

LOBSTER: Combining AI and Database Techniques for GIS

Max J. Egenhofer and Andrew U. Frank

National Center for Geographic Information and Analysis and Department of Surveying Engineering, Boardman Hall, University of Maine, Orono, ME 04469, U.S.A. {MAX, FRANK}@MECAN1.bitnet

Abstract. The powerful logic-based concept of Prolog has been integrated with a database suitable for spatial data handling to form a database query language that is more flexible and powerful than the currently used SQL. This experimental implementation, called LOBSTER, allowed researchers to explore a number of areas of a GIS. Examples from object-oriented modeling, geomorphology, and query optimization show the application of such a language. Problems encountered during the application of LOBSTER include the absence of consistency checking during input of rules and facts, and the lack of appropriate techniques to detect cyclic rule definitions. Nevertheless, the experimental implementation showed that these techniques were extremely valuable for GIS.

1 Introduction

To develop a flexible and powerful program system for geographic information systems (GIS) is a challenging task. It is particularly difficult to construct programs that can assist users for all the different functions they expect from a GIS [Frank, 1984, Smith *et al.*, 1987]. During the past decade, advanced methods and techniques from computer science have been integrated into GIS. The use of high-level programming languages is commonplace today and the designers of GIS software use modern software engineering techniques [Aronson and Morehouse, 1983]. Database management systems and their principles have been applied and the specific requirements of spatial data handling studied [Frank, 1988]. Finally, it is recognized that some methods from artificial intelligence (AI) could be beneficial for GIS [Abler, 1987, Peuquet, 1987, Robinson *et al.*, 1987, McKeown, 1987].

Computer systems are essentially formal systems that manipulate symbols according to formal rules. These systems do not understand—in the sense that human beings understand—the meanings of the symbols or what they stand for. They follow the instructions of their programs with blinding speed, but without any “common sense.” Computer systems treat *formal models* which consist of two parts: (1) a *theory* with a collection of expressions in a formal language and (2) an agreed-upon *interpretation* of formal expressions which link the symbols used in the formal system with reality. The derivation of information is the process of proving a specific proposition within such a theory. The best-known language for a formal model is *first-order logic* which expresses *facts* and *rules* in a single, formalized matter [Gallaire *et al.*, 1984] and derives knowledge by using formal rules.

The deductive power of logic inference systems is typically used in AI systems [Barr and Feigenbaum, 1982, Hayes-Roth *et al.*, 1983]. Geographic Information Systems need these methods to help integrate data from different sources into a unified system [Robinson and Frank, 1987]. A deficiency of any AI-based system is the quantitative difference between AI/expert systems and database management systems [Mylopoulos, 1981]: while database management systems are good for the storage of large amounts of data elements (records) from a very few types (structured data), AI systems store a smaller number of facts, but of a much larger variety of types (unstructured data). In this paper, methods from AI research are combined with database management techniques to make both available to GIS. A particular system has been implemented which allowed us to conduct a number of experiments in promising areas for the use of AI in GIS.

The remainder of this paper is organized as follows: the next section discusses the need for intelligent GIS query languages. Prolog, an AI programming language, is proposed as a powerful query language if integrated with a database management system. LOBSTER is such a persistent language combining concepts of the Prolog programming language with database management techniques. The integration of DBMS and AI language are discussed as well as the implementation of the inference machine. The last section reports on some of our GIS test applications using LOBSTER, such as the implementation of object-oriented abstraction mechanisms, feature extraction in geomorphology, and query optimization in a distributed database environment. The paper concludes with a summary of drawbacks encountered during the use of LOBSTER as a Prolog-based query language.

2 GIS Query Languages

The production of spatial information on demand is the motivation for spatial query languages. A *query language* is a general means to request information about the contents of a database. Users formulate their requests to the database by describing their needs (“What to retrieve”) and the desired representation of the result (“How to represent the results”). A *spatial query language* is a tool suitable to interrogate spatial databases.

2.1 Database Query Languages

Database query languages are tools to facilitate access to a database and have been investigated by computer scientists for more than a decade. The term *query* refers to a statement requesting data to be retrieved from a database. Query languages are best-known with respect to (relational) databases. SQL, an acronym for Structured Query Language [Chamberlin *et al.*, 1976], is the standard relational query language [ANSI, 1986] and enjoys popularity in traditional database applications, such as accounting. Based on the underlying relational data model [Codd, 1970], SQL deals exclusively with relations, combinations of relations, and some “syntactic sugar” added to relational algebra, such as arithmetic capabilities, assignment of results to relations, and aggregate functions. Although SQL is very popular and has been standardized, there has been criticism that SQL queries can be difficult to understand [Luk and Kloster, 1986] and are particularly cumbersome to use for complex engineering applications.

The fundamental structure of SQL is the SELECT-FROM-WHERE block. The SELECT clause determines the attributes to display; the FROM clause describes the data sets needed to solve the query; and the optional WHERE clause specifies constraints upon the items to be retrieved. For example, the request for all lines with start or end nodes within a box described by two pairs of *x*- and *y-values* is formulated in SQL as follows:

```
SELECT line.id
FROM line, node
WHERE 20000 < x and x < 30000 and
      25000 < y and y < 30000 and
      (line.start = node.id or line.end = node.id);
```

Asked against a database containing the relations *node* and *line* (figure 1), the result is a relation with the two tuples B and C which are automatically displayed on the screen in a tabular format.

point id	x	y
2	22399.28	22379.72
3	23874.39	25479.93
8	19829.83	29878.98
line id	start	end
A	2	3
B	3	8
C	8	2

[h]

Fig. 1. The data sets *point* with the attributes *id*, *x*, and *y*; and *line* with *id*, *start*, and *end*.

Quel, the query language for the Ingres database management system [Stonebraker *et al.*, 1976], closely imitates the tuple relational calculus [Codd, 1972] and has the same expressive power as SQL, i.e., any query asked in SQL can be also asked in Quel.

The third major query language is Query-by-Example [Zloof, 1977]. It supports users with *skeleton tables* to be filled out like forms making the language more user friendly and easier to learn [Reisner, 1981] than conventional one-dimensional languages as command strings. Some additional conventions are used, e.g., an underscore character precedes domain variables to distinguish them from constants. Figure 2 shows the same query as above in Query-by-Example.

line	id	start	end	point	id	x	y
	P..n	_ <i>x</i>			_ <i>x</i>	> 20,000 and < 30,000	> 25,000 and < 30,000
	P..n		_ <i>x</i>				

[h]

Fig. 2. A Query-by-Example instruction to print (P.) the lines starting or ending inside of the rectangle ($20,000 < x < 30,000$, $25,000 < y < 30,000$).

2.2 Requirements for GIS Query Languages

GIS applications place specific demands on the expressive power and capacity of their query languages. Conventional query languages can certainly be used to access spatial objects stored in databases; however, it is difficult for them to express queries which involve particular spatial properties [Frank, 1982a, Egenhofer and Frank, 1988, Laurini and Milleret, 1989]. The following examples demonstrate typical GIS queries and underscore the problems traditional (relational) query languages have with their formulation and processing.

Frequently, GIS users ask for quantitative spatial information, such as the distance between two objects. Traditional query languages lack geometric concepts and do not support the formulation of user queries with spatial terms. Users with limited mathematical skills have difficulties in handling such a system. For example, to retrieve not only the lines starting or ending in a box, but also those crossing through it, users must explicitly formulate complex equations for line intersections. The requirement of such detailed mathematical knowledge makes “pure” SQL too complex to use for spatial applications.

Another complex query in the context of a GIS is to find the largest connected forest area which contains a specific parcel. This request for the *transitive closure* translates into the two operations: (1) to find the parcel X and place it in a set S and (2) to repeat—until the set S does not grow anymore—the operation: for each parcel P in S find all its neighbors N , and if the parcel type is forest then add N to the set S . Traditional database query languages lack the concepts of loops and recursion necessary to solve such queries, and therefore, cannot be used to formulate such queries.

Other GIS query language requirements include the graphical representation of query results and the display of context to make certain queries understandable [Egenhofer, 1989].

3 Prolog and Database Management Systems

3.1 Prolog

A Prolog-like language may be used as a query language to a GIS based on a database management system that can deal with large numbers of (spatial) data records. Prolog [Clocksin and Mellish, 1981] is an implementation of a subset of first-order predicate logic [Gallaire *et al.*, 1984]. It is based on *facts* and *rules* which are expressed as Horn clauses. A *clause* is a canonical representation of *predicates* $a_0 \cdots a_n, b_0 \cdots b_m$ in the form

$$a_0 \text{ OR } a_1 \text{ OR } \cdots \text{ OR } a_n \text{ IF } b_0 \text{ AND } b_1 \text{ AND } \cdots \text{ AND } b_m. \quad (1)$$

The left hand side of the clause, called the *consequent*, is the combination of all disjunctions (ORs) and the right hand side, the *antecedent*, has all conjunctions (ANDs). In a *Horn clause* the consequent predicates, i.e. the a_i 's, are restricted to zero or one instance, that is

$$a_0 \text{ IF } b_0 \text{ AND } b_1 \text{ AND } \dots \text{ AND } b_m. \quad (2)$$

The following example demonstrates the use a Prolog language based on the set of points and lines in figure 1. Predicates tagged by asterisks denote the definition of clauses, while untagged clauses stand for queries, and constants are capitalized, whereas variables start with lower case.

```
*point (2, 22399.28, 22379.72).
*point (3, 23874.39, 25479.93).
*point (8, 19829.83, 29878.98).

*line (A, 2, 3).
*line (B, 3, 8).
*line (C, 8, 2).
```

Given the implementation of the predicate `inBox (x, y, xLow, xHigh, yLow, yHigh)`, the following clauses define the rules for the inclusion of start and end point within a box:

```
*linePoints (l, p) IF line (l, p, end).
*linePoints (l, p) IF line (l, start, p).
*lineInBox (l, xl, yl, xh, yh) IF linePoints (l, p),
    node (p, x, y), inBox (x, y, xl, yl, xh, yh).
```

Prolog's inference mechanism allows then for the derivation of the query result:

```
lineInBox (lineId, 20000, 25000, 30000, 30000).

lineId = B
lineId = C
```

3.2 Combining Prolog with a Database Management System

A Prolog system is often viewed as a programming language, but it also contains certain aspects of a database management system [Kowalski, 1979], such as storage and retrieval of data and information. The use of Prolog or similar logic programming methods for database management have been proposed [Gallaire and Minker, 1984]. Using a Prolog system as a database [Motro, 1984] allows users to store unstructured—or minimally structured—facts without being aware of a database schema. Data and metadata are stored in the same format, so that users need not distinguish between and can search them the same way. Another approach, coupling an existing database management system with a separate Prolog system [Vassilou *et al.*, 1983], makes the potential of the Prolog programming language available for user interaction in an interactive environment with existing, traditionally structured databases [Jarke *et al.*, 1984]. In this combination, the database system stores the structured data, while the Prolog system is used

as an expert system or a tool for an enhanced user interface. Such interfaces to database management systems have been recently integrated into some commercial Prolog systems. Specific attention has to be paid to query processing. Performance will seriously degrade if the inference engine frequently passes control and data from the Prolog system to the database and vice-versa to process predicates one instance at a time.

3.3 Persistent Programming Languages and Prolog

The extension of a programming language with database management capacities is frequently referred to as a *persistent programming language*. Persistent programming languages have been designed and implemented as extensions of object-oriented programming languages, such as Smalltalk [Goldberg and Robson, 1983] and C++ [Stroustrup, 1986], but lack the simplicity and inference power of a Prolog language.

Standard Prolog [Clocksin and Mellish, 1981] leaves the provision for long term storage of facts and rules to file storage. Hence, we combined Prolog with a database management system to construct a *persistent Prolog*. Users can store data, structured according to the database schema, with Prolog facts and rules in the same database representing unstructured data. Simultaneously, they can use the inference mechanism to exploit the data.

Generally, most database management systems based on the network or relational data model can be used to support an inference mechanism of the form described. A database management system then serves as a general storage and retrieval system for clauses. This replaces the particular systems built in present Prolog implementations. The major extension is the use of disk storage and access methods; however, for GIS applications, the database system must respond to a number of specific requirements [Frank, 1988], for example:

- object-oriented database design [Dittrich, 1986],
- generalization/specialization as abstraction methods [Borgida *et al.*, 1984],
- suitability for modeling of geometric data [Härder and Reuter, 1985] with high-level abstractions of geometric objects, operations, and classes [Egenhofer and Frank, 1988, Güting, 1988],
- fast access based on spatial location [Frank, 1981].

The change in the environment—database in lieu of programming—aggravates some of the well-known problems of Prolog:

- User input of new facts and rules must be checked for consistency, e.g., comparing the spelling of new facts against previously stored ones.
- Execution speed with large spatial data collections must be improved so that acceptable response times can be guaranteed.

4 LOBSTER

LOBSTER is a persistent Prolog interpreter [Frank, 1984] using the PANDA database management system [Frank, 1982b]. PANDA incorporates many object-oriented concepts, such as generalization and association, extensibility with user-defined abstract data types, and spatial storage and access methods [Egenhofer and Frank, 1989b].

LOBSTER can be distinguished from the standard Prolog implementation [Clocksin and Mellish, 1981] in several aspects:

- Persistency of rules and facts: The rules and facts users store are kept on disk in a permanent database and available for any future work. In contrast, standard Prolog demands that the used rules and facts are loaded into main memory at the beginning of each session.
- Organization of rules and facts into groups: The persistency of all facts and rules requires that users have some tools to organize them so that they can keep track of what they had previously defined.
- Extensibility: New built-in predicates, written in a conventional programming languages, such as Pascal, can be easily implemented and integrated into the LOBSTER environment so that their actual implementation is hidden from the users.

Newer commercial Prolog products do provide some similar features, including access to relational database management systems.

Central to LOBSTER is the combination of a Prolog interface with a database management system to allow users to store GIS data and use the Prolog language and interpreter for building a query language. These two systems must be linked so that the Prolog interpreter has access to the data stored in the database management system and that these data may appear as facts in the Prolog system. The link between the two systems is achieved in two steps:

- Operations for database access are coded and integrated into the Prolog interpreter such that they appear as regular Prolog predicates, so-called *built-ins*, providing a low-level access to database facts from Prolog.
- Mappings are defined from the conceptual database schema to Prolog predicates and then implemented as Prolog rules using the built-ins for database access.

In order to store facts in a database, a database schema had to be designed. Figure 3 shows a solution in an extended entity-relationship diagram. The following example demonstrates how a Prolog rule is stored in the database according to this structure. The rule

```
grandFather (x, y) IF father (x, xy) and father (xy, y)
```

is stored as a *clause* with two *predicates* (`grandFather`, `father`) and three *symbols* (`x`, `y`, `xy`). The clause consists of three *atomic formulae* (`grandFather (x, y)` as the *consequent* atom, `father (x, xy)` and `father (xy, y)` as the *antecedent* atomic formulae). For each atomic formula, the corresponding variables are recorded with their number in the clause (e.g., `x=1`, `y=2` for `grandFather`). In the clause, each variable is connected with the respective symbol, either as *const* for a constant or *bound/unbound* for a variable before and after binding it to a value, respectively.

The implementation of LOBSTER follows the Prolog method of a *depth first search* and uses the facts in the order they are encountered in the linkage of predicates, atoms, and consequent clause. For use as a database retrieval system, the interpreter has to return with each success so the application can use the data in any way it is necessary, i.e., the Prolog interpreter works as though it was a co-routine. This excludes a simple

[ht]

Fig. 3. Database schema of LOBSTER in an extended Entity-Relationship diagram.

recursive implementation of the interpreter and requires explicit storage of the state of the interpreter during query processing to continue the search with backtracking after a solution has been found and processed. The backtracking algorithm is inherently sequential, using one data element at a time and its formulation in a navigational data manipulation language makes no problems. It cannot easily take advantage of the set oriented interface of a relational database. In order to reduce the number of physical disk accesses necessary for each step, physical clustering of records, as provided by PANDA, is beneficial.

5 Applications

In this section, some experimental applications will be presented which were built upon LOBSTER exploiting the power of logic programming, the database approach, and the extensibility.

5.1 Object-Oriented Abstraction Mechanisms

Object-oriented modeling is an innovative approach to designing software systems for complex situations in dealing with real-world problems as they occur in GIS [Dittrich, 1986]. It pursues the integration of traditionally separated methods used in DBMS, programming languages, and AI [Mylopoulos, 1981, Brodie *et al.*, 1984] and employs powerful abstraction mechanisms, such as *generalization* [Borgida *et al.*, 1984], *association* [Brodie, 1981], and *aggregation* [Smith and Smith, 1977]. The concepts of *inheritance* [Goldberg and Robson, 1983, Cardelli, 1984] and *propagation* [Rumbaugh, 1988], originating from programming languages, play an important role for GIS modeling [Egenhofer and Frank, 1989a]. A system like LOBSTER is particularly well-suited to demonstrate these sophisticated abstraction mechanisms in a concise fashion. Data and metadata are described in a uniform way so that users may easily exploit metadata for the formulation of rules and queries.

Inheritance is a method of defining a class in terms of one or more other, more general classes [Dahl and Nygaard, 1966], called an *is_a* hierarchy [Mylopoulos and Levesque, 1984]. Properties common for a superclass and its subclasses are defined only once—with the superclass—and inherited by all objects in the subclass. Subclasses may have additional, specific properties and operations which are not shared by the superclass. Figure 4 shows a generalization hierarchy with three levels of classes. The properties of a *building*, such as *address* and *owner*, are inherited to the subclass *residence*, and also transitively to the sub-subclasses *rural residence* and *urban residence*.

[ht]

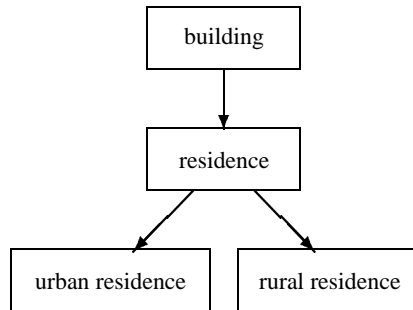


Fig. 4. Properties are transitively inherited from a superclass to all its subclasses, the sub-subclasses, etc.

LOBSTER has been used to prototype these concepts so that experiments could be run in a GIS environment [Egenhofer and Frank, 1989a]. Each property of a class is expressed as a predicate of the form $p(\text{class}, \text{property})$. Generalization is described as the *is_a*-predicate of the form $\text{is_a}(\text{subclass}, \text{superclass})$. The following facts describe the model depicted in figure 4.

```

*p (Building, Address).
*p (Building, Owner).
*p (Residence, Resident).

*is_a (RuralResidence, Residence).
*is_a (UrbanResidence, Residence).
*is_a (Residence, Building).
  
```

Inheritance is then defined by the predicate *properties* which recursively derives the properties associated with a class and all its superclasses.

```

*properties (class, property) IF p (class, property).
*properties (class, property) IF is_a (class,
    superclass),
    properties (superclass, property).
  
```

All properties of the class *urbanResidence* can then be determined with the predicate

```
properties (UrbanResidence, prop).
```

which the following values for the variable `prop` fulfill:

```
prop = Resident
prop = Address
prop = Owner
```

In aggregation and association hierarchies, two types of property values occur: (1) values that are specifically owned by the composite object and, therefore, distinct and independent from those of its components and (2) values of the composite object which depend upon values of the properties of all components [Egenhofer and Frank, 1986]. The mechanisms to describe such dependencies and ways to derive values is called *propagation* [Rumbaugh, 1988]. Propagation guarantees consistency, because the dependent values of the aggregate are derived and need not be updated every time the components are changed. For example, the property *population* of the class *county* is the sum of the *populations* of all related instances of the class *settlement*.

LOBSTER was used for a prototype implementation of propagation. The following (simplified) facts describe the county Penobscot as an aggregate of two settlements Bangor and Orono—and some more in the rural areas—with the property `settlementPopulation`.

```
*p (Orono, SettlementPopulation, 10,000).
*p (Bangor, SettlementPopulation, 50,000).
*p (Orono, PartOf, Penobscot).
*p (Bangor, PartOf, Penobscot).
```

The population of the largest settlement in a county is derived from the settlements as the maximum of their populations. This dependency is expressed by the following rule, stating that the population of a specific county is the maximum of the population of all settlements which are part of it.

```
*propagates (PartOf, SettlementPopulation,
             PopulationOfLargestSettlement, Maximum).
```

The generic rule for propagation is the following predicate. It describes the value of the property of an aggregate in terms of the values of the components using a specific aggregation function.

```
*p (aggregateClass, aggregateProperty,
    aggregateValue) IF
  propagates (relation, componentProperty,
             aggregateProperty, operation),
  p (componentClass, relation, aggregateClass),
  p (componentClass, componentProperty,
    componentValue),
  p (operation, componentValue, aggregateValue).
```

For example, the value of the property `countyPopulation` is then evaluated with

```
p (County, PopulationOfLargestSettlement, x).
```

and results in

$$x = 50,000$$

5.2 Geomorphology

The following experiment with LOBSTER describes how to define complex properties that can be derived from stored base properties. This example from geomorphology shows the definition of complex application-related terms in a rigorous manner so that users can understand them. These definitions are easy to program, but more important they make assumptions explicit so that different experts' opinions can be discussed. The distinction between different types of landscapes is evident for human observers, but at the same time they are difficult to express in formal terms. Verbal definitions of terms in natural language for geophysical phenomena have the substantial drawback that they are frequently based on other expressions which are not exactly defined, but are assumed to be generally understood [Frank *et al.*, 1986].

Symbolic processing for the extraction of geomorphologic features from landscape models has been proposed as a basis for formal analysis of terrain features [Palmer, 1984]. This method uses a triangulated irregular network to describe a digital terrain model. In such a tessellation, nodes have an identifier and x, y, and z coordinates, and edges are described by an edge-identifier, the identifiers of the start and the end node, and the identifiers of the left and right area. The definition of terrain features is then based on the classification of an edge according to the downslopes of their adjacent triangles [Frank *et al.*, 1986]:

- an edge is *confluent* if the slope of both adjacent triangles is towards the edge;
- an edge is *diffluent* if the slope of both adjacent triangles is off the edge; and
- an edge is *transfluent* if the slope of one adjacent triangle is towards the edge and off the edge for the other triangle.

Two edges are connected if they share a common node and a *valley* is then a sequence of connected confluent edges.

These rules can be easily expressed as predicates in first-order predicate logic and implemented in a Prolog language [Robinson *et al.*, 1987]; however, pure Prolog [Clocksin and Mellish, 1981] lacks arithmetic operations, such as trigonometric functions, necessary to calculate the slope and determine the direction of flow over an edge. Such calculations can be easily performed in a traditional programming language, e.g., FORTRAN or Pascal, and then integrated with LOBSTER. The definitions given here can be directly executed. It is not necessary to manually translate them into code with the usual risk of introducing errors and misunderstandings. The predicates defined are also available in an interactive setting for experimentations.

```
*connectedEdge (e1, e2) IF edge (e1, s, e) AND
                             edge (e2, s, ee) AND
                             notEqual (e1, e2).
*connectedEdge (e1, e2) IF edge (e1, s, e) AND
                             edge (e2, ss, e) AND
```

```

                                notEqual (e1, e2).
*connectedEdge (e1, e2) IF edge (e1, s, e) AND
                                edge (e2, e, ee) AND
                                notEqual (e1, e2).
*connectedEdge (e1, e2) IF edge (e1, s, e) AND
                                edge (e2, ss, s) AND
                                notEqual (e1, e2).

*confluentEdge (e)   IF edgeFlow (e, In, In).
*diffluentEdge (e)  IF edgeFlow (e, Out, Out).
*transfluentEdge (e) IF edgeFlow (e, In, Out).
*transfluentEdge (e) IF edgeFlow (e, Out, In).

```

The following two rules define a valley as a sequence of confluent edges and draw the resulting valley using the built-in predicate drawEdge as a co-routine.

```

*drawNextEdge (e) IF confluentEdge (e) AND
                                connectedEdge (e, ne) AND
                                drawNextEdge (ne).
*drawValley (e) IF confluentEdge (e) AND
                                drawEdge (e) AND
                                connectedEdge (e, ne) AND
                                drawNextEdge (ne).

```

5.3 Query Optimization

A common solution integrating multiple databases is the definition of a unifying query language which provides users with a view as if they dealt with a single system [Dayal, 1986]. This is a likely scenario for all those GIS which use a special purpose data storage system for recording spatial data and a standard database management system for non-spatial data. Particularly important in such a distributed database environment is the determination of an efficient query processing strategy. The term *query optimization* refers to the process of calculating various strategies to process a specific query and selecting a plan which most likely provides the least expensive execution time. Various factors must be considered, such as the size of the database, the number of records involved in processing a particular operation, and the time to access records which may be distributed across different sites.

Query optimization is an important issue in a Prolog environment with large amounts of facts. Assume a rule of the form

$$a(x, G) \text{ IF } b(x, z), c(z, G).$$

If the first predicate $b(x, z)$ is a large database relation then it is not economical to use every fact stored to bind x and z , and then to try to prove the rest of the clause. A more sophisticated method must consider the approximate size of database relations and the existence of access paths [Warren, 1981].

Based on the same principles as used in LOBSTER, a query optimizer has been implemented for a distributed spatial database [Hudson, 1989]. It uses Horn clauses as an internal representation into which the user queries are translated and then applies rules to determine an optimal strategy. Since the sequence of predicates within a clause is immaterial to the logic of the clause, the predicates may be regrouped. This reordering is based on

- the (estimated) size of the relation for which a predicate stands,
- the estimated size of the result of a predicate,
- the estimated cost of verifying a predicate, and
- the physical location of the data sets so that the transfer of data between various sites is minimized.

Query processing in logic databases [Bancilhon *et al.*, 1986, Sagiv, 1988] and rule-based query optimization [Freytag, 1987, Graefe and DeWitt, 1987] are ongoing research topics.

6 Drawbacks of a Prolog-Based Query Language

Prolog was designed to express logical relations in a short-lived environment where users are aware of all facts and rules stored. Facts and rules are stored in files and users recall them explicitly when they need them. This approach is dramatically different from a database situation which is used over a long time and users do not remember all previously entered facts and rules. Furthermore, the database concept allows several users to share facts and rules in a multi-user environment.

6.1 Integrity Constraints

The schema definition in a database usually contains integrity constraints to prevent users from entering data which are not in accordance with the stated goals. This restriction is necessary so that users and application programs may rely on certain properties of the data. Violations of these rules produce incorrect results or fail to find data stored. Prolog contains no provisions to prevent the entry of invalid or contradictory data. Simple spelling errors in the name of a predicate while entering a rule will make that predicate fail and the result will be “false.” Such errors are extremely difficult to detect. If the database contains large numbers of facts, visual inspection by browsing is not possible anymore.

If an expert system should work for a long time, integrity constraints must be included and new data entered must be checked against them. Some examples may clarify this problem:

- Predicate names must be checked against previously used ones.
- New predicates are required to be explicitly declared by the user. Prolog implicitly declares a predicate with its first use, similar to the creation of variables with their first use in BASIC or FORTRAN and their known problems.

- In order to assist users in avoiding redundant declarations of the same or similar predicates [Kent, 1981], have the user enter a description of the meaning of every new predicate declared. In addition, a query mechanism must be provided so that users may examine these descriptions later.
- The introduction of type constraints may help the checking of variables in predicates.

In LOBSTER, such assistance has been integrated and users have found them helpful.

6.2 Cyclic Rule Definitions

Systems using Prolog inference mechanisms are not well-protected against cyclic rule definitions, such as

```
a (x) IF b (x);
b (x) IF a (x);
```

Collections of rules established for use during a short time, or used as a package and not expected to be expanded by the user, are generally checked by the programmer against cyclic rules.

LOBSTER contains some mechanisms that detect cycles; however, checking for cyclic structures during run time is costly in terms of execution time. More appropriate techniques for the detection of cyclic rules, for instance during input of new rules, are an issue of active research in the Prolog research community.

6.3 Order of Rules

Some Prolog programs rely on the order in which facts and rules are entered into the database. The order of facts should not disturb the execution of a Prolog program in a strictly logical sense. It may produce the results in a different order, but the results should be the same.

On the other hand, the order of rules is important for many recursive rules, especially if a specific stop rule (containing a *cut*) needs to be tested and the general recursive rule follows. For instance, the order of the two rules in the following example is crucial to correctly formulate `notEqual`-predicate. This rule says that if the two predicates `x` and `y` are equal, then `notEqual` is false; otherwise, `notEqual` is true.

```
notEqual (x, y) IF equal (x, y), cut, fail.
notEqual (x, y) IF .
```

If the order of the two rules was exchanged, the intended logic of the operation would have been changed.

```
notEqual (x, y) IF .
notEqual (x, y) IF equal (x, y), cut, fail.
```

For any two predicates `x` and `y` the result would be true from the clause `notEqual (x, y) IF .`, and the second clause would never be tested.

It may be necessary to extend the data structure in figure 3 to include a class *program* consisting of several rules which are maintained and used in the given order.

7 Conclusions

We conclude that a GIS query language must provide a high-level abstraction of spatial data and geometric operations so that a user needs no explicit knowledge about their actual implementation; extensibility so that users may define new rules, maybe in the same system where data are stored and accessed; and recursion and loop constructs to formulate queries with transitive closure.

The use of AI methods and techniques for GIS are necessary to build flexible and powerful systems demanded by the user community. This paper reported on the integration of a specific AI method into a GIS programming environment. The result was LOBSTER, a persistent programming language based on Prolog. LOBSTER permitted us to study a number of areas in which Prolog and database techniques could be beneficial for GIS. The use of logic-based languages as GIS query languages has been explored as an alternative to the currently popular SQL type query languages.

LOBSTER was found to be powerful and flexible. Like any Prolog-based language, LOBSTER treats data and metadata in the same way; therefore, users may extend LOBSTER with appropriate rules whenever necessary. Furthermore, the extensibility of LOBSTER allows for definitions based on predicates that may be sometime difficult to implement in a pure first-order language. The implementation of additional built-in predicates which may be used within the language interface proved to be crucial for the implementation of propagation. Users' gained additional possibilities to access information in a GIS through this combination. An experimental system for geomorphologic feature detection demonstrated that LOBSTER can also be used to define specific interfaces for applications in an easy but comprehensive way.

Acknowledgements

Over the years, many colleagues and friends of ours helped with and contributed to the implementation of LOBSTER and we want to thank them all. Particularly appreciated were many discussions with Vince Robinson on the use of AI for GIS. Doug Hudson and Bruce Palmer worked on the applications we used as examples in this paper. The team implementing LOBSTER included R. Michael White and Eric Carlson.

References

- Abler, R., 1987. The National Science Foundation National Center for Geographic Information and Analysis. *International Journal of Geographical Information Systems*, 1(4):303-326.
- ANSI, 1986. *X3.135-1986 Database Language SQL*. American National Standards Institute.
- Aronson, P. and Morehouse, S., 1983. The ARC/INFO Map Library: A Design for a Digital Geographic Database. In: *Auto-Carto VI*, pages 346-382, Ottawa.
- Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J., 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. In: *ACM Symposium on Principles of Database Systems*, pages 1-15, Cambridge, MA.

- Barr, A., and Feigenbaum, E., 1982. *The Handbook of Artificial Intelligence*. Pitman, London.
- Borgida, A., Mylopoulos, J., and Wong, H., 1984. Generalization/Specialization as a Basis for Software Specification. In: M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modelling*, pages 87–114, Springer-Verlag, New York, NY.
- Brodie, M., 1981. Association: A Database Abstraction for Semantic Modeling. In: *2nd International Entity-Relationship Conference*, Washington, D.C.
- Brodie, M., Mylopoulos, J., and Schmidt, J., editors. 1984. *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer-Verlag, New York, NY.
- Cardelli, L. 1984. A Semantics of Multiple Inheritance. In: G. Kahn, D. McQueen, and G. Plotkin, editors, *Semantics of Data Types*, pages 51–67, Springer-Verlag, New York, NY.
- Chamberlin, D., Astrahan, M., Eswaran, K., Griffiths, P., Lorie, R., Mehl, J., Reiser, P., and Wade, B., 1976. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development*, 20(6):560–575.
- Clocksin, W., and Mellish, C., 1981. *Programming in Prolog*. Springer-Verlag, New York, NY.
- Codd, E., 1970. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387.
- Codd, E., 1972. Relational Completeness of Data Base Sublanguages. In: R. Rustin, editor, *Data Base Systems*, pages 65–98, Prentice Hall, Englewood Cliffs, NJ.
- Dahl, O.-J. and Nygaard, K., 1966. SIMULA—An Algol-based Simulation Language. *Communications of the ACM*, 9(9):671–678.
- Dayal, U., 1986. Query Processing in a Multidatabase System. In: W. Kim, D. Reiner, and D. Batory, editors, *Query Processing in Database Systems*, pages 81–108, Springer-Verlag, New York, NY.
- Dittrich, K., 1986. Object-Oriented Database Systems: The Notation and The Issues. In: K. Dittrich and U. Dayal, editors, *International Workshop in Object-Oriented Database Systems, Pacific Grove, CA*, pages 2–4, IEEE Computer Society Press, Washington, D.C.
- Egenhofer, M., 1989. *Spatial Query Languages*. PhD thesis, University of Maine, Orono, ME.
- Egenhofer, M. and Frank, A., 1986. Connection between Local and Regional: Additional ‘Intelligence’ Needed. In: *FIG XVIII. International Congress of Surveyors, Commission 3, Land Information Systems*, Toronto, Ontario, Canada.
- Egenhofer, M. and Frank, A., 1988. Towards a Spatial Query Language: User Interface Considerations. In: D. DeWitt and F. Bancelhon, editors, *14th International Conference on Very Large Data Bases*, pages 124–133, Los Angeles, CA.
- Egenhofer, M. and Frank, A., 1989a. Object-Oriented Modeling in GIS: Inheritance and Propagation. In: *AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography*, pages 588–598, Baltimore, MD.
- Egenhofer, M. and Frank, A., 1989b. PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques. In: T. Härder, editor, *Database Systems in*

- Office, Engineering, and Science*, pages 74–79, Springer-Verlag, New York, NY.
- Frank, A., 1981. Applications of DBMS to Land Information Systems. In: C. Zaniolo and C. Delobel, editors, *Seventh International Conference on Very Large Data Bases*, pages 448–453, Cannes, France.
- Frank, A., 1982a. MAPQUERY—Database Query Language for Retrieval of Geometric Data and its Graphical Representation. *ACM Computer Graphics*, 16(3):199–207.
- Frank, A., 1982b. PANDA—A Pascal Network Database System. In: G.W. Gorsline, editor, *Fifth Symposium on Small Systems*, Colorado Springs, CO.
- Frank, A., 1984. Extending a Database with Prolog. In: L. Kerschberg, editor, *First International Workshop on Expert Database Systems*, pages 665–676, Kiawah Island, SC.
- Frank, A., 1988. Requirements for a Database Management System for a GIS. *Photogrammetric Engineering & Remote Sensing*, 54(11):1557–1564, November 1988.
- Frank, A., Palmer, B., and Robinson, V., 1986. Formal Methods for the Accurate Definition of some Fundamental Terms in Physical Geography. In: D. Marble, editor, *Second International Symposium on Spatial Data Handling*, pages 583–599, Seattle, WA.
- Freytag, J.C., 1987. A Rule-Based View of Query Optimization. In: *SIGMOD Conference*, pages 172–180, San Francisco, CA.
- Gallaire, H. and Minker, J., editors. 1984. *Logic and Data Bases*. Plenum Press, New York, NY, 1984.
- Gallaire, H., Minker, J., and Nicolas, J., 1984. Logic and Databases: A Deductive Approach. *ACM Computing Surveys*, 16(2):153–185.
- Goldberg, A. and Robson, D., 1983. *Smalltalk-80*. Addison-Wesley Publishing Company, Reading, MA.
- Graefe, G. and DeWitt, D., 1987. The EXODUS Optimizer Generator. In: *SIGMOD Conference*, pages 160–171, San Francisco, CA.
- Güting, R., 1988. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In: J. Schmidt, S. Ceri, and M. Missikoff, editors, *Advances in Database Technology—EDBT '88, International Conference on Extending Database Technology, Venice, Italy*, pages 506–527, Springer-Verlag, New York, NY.
- Härder, T., and Reuter, A., 1985. Architecture of Database Systems for Non-Standard Applications (in German). In: A. Blaser and P. Pistor, editors, *Database Systems in Office, Engineering, and Science, Informatik Fachberichte, Vol. 94*, pages 253–286, Springer-Verlag, New York, NY.
- Hayes-Roth, F., Waterman, D., and Lenat, D., 1983. *Building Expert Systems*. Addison-Wesley Publishing Company, Reading, MA.
- Hudson, D., 1989. A Unifying Database Formalism. In: *ASPRS/ACSM Annual Convention*, pages 146–153, Baltimore, MD.
- Jarke, M., Clifford, J., and Vassiliou, Y., 1984. An Optimizing Front-End to a Relational Query System. In: B. Yormark, editor, *Annual Meeting ACM SIGMOD*, pages 296–306, Boston, MA.

- Kent, W., 1981. Data Model Theory Meets a Practical Application. In: C. Zaniolo and C. Delobel, editors, *Seventh International Conference on Very Large Data Bases*, Cannes, France.
- Kowalski, R., 1979. *Logic for Problem Solving*. Elsevier Science Publishing Co., New York, NY.
- Laurini, R., and Milleret, F., 1989. Solving Spatial Queries by Relational Algebra. In: *AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography*, pages 426–435, Baltimore, MD.
- Luk, W., and Kloster, S., 1986. ELFS: English Language From SQL. *ACM Transactions on Database Systems*, 11(4):447–472.
- McKeown, D., 1987. The Role of Artificial Intelligence in the Integration of Remotely Sensed Data with Geographic Information Systems. *IEEE Transactions on Geoscience and Remote Sensing*, 25(3):330–348.
- Motro, A., 1984. Browsing in a Loosely Structured Database. In: B. Yormark, editor, *Annual Meeting ACM SIGMOD*, pages 197–207, Boston, MA.
- Mylopoulos, J., 1981. An Overview of Knowledge Representation. *SIGMOD Record*, 11(2):5–12.
- Mylopoulos, J. and Levesque, H., 1984. An Overview of Knowledge Representation. In: M.L. Brodie and others, editors, *On Conceptual Modelling*, pages 3–17, Springer-Verlag, New York, NY.
- Palmer, B., 1984. Symbolic Feature Analysis and Expert Systems. In: *International Symposium on Spatial Data Handling*, pages 465–478, Zurich, Switzerland.
- Peuquet, D., 1987. Research Issues in Artificial Intelligence and Geographic Information Systems. In: *International Geographic Information Systems (IGIS) Symposium: The Research Agenda*, pages 119–127, Arlington, VA.
- Reisner, P., 1981. Human Factors Studies of Database Query Languages: A Survey and Assessment. *ACM Computing Surveys*, 13(1):13–31.
- Robinson, V. and Frank, A., 1987. Expert Systems for Geographic Information Systems. *Photogrammetric Engineering & Remote Sensing*, 53(10):1435–1441.
- Robinson, V., Frank, A., Karimi, H., 1987. Expert Systems for Geographic Information Systems in Resource Management. *AI Applications in Natural Resource Management*, 1(1):47–57.
- Rumbaugh, J., 1988. Controlling Propagation of Operations using Attributes on Relations. In: *OOPSLA '88*, pages 285–296, San Diego, CA.
- Sagiv, Y., 1988. Optimizing DATALOG Programs. In: J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698, Morgan Kaufmann, Los Altos, CA.
- Smith, J. and Smith, D., 1977. Database Abstractions: Aggregation. *Communications of the ACM*, 20(6):405–413.
- Smith, T., Peuquet, D., Menon, S., and Agrawal, P., 1987. KBGIS-II: A Knowledge-Based Geographical Information System. *International Journal of Geographical Information Systems*, 1(2):149–172.
- Stonebraker, M., Wong, E., and Kreps, P., 1976. The Design and Implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222.

- Stroustrup, B., 1986. *The C++ Programming Language*. Addison-Wesley Publishing Company.
- Vassilou, Y., Clifford, J., and Jarke, M., 1983. How Does an Expert System get its Data? In: M. Schkolnick and C. Thanos, editors, *Nineth International Conference on Very Large Data Bases*, pages 70–72, Florence, Italy.
- Warren, D., 1981. Efficient Processing of Interactive Relational Database Queries Expressed in Logic. In: C. Zaniolo and C. Delobel, editors, *Seventh International Conference on Very Large Data Bases*, pages 272–281, Cannes, France.
- Zloof, M., 1977. Query-by-Example: A Database Language. *IBM Systems Journal*, 16(4):324–343.