“Erotetic Logic as a Specification Language for Database Queries,” was published as the thesis in support of my Master’s degree in Computer Science at Kansas State University in 1987, and was delivered as a talk before the Naval Research Laboratory (Washington, DC) later that year.

**Introduction: An Overview of Fourth-Generation Logics**

In fairly recent times, logicians have developed a profusion of non-classical logics:

many-valued, modal, temporal, fuzzy and so on. Some of these logics, which I shall call "fourth- generation logics," have already been developed and exploited by computer scientists. Temporal logic has been used in AI and for the formal specification and verification of concurrent programs. Many-valued logic has been used in circuit design and fuzzy logic in AI. In this paper I would like to discuss an area of fourth generation logic, erotetic logic, which may be of use to computer scientists working AI and database theory.

I have coined the term "fourth generation logics" for two reasons. First, while the more customary term[[1]](#endnote-1) for these logics is "deviant logics," that phrase makes it sound as if people who do research on such logics are somehow strange or weird. Second, the phrase "fourth- generation" is at least roughly historically accurate.

We can distinguish generations of logic, in part on the basis of time of origin. Following Bartley (1977) and Jorgensen (1931), we can view the first generation of logic as beginning with Aristotle in the fourth century B.C. First-generation logic (also called "traditional," Aristotelian," or "syllogistic") logic, focused upon reasoning in natural language. Logicians were concerned to formulate rules for casting propositions into categorical forms and for assessing arguments ("syllogisms") built up out of those categorical propositions.

The second generation of logic, Boolean or algebraic logic, dates from the work of George Boole (1847) and the work of Auguste DeMorgan, John Venn, Lewis Carroll, and others later in the nineteenth century. Second-generation logicians kept the earlier concern with natural language argumentation, but extended logic to cover a wider class of arguments (ones involving relational propositions). More importantly, the algebraic logicians subtly shifted the focus of research to the application of mathematics to logic. Graphical methods (Venn diagrams, Euler diagrams, Carroll diagrams, and so on) for assessing syllogisms were developed, and methods for expressing propositions algebraically were devised.

Thus the second-generation logicians greatly expanded on the scope of analysis beyond traditional syllogistic, and greatly increased the formalization of logic. They also were concerned with devising techniques for *drawing* relevant inferences from premises, i.e., with *solving* problems or *discovering* solutions, as opposed to merely verifying the correctness of already formulated proofs. This concern shows up in the exercises found in second generation logic texts (such as Lewis Carroll's *Game of Logic*): puzzles that call upon the student to determine what conclusion about some situation can be derived from the given information. (Vestiges of the second-generation concern over logical puzzles are still found in some elementary logic texts, and several modern logicians—most notably E.R. Emmet and Raymond Smullyan—have written books of such logic puzzles.)

Third-generation logic is classical First Order Logic (“FOL”). While we might date the birth of FOL with the publication of Frege's *Begriffschrift* in 1879, it is more accurate to date the third generation in logic to the first decade of this century, when Russell brought Frege's work to public notice. Eminent logicians such as Peirce, Whitehead, Russell, Godel, Church, Gentzen and many others rapidly developed FOL, with many major results in place by the 1930's. Third- generation logic is concerned much less with argumentation in natural language than with the foundations of mathematics and proof theory. Hence the focus of research was on consistency proofs, axiomatizations, decision procedures, and so on. Indeed, many third generation logicians were avowed **logicists**, i.e., they wanted to reduce all of mathematics to logic and set theory.

Fourth-generation logic is harder to date. We might date it lat in the nineteenth century, with Peirce's work on many-valued logic. It would be more accurate, however, to fix the date in the 1920's and 1930's with the work of Lukasiewitz on three-valued logic and the work of C.I. Lewis on strict entailment (modal logic). But in terms of purpose, the difference between third and fourth generation is clear. Fourth-generation logicians are much less concerned with the foundations of mathematics, and are once again concerned with reasoning and knowledge expressed in natural language. Typically, the fourth-generation logician feels that FOL is inadequate to explicate some interesting feature of rational discourse, and he seeks to find some modification of or replacement for FOL. The fourth-generation logician has the same goal of the first- and second-generation logicians, but wants to use *some* formal apparatus in addition to (or in lieu of) FOL.

Most logicians would concede that FOL does not sufficiently explicate all natural language reasoning. (Indeed, some logicians have even argued that FOL should *not* be taught as "logic" at all—see, for example Shoenfield (1967) pp. 447-456.) This is perhaps not surprising: FOL explicates mathematical reasoning well, but in mathematics there is no cause and effect, before and after, or questioner and respondent.

In any event, in the last forty years there has been a proliferation of fourth generation logics. Even classifying them is controversial. Some, such as Turner (1985) and Haack (1978), have divided fourth-generation logics into those which seek to *extend* FOL and those which seek to *replace* it. For example, standard modal logic extends FOL by adding modal operators, but the theorems of FOL remain valid. On the other hand, many-valued and intuitionist logics replace FOL in the sense that various theorems of FOL (e.g., the Law of Excluded Middle) fail in those logics.

I find that distinction unhelpful. To begin with, modal logics *are* many-valued in that they add a third truth value (necessary truth) to the classical values (true and false). More generally, *logics* don't seek to do anything: *logicians* do. And most fourth generation logicians don't seek to replace FOL; like scientists in other fields, they view conservativeness as a virtue in a theory. That is, faced with two new proposed theories that aim to succeed where an established theory failed, scientists will naturally choose the one that least conflicts with the established theory (all this being equal). A fourth-generation logician doesn't seek to replace FOL—he may be driven to it, but he doesn't seek it.

It may be suggested that I have misconstrued the proposed division. The point is (it might be urged) that some alternate logics accept as theorems the theorems of FOL, and some do not. But this seems to be an unhelpful point. Many of those logics in which some classical theorems fail have analogues that replace the failed theorems--for example, while the Law of Excluded Middle fails in a three-valued logic, the Law of Excluded Fourth may still hold.

I suggest that a more interesting division of fourth-generation logic is between **assertion** (or “assertoric”) logics (those that concern logical connections between statements) and **non- assertion** logics (those concerned with questions and answers, commands, dialogic structure and such like). Some of the major types of fourth-generation logics as listed below.

FOURTH GENERATION LOGICS

|

| Non-classical Assertion Logics

| | many-valued (including Quantum logics and intuitionist logic)

| | modal, temporal and causal logics

| | entailment logics

| | fuzzy and probabilistic logics

| | other (epistemic logics, free logic, etc.)

|

| Non-assertoric Logics

 | question logics

 | imperative logics

 | dialog logics

Several references are worth noting. Haack (1978) has a good survey of fourth generation logics and the issues surrounding them from the philosopher's point of view, while Turner (1985) has a survey of temporal, many-valued and fuzzy logic from the AI point of view. The classic survey of modal logic is Hughes and Cresswell (1968)—updated by Hughes and Cresswell (1984). A fine survey of many-valued logic is Rescher (1969) and of temporal logic is Rescher and Urquhart (1971). AI researchers are currently exploring the potential of relevance logic, and the Bible of such logics is Anderson and Belnap (1975).[[2]](#endnote-2) 1

 A fourth-generation logic that is unfamiliar to many computer scientists is erotetic logic. Roughly put, **erotetic logic** is the logic of questions. More precisely, if standard logic (assertion logic) involves explicating statements and the relations between statements, erotetic logic is the extension of assertion logic to explicate questions and the relations between questions and answers. By "explicate questions" I mean reveal the structure of questions in a way analogous to the way assertion logic explicates the structure of statements. For example, in standard FOL the two relational sentences "John loves Mary" and "Mary is loved by John" are symbolized the same way ("Loves (John, Mary)") because while the surface structures of the English declaratives differ, the semantic structures of the statements are the same.

 Similarly, any adequate erotetic logic should symbolize

(1a) What are the good restaurants in Topeka?

(1b) Which Topeka restaurants are good?

identically, because while the surface structures of the English interrogatives differ, the semantic structures of the questions are the same.

Again, just as any assertion logic categorizes statements (as being conditionals, negations, existentially quantified or whatever), any adequate erotetic logic should be able to categorize questions, and reveal the differences between:

1. Which restaurants in Topeka are good?
2. What are some good restaurants in Topeka?
3. What is a good restaurant in Topeka?
4. Why are there no good restaurants in Topeka?
5. Why do you say that there are no good restaurants in Topeka?
6. Is “The Loft” a good Topeka restaurant?
7. Where is “The Loft” located?

By "explicate the relation between questions and answers" I mean the formulation of rules governing the acceptability of a given answer to a given question. For example, question

(1) admits the answer:

The Loft and the Amarillo Grill are the only good restaurants in Topeka.

while none of the others do. Crucial here is the notion of a **direct** answer, an answer that supplies the requested information, but no more. That is why "The Loft and the Amarillo Grill are the only good restaurants in Topeka" is not an acceptable answer to (3), because (3) only requests one example of a good restaurant in Topeka, not a complete list. Thus to construct an adequate erotetic logic we must do three things. First, we must decide upon a typology of questions. Second, we must develop a formal system—presumably an extension of some formal assertion logic—that will allow us to symbolize questions. Third, we must formulate a set of admission rules, i.e. rules that specify the syntactic form of an acceptable answer to a given type of question.

Developing a typology of question requires considerable work, mainly of a philosophic and linguistic nature. There is a large body of literature on this topic (see the bibliography to Belnap and Steele (1976)). To simplify the issues treated in this paper, we will adopt the following typology:

QUESTIONS

 | Simple

 | | whether

 | | which

 | | why

 | | how

 | \_Compound

**Whether-questions**: These questions present their subjects (their range of alternatives) explicitly and request the respondent to select from among them. Example:

* 1. Is Fred Rich?
	2. Which should Fred take: FORTRAN, COBOL, C++ or BASIC?
1. presents the alternatives:

 Fred is rich.

 Fred is not rich.

while (9) presents the alternatives:

 Fred should take FORTRAN.

 Fred should take COBOL.

 Fred should take C++.

 Fred should take BASIC.

**Which-questions**: These questions present their subjects by means of statement functions, and request the respondent to select the data satisfying those functions. Examples:

* 1. Which cars are expensive?
	2. Who will come to the party?
	3. What are Fred's office hours?
1. presents the subject by means of the proposition function "x is a car and x is expensive,"
2. by "x is a person and x will come to the party,” and (12) by "x is an office hour of Fred."

**Why-questions**: These questions are of two sorts. The first sort of why-question is a request of the respondent to explain an acknowledged fact. Examples:

* 1. What is the cause of the oversupply of Ph.D.'s?
	2. Why do dogs bark?

The second sort of why-question is a request of the respondent to prove a point that the questioner doubts. Examples:

* 1. Why do you say that Fred is unhappy?
	2. What are his reasons for buying that stock?

The topic of why-questions is important in philosophy of science and will probably become important in AI; however, it is not relevant to this paper.

**How-questions**: These questions also are of two types. The first type of how-question requests from the respondent a set of instructions. Examples:

* 1. How do I get to Oklahoma City?
	2. How can we buy a new car?

The second sort of how-question is a request for the respondent to impart a skill. Examples:

* 1. Can you show me the fox-trot?
	2. Do you know how to get this bottle-cap off?

Again, while the topic of how-questions is interesting and potentially relevant to AI, it is not relevant here.

**Compound questions**: Questions, like statements, can be compound. Examples:

* 1. Who is that masked man, and what does he want?
	2. Where and when shall meet?

Questions can be compounded with statements:

* 1. If Fred wants to buy a car, where will he get the money?

In this paper we will not pursue the topic of compound questions.[[3]](#endnote-3) Questions that are truth-functionally compounded together we can handle in an obvious way. For example, an acceptable answer to (22) would simply be a conjunction of acceptable answers to "Where will we get together?" and “When will we get together?" Moreover, most cases in which statements are compounded with questions seem to be cases in which the statement is a guard against presupposition failure. For example, we would ask (23) rather than the simple question

* 1. Where will Fred get the money to buy the care he wants?

because (24) presupposes that Fred wants to buy a car, and we may not be sure that he does. But presupposition failure will be accounted for in our treatment (below) of simple questions.

In short, the formal apparatus we develop will be geared toward which-and whether- questions, since they seem to be the most crucial for database theory (as opposed to AI or philosophy of science).

1. **A Formal Erotetic System**

Having picked out the types of questions in which we are interested, we now construct a formal system for putting those questions[[4]](#endnote-4). Since a formal erotetic language is an extension of an assertoric language, we need first to lay out the grammar of that underlying logic. The language we will take to be FOL. The vocabulary (or "alphabet") consists of:[[5]](#endnote-5)

* + Infinitely many individual variables: x, y, z, x1, y1, z1,...
	+ Individual constants: a, b, c, …
	+ Predicates of any arity: A, B, C, ... (The number of places of the predicate is indicated by the number of individual variables and constants that follows it).
	+ Punctuation symbols: (,), [,], {,}
	+ Logical constants: → (if then), & (and), v (or), (not), ↔ (if and only if), (universal quantifier), (existential quantifier)

In philosophic literature, predicates and constants are kept to one letter. In computer science literature, predicates are often whole words (e.g. "Restaurant (x)"). The difference is purely one of style, but for readability we prefer the computer science style.

The well-formed formulas are characterized as follows. (Here I will use bold-face font to indicate meta-linguistic variables). We say that variables and constants are **terms**. We say that an n-ary predicate followed by n terms is an **atomic formula P(t1,...,tn**). If all the ti's are constants (i.e., individual names), then **P(t1,...,tn)** is a **ground** atomic formula (a "particular statement"). Then the set of well-formed formulas (wffs) is the set defined by the rules:

* + Any atomic formula ‘**P**’ is a wff
	+ If **P** is a wff, then so is ‘**P**’
	+ If **P** and **Q** are wffs, then so are ‘**P** v **Q**’,’ **P** **Q**’, ‘**P** **Q**’ and ‘**P** & **Q**’
	+ If **x** is a variable and **P** a wff, the so are ‘(**x**)**P**’ and ‘(**x**)**P**’No other expressions are wffs.

We don't need a full-blown first-order language to formulate a database. We say that a first-order language is a **relational language** if and only if it has at least one but most finitely many constants, it has only finitely many predicates, has equality (denoted by 'x = y') as a predicate, and has a set of "types." A **type** is a unary predicate **T** (a **simple** type) or Boolean combinations of simple types. (Types allow us to capture the notion of the "domain" of a given relation).[[6]](#endnote-6)

When types are defined we allow ourselves the luxury of type-restricted quantifiers. We will use

(**x**/**Tx**) **Px**

to mean

(**x**)(**Tx** **Px**)

and

(**x**/**Tx**) **Px**

to mean

(**x**)(**Tx** & **Px**)

In a standard way we can interpret the language **L**. An **interpretation M** consists of a domain **D** of individuals over which the variables range, a mapping **F** of the constants of the language onto individuals of **D**, and a mapping **E** of the predicates of the language onto sets of tuples of individuals of the domain. We can then characterize truth or satisifiablity under **M** (written |=**M**) as follows. For a given mapping **V** of variables of **L** into **D**, define another mapping **R** as follows:

**R**(**c**) = **F**(**c**) for every constant **c** of **L R**(**x**) = **V**(**x**) for every variable **x** of **L**

Then define |=**R,M** as follows:

 1) |=**R,M P(t1,...,tn)** iff (**R(t1),...,R(tn)**)**E(P**) for every atomic formula of **L**.

 2) |=**R,M** (**A1** & **A2**) iff |=**R,MA1** and |= **R,MA2**.

1. |=**R,M** (**A1** v **A2**) iff |=**R,MA1** or |=**R,MA2**.
2. |=**R,M** **A** iff not |=**R,MA**.

5) |=**R,M** (**A1** **A2**) iff |=**R,M** (A1 v A2).

6) |=**R,M** (A1 A2 iff |=**R,M** [(A1A2) & (A2 A1)].

7) |=**R,M** (**x**)**A** iff for all **d** **D**, |=**R\*MA** where **R**\* is identical to **R** except for mapping

**x** to **d**.

8) |=**R,M** (**x**)**A** iff |=R,M (**x**) **A**

Then we can define |=**M A** as:

 |=**M A** (read "**A** is true in interpretation **M**") iff |=**R,M A** for all **R**.

**M** is said to be a model for **A** if |=**M A**.

In interpreting a relational language, we stipulate that equality hold of every individual in the domain, and we also stipulate a set of integrity constraints. An **integrity constraint** for a predicate **P** has the form (where **T1,...,Tn** are the types):

(**x1**)...(**xn**) [**P (x1,...xn)** (**T1x1** & ... & **Tnxn**]

For example, we might lay down a constraint upon

Spouse (x,y)

that x be a person and y be a person(we don’t allow people to marry animals, and we don’t apply the concept of marriage to pairs of animals):

(x) (y) [Spouse (x,y) (Person(x) & Person (y))]

Now that we have the assertoric logic set forth, we turn to the erotetic extension. Our goal is to develop a formalism that will allow us to specify what it is for a given statement to correctly answer a given question. We will do this in two stages: first, we will devise a set of rules that govern the *acceptability* of answers, and then (in the next section) we will state a criterion for *truth* of answers.

Following Belnap (1963) and Belnap and Steele (1976), whose term system we will adopt, we view a question as having the form ‘?**RS**,’ where **R** is the **request** and **S** the **subject**. Roughly defined, the request specifies the amount of information desired, while the subject specifies the sort of information to be examined. The question mark we view as an operator that takes request-subject pairs onto questions. A **query** is just a question applied to (or "raised within") a database. In this section we will focus on representing questions without concern for databases; we will look at queries later.

We discuss the subject S first. The subject of a whether-question presents it range of alternatives explicitly. That is, the subject has the form of a set of statements {**A1, A2,..., An**} where none of the Ai's are repetitions or conjunctions solely of the others of the set. For example, the subject of the question

Is John married?

is the pair

{John is married, John is not married} Symbolically

{Married (John), Married (John)} Again, the subject of

Has John stopped abusing drugs?

is

{Has\_abused (drugs, John) and Now\_abuses (drugs, John), Has\_abused (drugs, John) &

Now\_abuses (drugs, John)} A more elaborate example:

Which are on sale today: ham, turkey, or steak? has the subject

{On\_sale (ham, today); On\_sale (turkey, today); On\_sale(steak, today)}

The subject of a which-question presents a potentially infinite set of alternatives, by means of statement functions, some of which are type constraints and the others the matrix. For example, the question

Which Topeka restaurants take credit cards?

has as subject matrix

Accepts (x,credit\_cards) & Located (x, Topeka) and as type constraint

Restaurant (x).

We will separate the subject types from the matrix by a double slash:

{Restaurant (x)//Accepts (x, credit\_cards) & Located (x,Topeka)} In general we can symbolize the subject of a which-question as

{**T1x,...,Trxr** // **A(x1,...xn)**}

where 0 r n.

Note two things. First, we can push the type constraints over into the matrix if we choose, although there is no good reason for doing so. For example, the subject

{Woman (x), Man (y)//Wife(x,y)} can be put equivalently

{Woman(x)//Man(y) & Wife(x,y)

or even

{// Woman(x) & Man (y) & Wife (x,y)}

where '' is the null symbol.[[7]](#endnote-7)

Second, we should note that the subject of a whether-question is a special case of the subject of a which-question (and so whether-questions are a subset of which-questions). Consider the examples of whether-questions given earlier. The subject of

Is John married?

is

{Married (John), Married (John)}

But this can be put instead as the subject of a which-question (using the identity operator, present in any relational language)

{// (x = John) & Married (x)}

Again, the subject of

Has John stopped beating his wife? can be put

{// (x = John) & Stopped\_beating\_wife (x) & Now\_beats\_wife(x)}

We will take the view, then, that whether-questions are but a special case of which-questions, and state our rules (governing the acceptability of answers) in terms of which-questions only.[[8]](#endnote-8)

The **request** of a question is a four-tuple <**Rrange**, **Rcomplete**, **Rdistinct**, **Rdefault**>. The **selection-size specification Rrange** is a specification about the lower and upper limits of the size of the selection. The lower bound ‘l’ is some integer greater than or equal to one (any question requests at least an alternative be selected). The upper bound ‘u’ can be any finite positive integer greater than or equal to the lower bound, or may be omitted entirely (which we indicate by using the null symbol ‘’). Thus Rrange takes the form

l...u (1 l u)

or

l...(1 l).

The selection-size specification allows us to express the difference between

What is the only Indian restaurant in Topeka?

and

What are some Indian restaurants in Topeka?

The first question involves a request for a single alternative, so Rrange = 1...1, while the second question involves a request for at least one alternative but sets no upper limit on the answer, so Rrange = 1....

The second element of the request is the **completeness-claim specification Rcomplete**.

Some questions, like

Which Chinese restaurants in Topeka take credit cards?

implicitly require the respondent to make a claim that the list given is complete, while other questions, like

What are some Chinese restaurants in Topeka that take credit cards?

do not. The completeness-claim is always a comparison of the number of alternatives selected with the total number of true alternatives. The completeness-claim could be that all the true alternatives have been given, or all but one, or most, or 10%, but in practice only one completeness-claim is important, namely, the claim that all true alternatives have been given. This is an especially important claim if the database is incomplete. Accordingly, we will stipulate that Rcomplete take the form of either '' or '’ depending upon whether the maximum completeness-claim (that all true instances have been selected), or else that no such claim is specified.

The **distinctness-claim specification Rdistinct** is a specification regarding whether or not the selection given contains any redundancies. For example, the statement

The Greenhouse and the Holiday Inn South restaurant are restaurants in Topeka. would in most situations not be an acceptable answer to the question

What are two different restaurants in Topeka?

if it turns out that "The Greenhouse" is in fact just the name given to the restaurant in the Holiday Inn South.

As with completeness-claims, various sorts of distinctness-claims are theoretically possible (that some of the alternatives are distinct, that all but one are, and so on) but in practice only two are important. We will say Rdistinct has the form ‘‘ or ‘’, depending upon whether the answer is to include a claim that all the alternatives are distinct or whether no such claim is to be made. In most cases in ordinary contexts and with most databases, questions and queries assume that the selection given in response consists of entirely distinct individuals.

The final element of the request is the **default specification** Rdefault. Rdefault specifies what is to be done in the face of presupposition failure. A **presupposition** of a question is any statement that must be true if that question is to have any true direct answer. For example,

Has John stopped abusing drugs? takes as direct answers

John has stopped abusing drugs.

and

John has not stopped abusing drugs.

But for one of those alternatives to be true, it has to be true that he has beaten his wife.[[9]](#endnote-9) 8 Of course, this example is the classic example of a loaded question: whether John answers yes or no, he commits himself to the proposition that he has beaten his wife in the past. Questions that presuppose something false are called **loaded** and are best answered by a direct answer but a **corrective** answer (one that corrects false presupposition). Examples:

(question) Has John stopped beating his wife? (direct answer) Yes.

(corrective answer) He has never beaten his wife.

(question) What is an Indian restaurant in Topeka? (direct answer) The New Dehli Deli.

(corrective answer) There are no Indian restaurants in Topeka.

We will take the position that a person may well desire a corrective answer to his question if he cannot be given a direct answer—indeed, this seems to be the rule rather than the exception. We can allow the request for a correction in case of presupposition failure by indicating a **default** question **D**. (The guard question cannot, of course, be identical to the original question.) For example, the question

What is an Indian restaurant in Topeka?

can be guarded by the default question

Is there an Indian restaurant in Topeka?

The null symbol indicates that no default question is specified—we may suppose that an error message is printed in case of presupposition failure.

Here then are a few examples of questions and their formal expression.

 (1) Is Hunam a restaurant?

is formalized as:

?(1..1, , ,){ // (x = Hunam) & Restaurant (x)}

or

 ?(1..1, , , ) {Restaurant (x)//x = Hunam

if "Restaurant” is a type.

(2) Which of these are (different) Chinese restaurants: Hunam, The Peking Duck, and Solley's?

?(1.., , , ) {Restaurant (x)//Chinese(x) &[(x = Hunam) v (x = The\_Peking\_Duck) v (x = Solley's)]}

(3) Which if any of these are (different) Chinese restaurants: Hunam, The Peking Duck, and Solley's?

?(1.., , , ) {Restaurant (x)//Chinese (x) & ((x =Hunam) v (x = The Peking\_Duck) v (x = Solley"s))}

where

D = ? (1..1, , , ) {(Restaurant (Hunam) & Chinese(Hunam)) v (Restaurant (The\_Peking\_Duck) & Chinese(The\_Peking\_Duck)) v (Restaurant (Solley's) & Chinese(Solley's))}

 (4) What are two (different) Greek restaurants in Topeka?

?(2..2, , , ) {Restaurant (x)//Greek(x) & Located (x, Topeka)}

By making the exact form of the request and the subject explicit, we can formulate rules governing the form any acceptable answer to a given question must have. We will state these rules recursively. The first set of rules applies to questions without default questions specified. We then state a general rule that applies to any questions with a default question specified. In what follows, will represent the question and the answer.

First, consider a which-question that specifies a size range but specifies neither a completeness claim nor a distinctness claim. Then the constraint on any answer is that it be a conjunction of instances of the type constraints and matrix. Formally, where ‘**Q**’ is the question and ‘**A**’ the answer:

**Rule 1**: if **Q** is

?(l..u, , , ) {(T1x1 ,,, Trxr) // Ax1...xn} then **A** must be

(T1a1/1 &...& Tra1/r & Aa1/1...a1/n) &...& (T1as/1 &...& Tras/r & Aas/1...as/n), l s u Example:

What is an example of a friendly bear?

Rule 1 allows

Yogi is a bear and Yogi is friendly.

but does not allow

Yogi is not a bear and Yogi is friendly. nor does it allow

Yogi is a bear and Yogi is not friendly.

Next, consider a which-question that specifies a selection size range and a completeness- claim. Then any acceptable answer must involve a conjunction where each conjunct satisfies the type and matrix conditions and where the number of conjuncts falls in the range where the

number of conjuncts falls in the range indicated, and a claim that if any individuals satisfy the type and matrix conditions they are identical to the ones cited. Formally:

**Rule 2:** If **Q** is of the form

?(l. ..u, , , ) {T1x1,..., Trxr // Ax1...xn}

then **A** must be

{[T1a1/1 &...& Tra1/r & Aa1/1...a1/n &...& T1as/1 &....& T1as/r & Aas/1...as/n] &

x1...xn [(T1x1 &...& Trxr) (Ax1 ...xn ((x1,n = an/1,n) v...v (x1,n = as/1,n)))]}

where (x1,n = ak/1,n) is the conjunction (x1 = ak/1) &...& (xn=ak/n) and l s n. We could also state the completeness-claim using type-restricted quantifiers:

(x1/T1)...(xr/Tr) (xr+1)...(xn)[Ax1...xn (x1,n = a1/1,n) v...v (x1,n= as/1,n))] Example:

Which bears are friendly?

Rule 2 allows

Yogi is a bear and Yogi is friendly and for any x, if x is a bear then if x is friendly then x is identical to Yogi.

but does not allow

Yogi is a bear and Yogi is friendly.

Next, consider a which-question that specifies a selection-size range and a distinctness- claim, but no completeness-claim. Then an acceptable answer will assert a conjunction where each conjunct satisfies the type and matrix conditions and where the number of conjuncts falls in the specified range and where an assertion is made that no two individuals cited are the same.

Formally:

**Rule 3:** If **Q** has the form

?(l...u, , , ) {T1x1,...Trx r // Ax1...xn}

then **A** must have the form

{[T1a1/1 &...& Tra1/r & Aa1/1...a1/n & T1as/1 &...& T1as/r & Aas/1...as/n] & [**&**(1ijp)

**V**(1kn) (ai/k ak/k)]} where l s u and where

**V**(1 k n) (ai/k aj/k) abbreviates

(ai/1 aj/1) v...v (ai/n aj/n)

i.e. the statement that the ith and jth conjuncts are distinct, and so the full claim that all conjuncts are distinct will look like

**&** (1 i j p) **V** (1k n) (ai/k aj/k).

So, for example, to say that the list

(11, 13, 17)

is not redundant is to say

(11 13) and (11 17) and (13 17).

Also, to say that the list of pairs

((1,2), (1,3), (3,4))

is not redundant is just to say

{[(1 1) v (2 3)] & [(1 3) v (2 4)] & [(1 3) v (3 4)]}.

Example:

Which different bears are friendly?

Rule 3 accepts

Yogi is a bear and Yogi is friendly and Winnie is a bear and Winnie is friendly and Yogi is not identical to Winnie.

but does not accept

Yogi is a bear and Yogi is friendly and Winnie is a bear and Winnie is friendly.

Finally, consider a which-question that specifies a size range, completeness-claim, and distinctness-claim. Then the form of the answer is just the fusion of the two rules above.

**Rule 4:** If **Q** has the form

?(l...u, , , ) T1x1,...Trxr // Ax1...xn

then **A** must be of the form

{[T1a11 &...& Tra1/r & Aa1/1...a1/n &...& T1as/1 &...& Tras/r & Aas/1...Aas/n] & (x1)...(xn) [(T1x1 &...& Trxr) (Ax1...xn ((x1,n = Aj/1,n) v...v (x1,n = as,n)))] & [**&**(1 i j r) **V** (1 k n) (ai/k aj/k)]}, l s u.

Example:

Which different bears are friendly?

Rule 4 sanctions:

Yogi is a bear and Yogi is friendly and Winnie is a bear and Winnie is friendly; and for any x, if x is a bear then if x is friendly then either x is identical to Yogi or is identical to Winnie; and Yogi is not identical to Winnie.

In this manner we can explicate questions that do not involve a default specification.

Questions that do specify a default question can be handled easily.

**Rule 5:** Let **QD** be a question involving a specified default question **D** and let **Q**be the same question without a default specification. Then any acceptable answer **A** to **QD** will have the form **A1** or **A2**, where **A1** is an acceptable answer to **Q**and **A2** is an acceptable answer to **D**.

Example:

What (if any) is an Indian restaurant in Topeka?

Rule 5 allows

The New Delhi Deli

And

There are no Indian restaurants in Topeka

but not

<error>.

The grammar of elementary question, then, consists of FOL plus a new operator ‘?,’ a few obvious additional rules of construction (for the various particular forms of ‘?<Rrange, Rcomplete, Rdistinct, Rdefault> S’) and five rules governing the acceptability of answers. These latter rules are the heart of the subject, of course; they are perhaps reminiscent of rules of inference or transformation in FOL itself, but, of course, are not to be considered as such. They

are better called "rules of response," rules which govern, not how information is to be transformed, but how (in what form) it is to be conveyed.

1. **Formal queries**

We now have an apparatus for specifying questions exactly. This apparatus allows us to get a start on the formal specification of the correctness of answers. An answer to a given question is **correct** if and only if: (a) it has acceptable form and (b) it is true. And we can now specify (by rules 1-5) what it is for an answer to have an acceptable form. We now want to specify formally what it is for an answer to be true.

Here databases enter the picture. We take the view that answers are true or false only against the backdrop of some body of information, i.e., some database. And so if we are to develop a formal account of truth relative to a database, we need to develop some formal model of databases.

Reiter(1984) contrasts two ways to formally represent databases. The first he calls the "model theoretic" view, the dominant view since the work of Codd and others in the early 70's. The second way of formally representing databases he calls the "proof theoretic" view. Reiter argues that the proof theoretic view is better in that it naturally allows for the representation of disjunctive information, null values, general facts, events, property inheritance (IS-A) hierarchies, and other "world knowledge." I find his arguments persuasive, but will not rehearse them here. Rather, I will work with the proof theoretical model because it very nicely compliments the erotetic apparatus outlined earlier.

Under the proof theoretic view, a database is taken to be a first order theory (in an underlying first order relational language). A **first order theory T** of language **L** is just a subset of the well-formed formulas of **L**. For example, with FOL as defined earlier, we might have a first order theory with ground atomic formulas:

 **T** = {Restaurant (Hunam); Restaurant (The\_Peking\_Duck); Restaurant (The\_New\_Dehli\_Deli); Chinese (Hunan); Chinese(The\_Peking\_Duck); Indian (The\_New\_Dehli); Located (Hunan,Topeka); Located(The\_New\_Dehli\_Deli, Topeka); Hunam=Hunam;The\_New\_Dehli\_Deli=The\_New\_Dehli\_Deli; The\_Peking\_Duck=The\_Peking Duck; Topeka = Topeka}

We can then express truth with respect to the database in terms of derivability:

**T** |-- **A**

means that **A** is derivable from **T** taken as a set of premises. In our example above, clearly

Restaurant (Hunam)

i.e., the fact that Hunam is a restaurant is derivable from the database—trivially, since it is an explicit member of the set. Less trivially,

(x) [Chinese(x) & Restaurant (x) & Located (x, Topeka)]

i.e., we can derive from the theory that there is at least one Chinese restaurant in Topeka. We can now try a plausible approach to defining what it is for an answer **A** to a query **Q** applied to a database **T** to be true: **A** is true with respect to **Q** if and only if **T**|-- **A**. More generally, a given question **Q** applied to a database **T is correctly answered** by **A** if and only if (a) **A** is of acceptable form as answer to **Q** (i.e., meets rules 1-5), and (b) **T** |-- **A**.

Let's follow this suggestion to see where it leads. Suppose we ask (of our sample database)

Which restaurants in Topeka are Indian restaurants?

The correct answer should be

[Restaurant(The\_New\_Dehli\_Deli) & Indian(The\_New\_Dehli\_Deli) & Located (The\_New\_Dehli\_Deli, Topeka)] & (x) [Restaurant (x) & Indian (x) & Located (x,Topeka)) (x= The\_New\_Dehli\_Deli)]

But while

|-- [Restaurant (The\_New\_Dehli\_Deli) & Indian (The\_New\_Dehli\_Deli & Located (The\_New\_Dehli\_Deli, Topeka)]

we cannot derive the completeness claim

 (x)[(Restaurant(x) & Indian(x) & Located(x,Topeka))  (x = The\_New\_Dehli\_Deli)]

Since universal claims cannot be derived from a theory consisting solely of ground atomic formulas, Reiter suggests first adding to a **domain closure axiom** to the effect that all individuals in the interpretation are named (present in) in the database. In the example above, this axiom has the form:

(x) [(x = Hunam) v (x = The\_Peking\_Duck) v (x = The\_New\_Dehli\_Deli v (x= Topeka)]

More generally, if the domain of the interpretation is (d1,...,dn) the domain closure axiom is:

(x) [(x = d1) v...v (x = dn)]

Also, we must add to completion-axioms for each predicate (to allow the derivability of

negations such as

Restaurant (Topeka).

In our small example, the completion-axioms would be:

(x) [Restaurant (x) ((x = Hunam) v (x = The\_Peking\_Duck) v (x = The\_New\_Dehli\_Deli)]

(x) [Indian (x) (x = The\_New\_Dehli\_Deli)]

(x) [Chinese (x) ((x = Hunam) v (x = The\_Peking\_Duck))]

(x) (y) [Located (x,y) (((x Hunam) v (x = The\_New\_Dehli\_Deli) v (x = The\_Peking\_Duck)) & (y = Topeka)))]

Happily, this all accords with our intuitions: to correctly answer a question that demands a completeness-claim, we have to know that the database to which the question is addressed is complete.

Again, suppose that we ask of our sample database: What are two different Chinese restaurants in Topeka?

The correct answer should be

[Chinese (Hunam) & Resaturant (Hunam) & Located (Hunam, Topeka)] & [Chinese (The\_Peking\_Duck) & Restaurant (The\_Peking\_Duck & Located (The\_Peking\_Duck, Topeka)] & (Hunam = The\_Peking\_Duck)

But from , even including the closure and completion axioms, we cannot derive the distinctness- claim

(Hunam = The\_Peking\_Duck)

We need to add more axioms to , namely, **unique name axioms**

(C1 = Cj)

for all (Ci, Cj), i j, in the set of constants of our relational language. When those axioms *are*

added, the distinctness claim is derivable.

Again, happily this accords well with our intuitions. To answer correctly a question that demands a distinctness claim, we have to know that our database contains no duplicate names. This is standardly assumed about databases in practice, and it is a very nice result that our formal specification mechanism *forces us to make that assumption explicit*.

When consists of the ground atomic formulas, a domain closure axiom, unique name axioms, completion axioms (and equality axioms specifying the reflexibility, transitivity and symmetry of the equality predicate), the only model for is the original interpretation, and thus truth in the interpretation amounts to provability in the theory. This completes our definition of correctness of answers to queries.

1. **Conclusion**

I have argued above that a relatively unnoticed area of fourth-generation logic, erotetic logic, devised by philosophical logicians over the last thirty years or so, can be use to complement database theory. In particular, erotetic logic complements very nicely the approach to representing databases taken by Reiter and other workers in the area of the domain calculus, enabling us to formally represent questions and the correctness of answers.

Of course, I suspect that there are other benefits to employing erotetic logic in database theory. One area of possible interest is the verification of relational database queries. That is, given a question posed in natural language, we may be able to verify that a query posed in one of the relational database query languages (domain algebra, domain calculus, transform languages such as SEQUEL/SQL, or graphic systems such as Query-By-Example) will correctly answer the question. For example, to verify that

Which Chinese restaurants in Topeka take credit cards? will be correctly answered by the SEQUEL query

SELECT NAME FROM RESTAURANT

WHERE TYPE = 'CHINESE' AND CREDIT = 'YES'

AND LOCATION = 'TOPEKA

applied to a database containing the relation

RESTAURANT (NAME, LOCATION, TYPE, CREDIT)

we need to verify that

?(l..., , , ) {Restaurant(x) // [Chinese(x) & Located (x, Topeka) & takes\_credit cards (x)]}

will be correctly answered by the query commands. Assuming that the *implementation* of the select command has been verified (it is presumably just a linear search routine, which can be verified in the standard way), we can show that the query will produce a complete list of names that fit the matrix and type constraints. However, we still need to conjoin the completeness- claim explicitly since it is not present as a datum in the relational database. The point here is that Reiter's view of a database as a first order theory is not something we can immediately implement in a standard relational database. A suggestion worth pursuing is that Reiter's view be implemented in PROLOG. After all, PROLOG does allow the inclusion of general rules as well as particular facts in the database.

In addition, AI researchers may find erotetic logic useful. Generally, erotetic logic gives us a handle on an important aspect of natural language, viz., the asking and answering of questions. More specifically, a number of philosophers of science have pointed out the central role why-questions play in scientific research (see especially Bromberger(1966)). Besides "capturing" (i.e., representing) world knowledge, we need to understand the various and intricate ways people interrogate that knowledge. Erotetic logic can help us in that quest.

**Gary James Jason**

**Department of Philosophy**

**Washburn University**

1. **Notes:**

 The phrase “deviant logic” appears, for example, in W. V. O. Quine’s book *Philosophy of Logic* (Englewood Cliffs, NJ: Prentice-Hall 1970). [↑](#endnote-ref-1)
2. Logicians vary in their enthusiasm for fourth generation logic, from unbridled enthusiasm to skepticism (even outright hostility). Typical of the enthusiasts is Routley:

On the whole there has been far too much effort expended on trying to accommodate philosophical clarifications to going logical systems and strait-jackets...rather than trying to develop logical systems to handle the evident data and to deal with going philosophical problems. Classical logical (i.e., FOL), although once and briefly an instrument of liberation and clarification in philosophy and mathematics, has, in becoming entrenched, become rigid, resistant to change and highly conservative, and so has become an oppressive and stultifying influence....Classical logic is, as now enforced, a reactionary doctrine.

Routley quoted in Brodie, et. al. (1984), p. 143

Routley touches upon one of the main arguments employed by many fourth generation logicians: that logic ought to be "empirical," that is, should square with the facts of cognitive life. And fourth generation logicians argue that FOL is not empirically adequate.

Many logicians are skeptical, however. Quine (1970) is only one of many who resist the move to go beyond FOL:

...let us not underestimate the price of a deviant logic. There is a serious loss of simplicity... And there is a loss, still more serious, on the score of familiarity...The price is perhaps not quite prohibitive, but the returns had better be good. (p. 86)

Those who are reluctant to embrace fourth generation logic seem to me to have two basic objections: first, that the newer logics are not extensional (hence lack many nice features present in FOL, which *is* extensionalist); second, that the newer logics are proliferating, yet don't fit together well (hence lack the elegance and simplicity of FOL). Let me elaborate.

We say that a logical expression is **extensional** if and only if co-referential expressions can be substituted for each other without changing the truth value. That means replacing singular terms with terms of the same denotation (e.g., 'Mark Twain" and 'Sam Clemens'), predicates with predicates of the same extension (e.g., 'man' and 'featherless biped'), or component sentences with sentences of the same truth value, does not change truth-value of the whole proposition. While FOL is extensional, fourth generation assertoric logics are typically intensional (i.e., non-extensional). Thus they often lack the simple decision procedures found in classical propositional logic.

And again, fourth generation logics are bewildering in their variety. Some seem to be more extensions of FOL than replacements; others seem to be wholesale replacements. It is quite unclear whether they all can be fitted together, and if so, just how. [↑](#endnote-ref-2)
3. The reader may wish to see Belnap and Steele's discussion of compound questions (Belnap/Steele (1976), pp. 87-107. [↑](#endnote-ref-3)
4. This system is similar to that of Belnap and Steele (1976) which grew out of Belnap's earlier work (Belnap (1963) with a number of simplifications and modifications. In particular, I view whether-questions as special cases of which-questions and I build default questions (guards) into the specification of simple questions. I do this to bring the symbolism closer to actual database queries. (The reader may wish read the alternate system devised by Aqvist (Aqvist (1965). For an elegant and highly formal account, see Kubinski (1980). [↑](#endnote-ref-4)
5. For a review of FOL, see Mendelson (1979) or Schoenfield (1967). In particular, the notion of interpreting a first language is discussed in Chapter Two of Schoenfield(1967). [↑](#endnote-ref-5)
6. Types are discussed in Schoenfield(1967), pp. 88-92; and also in Reiter(1984). Note that instead of using types, one can use a many-sorted logic to the same effect. For an explanation of the role of category or type conditions in erotetic logic, see Belnap(1963), pp. 72-73. [↑](#endnote-ref-6)
7. We can do the converse, i.e., push matrix conjuncts over into the type constraints, only if those conjuncts are types, or if we are willing to allow new types to be created at will—not a very practical approach. [↑](#endnote-ref-7)
8. This formal reduction can have a cost in simplicity. For example, the subject of the whether-question

Which of the following are on sale: ham, chicken, steak, lobster and turkey?

is

{(On\_sale (ham, today), On\_sale (chicken, today), On\_sale(steak, today), On\_sale (lobster, today), On\_sale (turkey, today))}

but turned into the subject of whether -question is

{//On\_sale (x, today) & ((x = ham) v (x = chicken) v (x = steak) v (x = lobster) v (x = turkey))}

which is rather more awkward.

Even more awkward are questions involving quantified statements, e.g. the subject of Are all swans white?

is

{(x)[Swan(x) White (x)] , (x) [Swan(x) White(x)]}

That can only be represented as the subject of a which-question if we use a universal type Ux true of all individuals (or if dealing with a language with no types, the function Ax v Ax will serve as well). Then the equivalent subject is

{// Ux & (x) (Swan(x) White(x))} [↑](#endnote-ref-8)
9. This point is more subtle if we interpret the question as a whether-question:

{// (x = John) & Has\_beaten\_wife (John) & Now\_beats\_wife (John)} Posing the question with that subject gets as direct answers

Has\_beaten\_wife (John) & Now\_beats\_wife (John)

and

[Has\_beaten\_wife (John) & Now\_beats\_wife (John)]

That latter alternative does not logically imply that John did beat his wife. However, it is still true under this construal that to answer "Has John stopped beating his wife?" by "No, it is not the case that he has stopped beating his wife" does not correct the presupposition of the question.

**Bibliography:**

	1. Anderson, Alan Ross and Nuel Belnap *Entailment: The Relevance and Necessity (vol. I)* Princeton: Princeton University Press (1975).
	2. Aquist, Lennart *A New Approach to the Logical Theory of Interrogatives* Uppsala: Filosofiska Studier (1965).
	3. Bartley, W. W. *Lewis Carroll's Symbolic Logic* NY: Clarkson Potter, Inc. (1977).
	4. Belnap, Nuel D. *An Analysis of Questions: Preliminary Report* Technical Memorandum issued by System Development Corporation (1963).
	5. Belnap, Nuel and Thomas Steel *The Logic of Questions and Answers* New Haven: Yale University Press (1976).
	6. Bridge, Jane *Beginning Model Theory* Oxford: Oxford University Press (1977).
	7. Bromberger, Sylvain "Why-Questions," in *Mind and Cosmos* ed. by Robert Colodny Pittsburgh: University of Pittsburgh Press (1966).
	8. Haack, Susan *Philosophy of Logics* Cambridge: Cambridge University Press (1978).
	9. Hughes, G. E. and M. J. Cresseell *An Introduction to Modal Logic* London: Methuen (1968).
	10. Hughes, G. E. and M. J. Cresswell *A Companion to Modal Logic* London: Methuen (1984).
	11. Jorgenson, Jorgen *A Treatise of Formal Logic* London: Oxford University Press (1931).
	12. Kubinski, Tadeus *An Outline of the Logical Theory of Questions* Berlin: Akademie-Verlag (1980).
	13. Mendelson, Elliot *Introduction to Mathematical Logic (2nd Ed)* NY: Van Nostrand (1979).
	14. Quine, W. V. *Philosophy of Logic* Englewood Cliffs, N.J.: Prentice-Hall (1970).
	15. Reiter, Raymond "Towards a Logical Reconstruction of Relational Database Theory" in Michael Brodie et. al. (eds) *On Conceptual Modeling* NY: Springer-Verlag (1984).
	16. Rescher, N. *Many-Valued Logic* NY: McGraw-Hill (1969).
	17. Rescher, Nicholas and Urquhart, A. *Temporal Logic* NY: Springer Verlag (1971).
	18. Shoenfield, Joseph *Mathematical Logic Reading* Mass: Addison-Wesley (1967).
	19. Thomas, Stephen *Practical Reasoning in Natural Language (3rd Ed.)* Englewood Cliffs, N.J.: Prentice- Hall (1986).
	20. Turner, Raymond *Logics for Artificial Intelligence* Chichester: Ellis Horwood Ltd (1985). [↑](#endnote-ref-9)