IA*s under the necessity criterion (**III**). Answers to queries which consist of exhaustive lists of values may be represented more succinctly by class descriptions. For example,

the query *"Who works from 9 to 5?"* might have {*tom, sue, ann, mary*} as its answer set. If these people constitute all the employees except for Carol and José, then $employee(X) \wedge X \neq \text{carol} \wedge X \neq josé$ is an *IA*. A simpler, alternative answer would be {*4/6 employees*} since the set constitutes four out of six employees in total in the database. Schum and Muntz's technique requires an explicit type taxonomy over the database domain. This considers the notion of intensional answers as *necessary* conditions on answers as well as sufficient conditions.

Say that Tom and Sue constitute all the secretaries and that Ann and Mary constitute all the technicians in the database above. Then the answer may be represented as {*secretaries, technicians*}. Corella (1984, 1989) describes how to obtain expressions that are equivalent to the extension of the query (criterion **I**) by finding a set of *types* (unary predicates in the first-order predicate calculus) that when unioned together cover all the answers and only the answers.

3.5. Generalizations

Another type of cooperative strategy is to rewrite a query to a generalized form. The scope of the query is extended so that more information can be gathered in the answers. One might seek and include information on related topics, using heuristics and past queries to induce which topics are of interest to a user. Alternatively, one might look for answers that are related to the original answers, but are not necessarily literal answers of the original query.

Cuppens and Demolombe (1988) give methods to rewrite the query so that variables are added to the query vector which carry relevant information for the user. For example, the query

$$\leftarrow travel(washington, toulouse).$$

might be modified so that its answer reports cost. They define a meta-level definition of a query that specifies the query in three parts: *entity, condition,* and *retrieved attributes*. Answers to queries provide values to the variables designated by the retrieved attributes. Methods are defined to extend the retrieved attributes according to heuristics about *topics of interest* to the user. All attributes from the original query along with any new added attributes appear in the rewritten query.

Cuppens and Demolombe (1988) also introduced the notion of loosening some of the "constraints" in a query to find answers close to those asked for. For instance, someone might ask

$$\leftarrow flight(No, \text{'Orly'}, \text{'Dulles'}, Time), Time \geq 17{:}00, Time \leq 21{:}00.$$

An answer could be returned *flight(delta714, 'Orly', 'Dulles', 21:05)* even though it does not strictly match the query. It is assumed the user might be interested in this "close" answer.

Wahlster, et al. (1983) allow for over-answering of yes/no questions when further questions are anticipated from the user on the same topic. Just as Janas' answers (1981), which identify the part of the query that fails, may save the system processing time by eliminating futile follow-up questions, these responses can eliminate the need for many queries to be asked. Their method provides the user with the information he or she probably needs without necessitating that the user engage in an exhaustive question/answer session. Their system employs domain knowledge to ascertain which types of follow-up questions are likely.

Q: *"Has a yellow car gone by?"*

The system adds that the user will want to know where it went by.

Q': *"Has a yellow car gone by? If so, where?"*

A: *"Yes, one went by on Hartungstreet."*

In the relational database community, Chu, Chen, and Lee (1990, 1992) have explored an *abstraction/refinement* method of providing related answers to the original query, pursuing this notion of close answers. A query is abstracted into a more general query that is then refined into a set of new queries to be evaluated against the database. The abstraction and refinement rely on the database having explicit hierarchies of the relations and of the terms in the domain. Chu, et al. (1990, 1992) define such a structure called the *type abstraction hierarchy*. A query rewrite is accomplished by replacing relations and terms from the query with corresponding relations and terms from higher in the hierarchy. The resulting query is considered more general than the original.

Along similar lines, we have introduced a cooperative method called *relaxation* for expanding deductive database and logic programming queries (Gaasterland, et al., 1991). The relaxation method expands the scope of a query by relaxing the logical constraints implicit in the query. Thus, the database may return answers related to the original query as well as the literal answers themselves.

travel(From, To)←*serves_area(A, From),*
serves_area(B, To), flight(A, B).* (C_6)

←*flight('Dulles', 'JFK').* (Q_8)

←*serves_area('Dullus', From),*
serves_area('JKF', To), travel(From, To). (Q_9)

Query Q_8 seeks a flight from Washington's Dulles airport to New York City's JFK. If no adequate flights can be found,[11] then the query can be relaxed to query Q_9 to look for other means of travel. The clause C_6 is market as a *reciprocal* clause (marked with an asterisk in the body), meaning it can be used in an abductive direction to relax a query to a more general query, for instance Q_8 to Q_9. (The market atom, *flight(A, B)* in C_6, unifies an atom in the query

and is replaced with the head of the clause and the other atoms in the body. See Gaasterland, et al., 1991.)

Our relaxation method is a general approach to seek additional answers to a query that may or may not be of direct interest to the user. We plan to extend this work to consider user models and user preferences to help determine when a query relaxation might be relevant. In the system *FLEX*, Motro proposes allowing the user to select directions of relaxation, and thus to indicate which relaxed answers may be of interest (Motro 1990). We also have considered meta-interpreter methods to provide users with choices of relaxed queries, allowing the user to navigate the database (Gaasterland, et al., 1991).

4. User goals and models

In query/answer systems, user interfaces, and human-machine interactions, much attention has been paid to classifying and characterizing users. Knowing more about a given user, a system can better attend to a person's questions. We do not attempt to survey fully the area user modeling. This is a large and rich field, with much outside the limited scope of this survey. In this section, we review the work that is relevant within cooperative answering.

In the previous section, the focus was on cooperative answering styles that are universal: they pertain to all users and questions in the same way. For instance, Grice's maxims should be adhered to in all cooperative dialogue. But, beyond these basics of cooperative response, different users have different interests, needs, and intentions. The better the system can detect these, the better the response and service it can provide the user. This requires the system to know of particular users and to have models of users. It requires the system to be able to detect interests, needs, and intentions in the dialogue with the user. The system must be capable of employing this information to tailor answers to best satisfy these criteria.

Three types of knowledge about a user are relevant to cooperative answering:

1. Interests and preferences
2. Needs
3. Goals and intent

Interests and preferences (1) direct the content of and type of answers that should be provided. Cuppens and Demolombe (1988) rewrite queries to include more information than would the original query that is of interest to the user. (Refer back to Section 3.5.) They claim extensibility of their work to incorporate user models. The models would identify *topics of interest* for different users and classes of users. The query rewrite mechanism would be directed to rewrite queries in order to include the relevant information of interest to a particular user, but directed to avoid explicitly including information not of interest.

Preferences (soft constraints) are met if possible, but are abandoned when they are at odds with the hard constraints of the query. If answers are provided sequentially (in a top-down manner as in *PROLOG*), then preferences can dictate the order in which answers are found and presented. Answers that meet the preferences *best* are presented first, while those meeting the preferences most poorly are saved for last.

Needs (**2**) may vary from user to user as well. A user may be well-versed or may be naive in the system's domain. The type and level of detail of answers will depend on the user's experience. The appropriateness of different potential responses can be evaluated with respect to the user's needs. Paris (1987, 1988) addresses the user's level of expertise in the content of the answer.

Q: *"What is a telephone?"*

System knows user is an engineer. Then

A_1: *(technical description of parts and how they work)*

System knows user is an eight-year old. Then

A_2: *(visual description of object and what it does)*

The two answers use different vocabulary and describe different properties of the telephone.

User constraints are introduced in (Gaasterland, Minker, and Rajasekar, 1990) and formalized in (Gaasterland, 1992). The notion of user constraints (*UC*s) is analogous to that of integrity constraints in deductive databases. A *UC* is of the same syntactic form as an *IC*, but does not have the same meaning; an *IC* must be logically consistent with the database, whereas a *UC* does not. A *UC* dictates what answers are acceptable to the user and filters out unacceptable answers. To a certain extent, the same mechanisms for finding residues from *IC*s in semantic query optimization can be used for *UC*s. Unlike residues from *IC*s, these *UC* residues will block search space that does contain answers, but only answers that are not of interest to the user anyway.

Goals and intent (**3**) do not vary inherently from user to user; rather, they vary from session to session and depend on what the user is attempting to accomplish at the time. Past dialogue, user models, and other factors can help a system to determine the probable goals and intents of the user. A number of researchers (Allen, 1987; Allen and Perrault, 1986; Joshi, et al., 1984; Pollack, 1983; Wahlsten, et al., 1983; Webber and Mays, 1983) try to determine user goals in order to choose cooperative information.

Of course, it is possible for the system to assume things incorrectly. Short of the ability to ask the user directly whey certain information has been requested (and the ability to comprehend the user's response), the system will be prone to mistakes. So, while accounting for users' goals and intent can be a powerful addition to a cooperative answering system, it can also be dangerous. The risk is when the system is wrong about what the user really wants, the answers it

gives can be more uncooperative than if the system did not consider the users' goals and intent at all. Special care must be taken to minimize the costs of such mistakes.

Allen and Perrault (1987, 1986) seek to detect user goals to help identify potential obstacles to the user meeting the goal. Allen's interest is in temporal issues and knowledge about time and how assumptions about time should effect responses to a user. His work conceptually extends Cuppens and Demolombe's idea (1988) in which user models dictate what information to add to the answers (via rewriting the query) by accounting for the user's current goals. For instance,

Q: *"When does the Windsor train leave?"*

A: *"At 4:00 p.m. (five minutes from now), at gate 7."*

The answer offers two cooperative pieces of information beyond the literal answer. First, it warns the user that the train is leaving in 5 minutes, an obstacle if the user is trying to catch the train. Second, it tells the user about the gate number, potentially important if this is not the usual gate or if it is somewhat distant from the user's current location.

Pollack (1983) detects how the questions users ask fit into their plans. Then answers that facilitate a user's plan will be given. The answer should, in essence, help to solve the user's problem. Suppose a user has typed a *cntl-Z* during a **vi** editing session and wants to *undo* the results. The user might ask:

Q: *"In the* **vi** *editor, how can I delete cntl-Z?"*

A: *"Cntl-Z has stopped your* **vi** *process.*
Type 'fg' to resume it."

Once the system realizes the goal, it can reinterpret the query and give a response that helps in achieving the goal.

Carberry (1988) emphasizes the dynamic construction of a model of the user's current task-related plan during the user/system dialogue. Such a model can be used to identify conflicts that arise between a user's questions and supposed plan. Misconceptions as considered in Section 3.3 arise from conflicts between the user's world view (as indicated by his or her queries) and the system's world knowledge. With a model of the user's task, possible further misconceptions can be identified when the user's queries and task at hand do not align. Care must be taken to try to determine when such an alignment is due to a misconception and when it is due to changes in the user's plan or to mistakes in the system's assumptions and modeling of the plan.

These ideas have been incorporated in the *IREPS* system (Intelligent REsPonse System), part of a research effort at the University of Delaware to provide a robust natural language interface for information systems. A component called *TRACK* infers the user's task-related plan. The heuristics and processing strategies

employed are domain-independent; only the domain-dependent plans and goals must be provided for a given application domain.

McCoy (1988) extends her work as discussed in Section 3.3 to consider user models and to respond to misconceptions that a given user may be known to hold. A model of what the user believes cues the system about misconceptions this user may have. The system has a set of basic, general strategies to resolve misconceptions of the user. Given a particular misconception, a suitable strategy is chosen to resolve it by adding appropriate statements to the response.

Quilici, Dyer, and Flowers (1988) also seek to recognize and respond to plan-oriented misconceptions. They consider advice-seeking dialogues between an advisor (an automated system) and a novice (the user). A cooperative response consists of explaining the beliefs of the system that conflict with the user's supposed beliefs in order to resolve the user's misconception. Similar to McCoy's explanation strategies (McCoy, 1988), they provide a taxonomy of domain-independent explanations as strategies for resolving different kinds of potential misconceptions.

Quilici, Dyer, and Flowers (1988) present a UNIX advisor based on their system for answering UNIX users' questions when they do not understand the results of their actions. This is based on a system called *AQUA* developed at the University California Los Angeles. The *AQUA* system infers users' beliefs, determines which are incorrect, and attempts to trace the cause of the beliefs. The system has an extended representation schema in order to store information about users' plans.

5. Cooperative answering at the University of Maryland

At the University of Maryland at College Park, we are engaged in building a cooperative answering facility for deductive databases. The system adds many of the cooperative behaviors and methods discussed previously to the deductive database query interface. Goals are that it

1. be a uniform system
    - defined in, and implemented through, logic
    - a uniform representation and support for all the cooperative methods
2. be portable and
    - a general approach for relational and deductive databases and for logic programs
    - domain-independent—applies to any relational or deductive database schema and state, and to any logic program
3. have a natural language interface
    - accept natural language queries
    - provide cohesive and coherent responses in natural language

Most of the cooperative methods we have considered in this paper have been implemented independently and employ different computational strategies. The need remains to integrate these methods and behaviors into a uniform system. All should rely on the same representations and the same computational strategies.

Our platform is deductive databases. Data is stored as the *EDB, knowledge* via the *IDB* and *IC*. The semantic query optimization technology of Chakravarthy, Grant, and Minker (1985, 1986a, b) provides a general strategy for finding conflicts and interactions between queries and *IC*s. This yields a uniform approach to handling misconceptions. Logic programming (we are using *PROLOG* for our implementation) offers us not just a uniform knowledge representation, but also a viable programming platform for handling the different cooperative methods in a general, uniform manner.

Deductive database and knowledge base systems often use an *interpreter* to read queries and to apply a proof procedure to find answers. For knowledge bases in *PROLOG*, this is usually just *PROLOG*'s interpreter. A *meta-interpreter* in such systems is used to supercede the *PROLOG* interpreter and can effect specialized behavior over the interpreter. A meta-interpreter reads clauses, queries, and *IC*s as data and can manipulate them to effect different control and query answering strategies. A meta-interpreter is often called a *shell* after the user-interactive shells of operating systems and interpreted environments. The following shell, *relax_solve/1*, is a simple *PROLOG* shell that enacts query relaxations as discussed in Section 3.5.

```
relax_solve(As)  ←  relaxing(As, Bs),
                    solve(Bs).
relaxing([A|As], [A|Bs])  ←  relaxing(As, Bs).
relaxing([A|As], Bs)  ←  clause(relax(A), Cs),
                         relaxing(Cs, Ds),
                         relaxing(As, Es),
                         append(Ds, Es, Bs).
relaxing([], []).
```

This shell is simplistic, but it illustrates how cooperative behavior can be achieved for logic programs. In reality, we would want relaxation to occur breadth-first. Also, this shell relaxes the query automatically once the original query's solutions have been exhausted. It would be better to provide the user with a menu of relaxation choices (Gaasterland, et al., 1991). Of course, relaxation is just one of the cooperative behaviors we want in our shell. We combine such meta-interpreters for the different cooperative behaviors into a single system.

Our cooperative answering system (Figure 3) is a comprehensive meta-interpreter for *PROLOG* deductive databases that effects many of the cooperative behaviors discussed. The meta-interpreter employs the *IC*s and rules of the database to produce cooperative answers. The system builds on the earlier

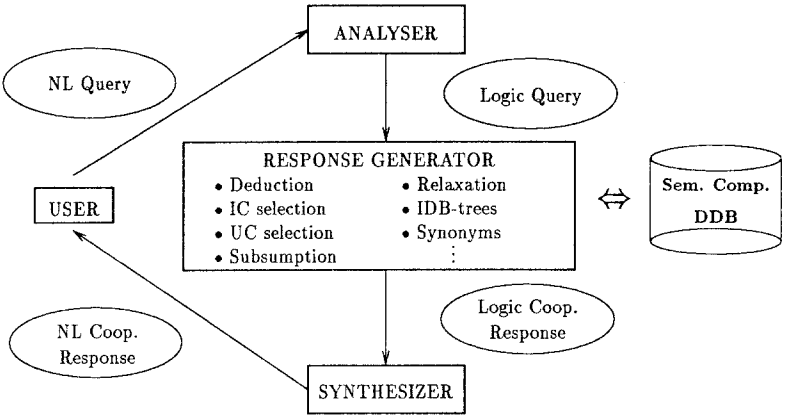


Fig. 3. Flowchart of COOPERATIVE ANSWERING SYSTEM at MARYLAND.

system of Gal (1988), Gal and Minker (1988), and Lobo and Minker (1988).

The cooperative answering system is portable. The methods and strategies employed for producing cooperative answers are domain-independent and no special tailoring is needed for a given database domain. The system is easily used with a relational database system since a simple deductive database interface can be implemented on top of any relational system. The shell can also provide cooperative response in more complex knowledge bases represented by logic programs.

The system supports limited natural language input for some domains. Some queries can be posed in natural language text. (Others must be posed in logic.) As discussed in Section 2.1, this becomes important when databases are too big or complex for a user to have a clear picture of the whole schema. The natural language translator builds a logical query. The interface has a dictionary to map natural language terms into the proper database predicates and constants.

The system can produce natural language responses. This is the more important direction. Cooperative responses can be cumbersome since they may incorporate knowledge as well as data from the database. When presented as logical formulas, they can be difficult to understand even for seasoned logic programmers. Answers can be large, full of notation, and connections between concepts may not be clear.

Natural language can alleviate many of these problems (Gaasterland, 1992). A linguistically motivated natural language back-end is developed to provide cooperative natural language responses to queries. Anaphora helps remove notational redundancies and makes the answer text much more readable and, hence, understandable. Coordination collapses parallel structures into more concise sentences. Cohesion is a global property that assures the answer sentences

form a cohesive response in which the components are related in clear, obvious ways and each contributes towards the answer. We have a response generator that uses these linguistic tools as well as others that produces good natural language response.

The cooperative answering shell handles the following types of cooperative response:

- natural language interface (Gaasterland, 1992; Gal and Minker, 1988a, b, 1985)
  - handles natural language queries
  - generates natural language responses
- false presuppositions (see Section 3.2)
  - finding minimal failed subquery
  - maximal failed relaxation
- misconceptions (constraint violations) (see Section 3.3)
  - *EDB* search
  - violations with respect to previous answers
- explanations of derivation paths (proofs)
- positive cooperative information
- employs heuristics to select best cooperative information (Gal 1988)
- detect redundancy in the query
- use *UC*s to filter answers (Gaasterland, 1992; Gaasterland, et al., 1990)
- relaxation (menu-driven) (Gaasterland, et al., 1991)
- query decomposition

Any violation of an *IC* by a query indicates a possible misconception on the user's behalf. The most general *IC* necessitating that the query fails is found. These *IC* violations are detected and reported to the user. Sometimes a query may not conflict with any one *IC* from the $\mathcal{IC}$ set in particular, but the query having an answer would be in violation of the *IDB* and $\mathcal{IC}$ together. In this case, the query will conflict with a *derived* constraint, an *IC* that can be deduced from the *IDB* and $\mathcal{IC}$ and is thus *true* over the database. Then, the derived constraint is presented, along with its derivation. Different subgoals of the query may fail for different causes (each conflicts with a different *IC*). This requires that the failure of each subgoal be explained, and an explanation of the proof tree is also required to explain how these subgoals arose.

Sometimes *IC*s may interact without indicating a cause for failure. Instead, such *IC*s offer semantic information about the query, other conditions that must be true for there to be an answer to the query. At times, certain of this information may be pertinent to include in the answer.

A query may be overspecified. Some subquery may find all the answers that the query would find. Such a case may also indicate a misconception. This situation can also be detected by the shell when *IC*s show a query and a subquery to be semantically equivalent. An example of this is seen in Section 3.3. Query $\mathcal{Q}_4$ is equivalent to the subquery $\mathcal{Q}_5$ due to $\mathcal{IC}_2$.

If there are no misconceptions but the query still fails, false presuppositions are present in the query. The shell informs a user of false presuppositions. The minimal failed subqueries are found as in Janas' method. (Refer to Section 3.2.) Once a failed subquery is identified, this query is relaxed to find the most general query that still fails. This failure is reported to the user.

There may at times be too much potential cooperative information to provide to the user. To provide all of it would violate the maxim of quantity and overwhelm the user. Choices must be made as to which information is to be included. For instance, a query failure due to a misconception is more pertinent than failure due to a false presupposition. A query that fails due to a misconception *must* fail, whereas a query that fails due to a false presupposition happens to fail given this state of the database, but it would not necessarily fail in some other state. Gal (1988) presents a number of heuristics for deciding which cooperative information to include, abiding the cooperative maxims as best as possible.

*UC*s provide a means for modeling the user in the system (Gaasterland, 1992; Gaasterland, et al., 1990). They are used to eliminate answers for a query that would violate the *UC*s of the particular user. Applicable *UC*s are attached to the query in an analogous manner to semantic query optimization for *IC* s. This effects a query rewrite, but unlike semantic query optimization it changes the meaning of the query in such a way that the rewrite only find the answers that do not violate the *UC*s.

The system provides relaxed queries to the user once the original query has been exhausted. This way, if the user did not find the answers from the original query to be adequate, he or she may continue to navigate the database by selecting related (relaxed) queries to find related answers. The shell handles relaxation in an efficient manner. Also, repeat answers to the relaxed query are suppressed (avoided in the computation) since the user has already seen (and rejected) them (Gaasterland et al., 1991).

A query is *disjoint* whenever it can be subdivided into two or more independent subqueries. The shell detects this and decomposes the query in such cases. This, and other such syntactic checks and optimizations, help catch users' mistakes and help to reduce the cost and overhead of the other cooperative techniques.

In our continuing research, we plan to

- more fully integrate the mechanisms supporting the cooperative behaviors
    - develop clear semantic support (in deductive databases) for cooperative answering
    - implement more efficient mechanisms for the different cooperative behaviors
- provide a better user interface and
    - provide a sophisticated, integrated natural language response generator
    - develop further the explanation facilities
- add more cooperative behaviors and facilities.
    - allow for useful *IA*s

- incorporate Cuppens and Demolombe's style of query rewrites
- further develop the theory of *UC*s and user models

Work on the cooperative answering system at Maryland is an ongoing project. We shall continue to extend its functionality, improve its performance, and to research new cooperative techniques. We are also engaged in foundational research to map out the requisite knowledge and semantics needed to support cooperative behavior at large, the requisite representations, and the requisite computational strategies for yielding cooperative response. Although the many cooperative behaviors discussed in this paper may seem at first disparate, we are seeking a general classification for cooperative response and the universal representations and strategies that will be able to provide them.

6. Summary

In this paper we have given a brief overview of the field of cooperative answering. Cooperative response is crucial to the success of automated query/answer systems. History has shown that providing *correct* answers to users is simply not enough. Ambiguities, misconceptions, and irrelevances plague such systems. As information systems play a more vital role in society, it is necessary that these problems be addressed.

Fortunately, a good deal of research has already been directed toward these ends, and exciting, promising work is underway. We have reviewed a number of disparate cooperative behaviors. Most of these cooperative techniques have been developed, but each in a different system, reliant on a different representation, and effected by different computational means. While much work remains to explore and develop new cooperative techniques, the current collection of cooperative response work needs to be pulled onto common ground.

Deductive databases offer a suitable platform for the development of uniform cooperative systems. Deductive databases provide a powerful, uniform representation via logic, semantics, and a computational means for effecting efficient cooperative response. Much high quality work has been done in cooperative answering within the deductive database paradigm due to this natural fit. Both the fields of cooperative answering and deductive databases has benefited as a result. The cooperative answering system we have developed, based on logic, is indicative that it is a unifying framework for such systems.

Acknowledgment

This work was supported by the Air Force Office of Scientific Research under grant number AFSOSR-91-0350 and by the National Science Foundation under NSF grant number IRI-89-16059.

Notes

*Invited Paper

1. *Data* and *knowledge* are distinguished. See Section 3.4 for a discussion.
2. We shall write constants beginning in lowercase or quoted, and write variables beginning in uppercase.
3. This has the same meaning as *static* integrity constraints in relational databases, a statement that must be *true* of the state of the database. We do not consider *dynamic* integrity constraints, which must be preserved across a database transaction.
4. We use the symbol "$\subset$" also commonly used for logical *if*, instead of "$\leftarrow$" within integrity constraints to distinguish them typographically from queries.
5. We write classical negation with the symbol "$\neg$."
6. This book is incorrectly referenced as *Mutual Beliefs* in many papers.
7. We have added this one to the example.
8. Each subquery in Figure 2 *logically subsumes* each query that it points to. A formula $\mathcal{F}$ *logical subsumes* $\mathcal{G}$ iff $\models \mathcal{F} \rightarrow \mathcal{G}$.
9. He does not define *connected* in (Janas, 1981) in quite this way, but to the same effect.
10. This will not occur in the case of deductive databases, which contain no function symbols, but can occur in the case of logic programs, which do allow function symbols.
11. A real query would specify much more, such as time, date, and cost.

References

Allen, J. (1987). *Natural Language Understanding*. Benjamin/Cummings. Menlo Park, CA.

Allen, J.F. and Perrault, C.R. (1986). Analyzing Intention in Utterances. In B.J. Grosz, K.S. Jones, B.L. Weber (Eds.), *Readings in Natural Language Processing* (pp. 441–458). Morgan Kaufmann. Los Altos, CA.

Carberry, S. (1988). Modeling the User's Plans and Goals. In A. Kobsa and W. Wahlster (1988) (pp. 64–78). Special Issue on User Modeling.

Chakravarthy, U. (1985). *Semantic Query Optimization in Deductive Databases*. Ph.D. thesis, University of Maryland, Department of Computer Science, College Park.

Chakravarthy, U., Grant, J., and Minker, J. (1986a). Semantic query optimization: Additional constraints and control strategies. In L. Kerschberg (Ed.) *Proc. Expert Database Systems* (pp. 259–269). Charleston.

Chakravarthy, U., Grant, J., and Minker, J. (1986b). Foundations of semantic query optimization for deductive databases. In J. Minker (Ed.), *Proc. Workshop on Foundations of Deductive Databases and Logic Programming* (pp. 67–101). Washington, D.C.

Chakravarthy U., Grant, J., and Minker, J., (1990). Logic Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2); 162–207.

Cholvy, L. (1990). Answering Queries Addressed to a Rule Base. *Revue d'intelligence artificielle*, 4(1), 79–98.

Cholvy, L. and Demolombe, R. (1987). Querying a Rule Base. In L. Kershberg, (Ed.), *Expert Database Systems*. Tysons Corner, VA.

Chomicki, J. and Imieliński, T. (1989). *Relational Specifications of Infinite Query Answers*. Technical Report CS-TR-2177, Department of Computer Science, University of Maryland, College Park, MD.

Chu, W.W., Chen, Q., and Lee, R. (1991). Cooperative Query Answering via Type Abstraction Hierarchy. In S. M. Deen (Ed.) *Cooperating Knowledge Based System 1990* (pp. 271–290). University of Keele, U.K.: Springer-Verlag.

Chu, W.W., Chen, Q., and Lee, R. (1992). A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*. To appear.

CODASYL (1971). *CODASYL Data Base Task Group April 71 Report*. ACM, New York.

Colmerauer, A. and Pique, J. (1981). About Natural Logic. In H. Gallaire, et al. (1981), (pp. 343–365).

Corella, F. (1984). Semantic Retrieval and Levels of Abstraction. In L. Kerschberg (Ed.) *Proceedings of the First International Workshop on Expert Database Systems* Vol. II (pp. 397–420).

Corella, F. (1989). *Mechanizing Set Theory*. Ph.D. thesis, Corpus Christi College, and RC 14706, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York.

Cuppens, F. and Demolombe, R. (1988). Cooperative Answering: A Methodology to Provide Intelligent Access to Databases. In L. Kershberg (Ed.) *Proceedings of the Second International Conference on Expert Database Systems* (pp. 333–353). George Mason University.

Gaasterland, T. (1992). *Cooperative Answers for Database Queries*. Ph. D. thesis, University of Maryland, Department of Computer Science, College Park.

Gaasterland, T. (1992). Cooperative Explanation in Deductive Databases. In A. Quilici (Ed.) *Working Notes: Symposium on Producing Cooperative Explanations*. AAAI Spring Symposium Series, Palo Alto, CA: Stanford University.

Gaasterland, T., Godfrey, P., and Minker, J. (1991). Relaxation as a Platform of Cooperative Answering. In T. Imielinski (Ed.) *Proceedings of the First International Workshop on Nonstandard Queries and Answers* (pp. 101–120). Vol. 2, Toulouse, France.

Gaasterland, T., Minker, J., and Rajasekar, A. (1990). Deductive database systems and knowledge base systems. *Proc. VIA 90*. Barcelona, Spain.

Gal, A. (1988). *Cooperative Responses in Deductive Databases*. Ph.D. thesis, Department of Computer Science, University of Maryland, College Park.

Gal, A. and Minker, J. (1988). Informative and Cooperative Answers in Databases Using Integrity Constraints. In V. Dahl and P. Saint-Dizier (Eds.) *Natural Language Understanding and Logic Programming* (pp. 277–300). Amsterdam: North, Holland.

Gal, A. and Minker, J., (1985). A natural language database interface that provides cooperative answers. *Proc. Second Conf. Artif. Intell. Appl.*

Gallaire, H. and Minker, J. (1978). (Eds.) *Logic and Databases*. New York: Plenum Press.

Gallaire, H., Minker, J., and Nicolas, J.M. (1981). (Eds.) *Advances in Database Theory*, Vol.1. New York: Plenum Press.

Gallaire, H., Minker, J., and Nicolas, J.M. (1984). Logic and Databases: A Deductive Approach. *ACM Computing surveys, 16*(2), 153–185.

Grice, H. (1975). Logic and Conversation. In P. Cole and J. Morgan (Eds.) *Syntax and Semantics*. New York: Academic Press.

Hirschberg, J. (1983). *Scalar Quantity Implicature: A Strategy for Processing Scalar Utterances*. Technical Report MS-CIS-83-10, Department of Computer and Information Science, the Moore School, the University of Pennsylvania, Philadelphia, PA.

Imielinski, T. (1988). Intelligent Query Answering in Rule Based Systems. In J. Minker (Ed.) *Foundations of Deductive Databases and Logic Programming*. Washington, D.C.: Morgan Kaufman.

Janas, J.M. (1981). On the Feasibility of Informative Answers. In H. Gallaire, et. al (1981) (pp. 397–414).

Joshi, A. (1982). Mutual Beliefs in Question Answering Systems. In Smith (1982).

Joshi, A., Webber, B., and Sag, I. (1981) (Eds.) *Elements of Discourse Understanding*. Cambridge: Cambridge University Press.

Joshi, A.K., Webber, B.L., and Weischedel, R.M. (1984). Living up to expectations: Computing expert responses. *Proc. Nat. Conf. Artif. Intell.* (pp. 169–175). University of Texas at Austin: The American Association for Artificial Intelligence.

Kaplan, S.J. (1981). Appropriate Responses to Inappropriate Questions. In A. Joshi, et al. (1981). (pp. 127–144).

Kaplan, S.J. (1982). Cooperative Responses from a Portable Natural Language Query System. *Artificial Intelligence, 19*(2), 165–187.

Kobsa, A. and Wahlster, W. (1988). (Eds.) *Computational Linguistics, 14*(3). MIT Press for the Association of Computational Linguistics, Special Issue on User Modeling.

Lehnert, W. (1981). A Computational Theory of Human Question Answering. In A. Joshi, et al. (1981). (pp. 145–176).

Lloyd, J. (1987). *Foundations of Logic Programming*, 2nd ed. New York: Springer-Verlag.

Lobo, J. and Minker, J. (1988). A Metaprogramming Approach to Semantically Optimize Queries in Deductive Databases. In L. Kerschberg (Ed.) *Proceedings of The Second International Conference on Expert Database Systems* (pp. 387–420). Tysons Corner, VA.

Mays, E. (1980). Correcting misconceptions about database structure. *Proc. CSCSI '80.*

McCoy, K. (1984). Correcting object-related misconceptions. *Proc. COLING10, Stanfrod, CA: Stanford University.*

McCoy, K. (1988). Reasoning on a Highlighted User Model to Respond to Misconceptions. In A. Kobsa and W. Wahlster (1988). (pp. 64–78). Special Issue on User Modeling.

McKeown, K. (1982). *Generating Natural Language Text in Response to Questions about Database Queries.* Ph.D. thesis, University of Pennsylvania.

Minker J. (1988). *Foundations of Deductive Databases and Logic Programming.* Los Altos, CA: Morgan Kaufmann.

Motro, A. (1986). Extending the relational model to support goal queries. *Proc. First Int. Workshop Expert Database Systems* (pp. 129–150). Benjamin/Cummings.

Motro, A. (1989). Using constraints to provide intensional answers to relational queries. *Proc. Fifteenth Int. Conf. Very Large Data Bases.*

Motro, A. (1991). Responding with knowledge to database queries (tutorial and survey) Presented *Proc. First Int. Workshop Nonstandard Queries and Answers.* Toulouse, France.

Motro, A. (1990). FLEX: A Tolerant and Cooperative User Interface to Database. *IEEE Transactions on Knowledge and Data Engineering, 2*(2), 231–245.

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

Paris, C. (1987). Combining discourse strategies to generate descriptions to users along a naïve/expert spectrum. *Proc. IJCAI* (pp. 626–632). Milan, Italy.

Paris, C.L. (1988). Tailoring Object Descriptions to a User's Level of Expertise. In A. Kobsa (1988). (pp. 64–78). Special Issue on User Modeling.

Pirotte, A. and Roelants, D. (1989). Constraints for improving the generation of intensional answers in deductive databases. *Proc. 5th IEEE Int. Conf. Data Engineering.*

Pirotte, A., Roelants, D., and Zimanyi, E. (1990). Controlled Generation of Intensional Answers. *IEEE Transactions on Knowledge and Data Engineering.*

Pollack, M.E. (1983). *Generating Expert Answers through Goal Inference.* Technical Report, Stanford, CA: SRI International.

Pollack, M.E., Hirschberg, J., and Webber, B. (1982). User participation in the reasoning processes of expert systems. *Proc. American Assoc. Artif. Intell.*

Quilici, A., Dyer, M., and Flowers, M. (1988). Recognizing and Responding to Plan-Oriented Misconceptions. In A. Kobsa (1988) (pp. 38–51). Special Issue on User Modeling.

Schank, R.C. (1975). *Conceptual Information Processing.* Amsterdam: North-Holland.

Shepherdson, J. (1984). Negation as Finite Failure: A Comparison of Clark's Completed Database and Reiter's Closed World Assumption. *J. Logic Programming, 1*(15), 51–79.

Shum, C. and Muntz, R. (1987). Implicit Representation for Extensional Answers. In L. Kershberg (Ed.) *Expert Database System*. Tysons Corner, VA.
Smith, N. (1982). (Ed.) *Mutual Knowledge*. New York: Academic Press.
Ullman, J.D. (1988). *Principles of Database and Knowledge-Base Systems*, Vol. 1. Rockville, MD: Computer Science Press.
Wahlster, W., Marburger, H., Jameson, A., and Busemann, S. (1983). Over-answering yes-no questions: Extended responses in a NL interface to a vision system. *Proc. IJCAI 1983.* Karlsruhe, West Germany.
Webber, B.L. (1981). Discourse Model synthesis: Preliminaries to Reference. In A. Joshi, et al. (1981) (pp. 145–176).
Webber, B.L. and Mays, E. (1983). Varieties of user misconceptions: Detection and correction. *Proc. Eighth Int. Conf. Artif. Intell.* (pp. 650-652). Karlsruhe, Germany.