Frank, A. U. "Extending a Network Database with Prolog." 14. Orono, Maine, USA: Department of Civil Engineering, 1984.

Department of Civil Engineering

SURVEYING ENGINEERING

Extending a Network Database with Prolog

Andrew U. Frank

Report No. 40

Abstract
A network database management system is extended with a Prolog interpreter built into it. This allows to store data structured according to the database schema together with Prolog facts and rules, to represent unstructured data. Further structured and unstructured data wil be available for the Prolog interpreter, so that Prolog can be used as a query language for the complete database.

The change in the environment - database in lieu of programming - aggravates some of Prologs known problems. User input of new facts and rules must be checked for consistency (spelling, no contradictions with already stored facts) and execution speed with large data collections. We conclude, that a typing mechanism for Prolog could help to maintain consistency. To improve execution speed, similar methods as used in dbms systems to optimize query processing can easily be applied.

This paper was presented at the First International Workshop on Expert DataBase Systems, held at Kiawah Island, S.C. in October 1984 and appeared in the proceedings, Editor Larry Kerschberg.

University of Maine at Orono
103 Boardman Hall
Orono, Maine 04469

1.Introduction

Prolog can be viewed strictly as a programming language, but a Prolog system has also certain aspects of a database management system [Kowalski 1979]. Many different proposals to use Prolog or similar logic programming methods for database management can be found in recent literature [Gallaire 1976]. They either propose to use a Prolog system as a database [Motro 1984] or coupling an existing database system with a separate existing Prolog system [Vassilou 1983].

The first approach allows storage of unstructured (i.e. minimally structured) facts, without bothering the user with a database schema and with no problems of updating the database schema to changing requirements. A browser lets the user search thru the database and the metadata alike.

The second idea makes the potential of the Prolog programming language available for user interaction in an interactive environment with an existing, traditionally structured database.

We started with the observation that database systems are extremely beneficial to many large applications, but they are not easily suited to organizing data in the personal environment, as found in the organization of the data for a research projects, in office automation, or Computer Aided Design etc. As an example, consider a data collection for your personal use: it should contain persons and their addresses, literature references (there might be an overlap between authors and your friends), lists of things to do, and other arbitrary clippings (look at your desk!). Some of these data are easily classified into datatypes and a database schema may be established. Other classes change often and finally there are a number of odd data elements (the handwritten notes in your address book) containing very different things. Including these in the db schema produces an extremely large schema with many different types, but for each type we will ever store only very few data (in the extreme case only one). However, there are typical database problems associated with such data collections, like consistency constraints (telephone numbers, form of bibliographic references etc) and, of course, securing the data against loss. We conclude that the present db systems are not well suited for the task.

Not only are they unable accomodate the changing requirements. A

personal database must follow the development of the job, the project, etc. A complete analysis of the requirements for drafting a db schema is not possible, because the requirements are not known at the beginning; this is an important problem but not the only one.

There are also quantitative differences. There will be a large number of schema relations populated with only a very few facts. This reflects the quantitative difference between AI and database, as observed by Mylopolous [Mylopoulos 1981]: databases are for storage of very large amounts of data elements (records) from a very few types(**structured data**), artificial intelligence systems store a smaller number of facts, but of a much larger variety of types (**unstructured data**).

In order to cope with changes in the requirements, some of the dbms based on the relational data model feature operations to change the db schema and help the user to rearrange his data. The limitations in the number of data types (record types, relation schemas) found in many dbms could principally be extended. Nevertheless, a traditional dbms does not seem to be an adequate solution for storing facts in an expert system, where only a few records of any type are stored, but extremely many different types are encountered. Problems become most obvious if we consider the difficulty of keeping track of the changing database schema.

2. Possible Solutions

Using an existing Prolog implementation may be attractive for experimentation and research, but the known systems can not cope with large numbers of facts simultaneously. Generally facts are grouped by the user and stored in files. For use they are read into memory ('consult'), and only facts presently in memory are used in the interference process. The amount of facts kept in memory is not only limited by the memory size and the address range, but, at least in some implementations, influences also the speed of inference.

Coupling an existing dbms with an existing Prolog system, using the first to store the structured data, the latter used as an expert system or to create an enhanced user interface, is reported in [Jarke 1984]. This solves the aforementioned problem, but cooperation between the two systems may cause problems. To pass control and data from the one to the other is most likely time consuming and response time may then become a problem.

Third, it seems feasible to augment an existing dbms with an inference

engine (i.e. a Prolog interpreter), storing facts and rules in a few additonal data type in the database. This permits the use of standard db techniques for storing the data, especially storing large amounts of data in structured form, but allows also for the inclusion of unstructured data without requiring new db definitions (relation schemas) for each type of fact. Having the Prolog interpreter within the dbms program gives fast access to the database during inference and storing facts and rules as database records within the database reduces the amount of programs for data access etc.

We had available the PANDA network database system [Frank 1982], completely written in modular Pascal, so we could attempt to join a Prolog intepreter internally with the database. This contribution will report this work and discuss the problems which became apparent.

The user's expectations from a Prolog programming system and from a database differ radically which should be reflected in the interface.

Second, applications which demand processing of large amounts of data are not well suited for the backtracking processing style of Prolog which is better suitable for treating the user interface. Providing the user with application-oriented built-in processing predicates, e.g. for graphical output which are written in a procedural and compiled language, may help to achieve the necessary speed and provide nevertheless Prolog's high level user interface.

3. Storing facts in a database

In order to store facts in a database a database schema had to be designed. Figure 1 shows our solution in an extended entity block diagram.

In this schema a Prolog fact 'grandparent (x,y) :- parent (x,z), parent (z,y)' is stored as one clause, two predicates ('grandparent' and 'parent'), three symbols ('x', 'y', 'z'), with three atomic formulae (abbreviated 'atom') 'grandparent (x,y)' stored as the head (consequent) and the two parent (x,z)' and parent(z, y)' as two antecedent atomic formulae. For each atomic formula, the variables used are recorded with their number in the clause (here x=1, y,=2 etc) and for the clause each of the variable is then connected (either as bound = constant or unbound = variablesymbol) with the respective symbol.

The present implementation uses an array to store the variable indices and

is therefore limited in the number of variables in an atom. We plan to install an extension mechanism for the rare predicates with higher arity. This seems to result in better performance, then to further normalize the 'atom' record and introduce a 'variable_in_atom(atom_key, variable_number)' relation (or the equivalent record and set in the network data model).

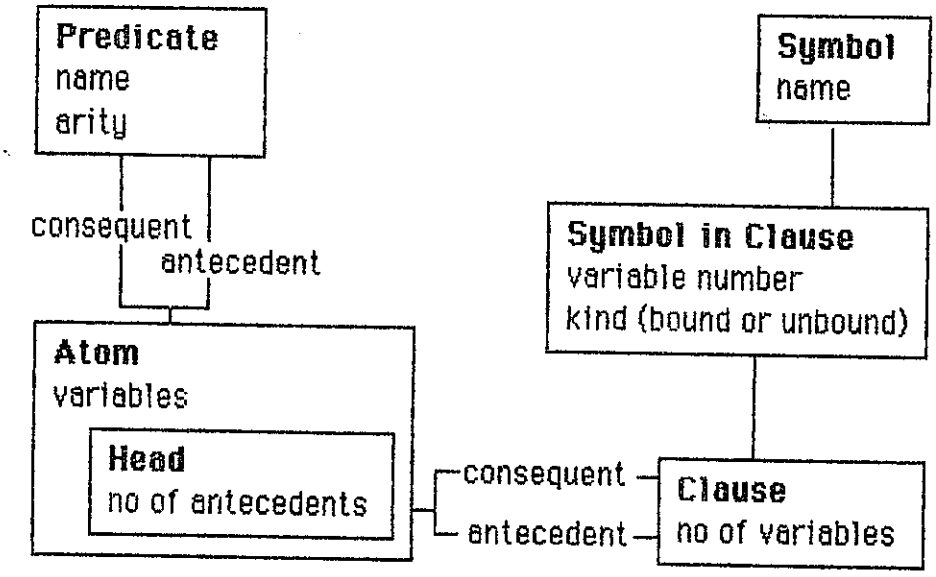

Figure 1

4. The implementation of inference engine

The present implementation follows the Prolog 'depth first' method and uses the facts in the order they are encountered in the 'predicate-atom-consequent' set (this represents at the moment the order in which the predicates were stored). 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 if it were a co-routine).This excludes the simple recursive implementation and requires explicit storage of the state of the interpreter in order to continue the search with backtracking after a solution has been found and processed. In fact, a proof is initiated as a backtracking on the null solution. The main body of the interpreter contains about 80 lines of Pascal code (four nested loops), and we used methods as introduced in [Dijkstra1976] for convincing us of its correctness(similar to [Emden1984]).

The rest of the programs were simple routines for storing and retrieving data in a navigating manner in the database, routines for managing the stack, input of facts and rules and output of results, and passing variables for unification. That is about 2500 lines of highly modular Pascal code. No new routines were necessary for memory management nor low level data storage.

5. Special requirements for a database to support inference

Principally most dbms based on the network or relational datamodel can be used to support an inference mechanism in the described form. The dbms then serves as a general form of storage and retrieval system for clauses and replaces the specific systems built in present Prolog implementations.

Depending on the implementation of the dbms, the resulting system may either be acceptable fast or impractically slow. The analysis must be based on the time required for each inference step and particularily, on the amount of disk access needed. The backtracking algorithm is inheritently sequential, using one data element at a time, and can not easily take advantage of the set oriented interface of a relational database and its formulation in a navigational data manipulation language causes no problems. In order to reduce the number of physical disk accesses necessary for each step, physical clustering of records may be used advantageous (e.g. System R[Astrahan1976] or in most network dbms).

Additional reduction results from buffering data once read in from disk. The PANDA dbms contains a two level buffering schema, first buffering db pages and then providing a large (presently about 4000 records) buffer for records, both managed in a least recently used strategy and fully transparent for the application program. We expect that proving a clause will use a limited number of database records over and over again, and a buffer of this size may reduce time consuming physical accesses to disk storage during backtracking.

6.Changes in the Prolog user interface necessary

In [Turner 1984] an interesting list of shortcomings of Prolog, as presently used, is given. At least three of the disadvantages he mentions, are of even greater concern in a database use of Prolog.

Prolog was designed to express logical relations in a short lived environment, were the user is fully conscient of all facts and rules stored. Storing facts and rules is done using files, and the user recalls ('consults')

them explicitly, when he wants to. This is radically different from a database situation which is used for a long period of time, and we may not assume that a user remembers all the previously entered facts and rules - independetly thereof wether the database is only for one or for several users.

The schema definition in a database usually contains integrity constraints to prevent users to enter data which are not in accordance to the stated goals. This is necessary in order that application programs (and users) may rely on certain properties of the data (invariants), and violation of these rules makes programs produce incorrect results, including not finding stored data. Prolog contains no provisions to this end. Simple spelling error in the name of a predicate when entering a rule will make this predicate to fail always (such errors are extremely difficult to detect; if the database contains large amounts of facts, visual inspection is not possible any more).

If an expert system should work for a long time, integrity constraints must be included and new data entered checked out. A few obvious examples:

- check predicate names against previously used ones and require users to explicitly declare new predicates (Prolog implicitly declares a predicate with its first use, which is similar to BASIC and FORTRAN where variables are created by their first use).
- when a new predicate is declared, let the user state its arity and check the arity in every later use
- when declaring a predicate, have the user enter a description of its meaning and provide a query mechanism for later use of this description (how can we avoid to have the same or a very similar predicate declared twic? cf. [Kent1982])
- use type constraints, to check variables in predicates. W-Grammars [vanWijngaarden1965] could be used as a theoretical framework. Turner works presently on a system WLOG, incorporating such ideas. [Turner 1984].

Further, systems using Prolog's type of inference mechanism are not well protected against circles in the rules (e.g a (x) :- b(x); b(x) :- a(x)). Collections of rules set up for use during a short period of time, or used as a package and not expected to be expanded by the user, are checked by the programmer against circles. If a system is open ended however, it may be appropriate to disallow storage of rules which lead to circles. How can

this be detected easiest? Check before storage of each rule if its reverse can be proven? A similar problem comes from introducing rules which contradict existing rules. Should the user be forbidden to insert such rules?

Finally, some Prolog program rely on the order in which facts and rules are entered into the database. This is clearly impractical in the envisaged environment, as a first writer of a rule may not know what other rules are later stored. We can differentiate order of facts and order of rules. Order of facts should not disturb the execution of a Prolog program (it may produce the results in a different order, but the results should be the same). Order of rules however are very important for many recursive uses of rules, where a special stop rule comes first (with a 'cut'), and the general recursive rule second. It may be necessary to extend the datastructure in figure 1 to include an entity 'program', consisting of several rules which are maintained and used in the given order.

7. Using the structured data in the database as facts

Obviously, the structured data in the database can be interpreted as facts and used by the inference mechanism. They can be interpreted as Prolog structures, or each record (tuple) split in several binary(ternary ..) facts. Similarly, information bearing sets in a network database can be regarded as binary fact. Logically this does not pose any problems; performance issues must be studied, however.

Assume a goal of the form a(x,Ann) :- b(x, z), c(z,Ann). If b(x,z) is a large database relation then it is clearly unpractical to use every fact stored to bind x, and z and then try to prove the rest of the clause. A more sophisticated method must take into account the approximate size of database relations and the existence of access paths. [Warren1981] explains a simple method, relying on estimates of the size of a relation and the size of the domains of the arguments. Principally, the database can keep track of the size of the realtion, whereas the domain size must be estimated by the user. The next goal to prove from a conjunction is then selected as to least increase the number of alternatives to be considered. Further, a conjunction of goals is checked to detect parts of it, which are independent from each other and can therefore be caried out separately - reducing backtracking greatly.

These are essentially the same methods that are used in query planning in database systems[Astrahan1976]. However, many Prolog rules are

sensitive to the order in which the atomic formulae in the antecedent are executed and reordering is not possible; this includes all rules using the 'cut' predicate and many others which use predicates with side effects. In a typed logic progamming system like WLOG, the 'not' predicate becomes a more natural interpretation and may make the 'cut' predicate unnecessary.

8. Access from the inference mechanism to the database schema

The database schema, including the estimates on relation size, integerity constraints, etc., must be considered as an additonal set of facts and rules which should be available for inference. Such information is useful for query planning and may reduce the amount of data to be retrieved (e.g. if a query can be answered using integrity constraints alone), in extremis, it may even help to answer questions when the pertinent facts are not stored at all.

This leads to a reduction of the difference between data and metadata in a database, a separation which is certainly necessary in a dbms to achieve fast response time, but not warranted on logical grounds. Considering a database with integrated inference mechanism, this difference vanishes and the previously established categories for metadata (relational schema, domain constraints, existency constraints [Fagin1980]) seem to be a very limited selection. We plan to use facts to store the schema information in the database and study at the moment, how to solve the bootstrap problem (the data structure to store the facts in must be described...) and methods to cache metadata for fast accesss during database operations.

Motro advocates that metadata should be available for answering user queries, and gives a number of interesting examples how generalization hierarchies can be used to improve responses [Motro1984]. Very similar data are necessary for the above mentioned typing structure to control consistency of facts and rules input by the user.

9. Building procedural rules into an inference mechanism

Despite the fact that almost any problem can be stated as a collection of rules, many problems are easily stated in a procedural language and execution of compiled loops is most of the time much faster than backtracking. Each Prolog system contains a number of built in rules, which can be understood as rules, but are implemented otherwise. In our area of application (cartography), a number of additonal built in procedures would be helpful, especially for graphical output, and

operations on geometric objects (e.g.calculating the surface of on area). This poses generally the question how to integrate a facility for the user to define additional built in operations into a Prolog environment. Beside of technical problems like argument passing, the solutions of which are heavily dependent on the programming language used, there are fundamental differences between a Prolog system and a usual high level procedural language subroutine.

A Prolog rule may be used with any of its variables initialized and computes result in any case, either one or many using backtracking. High level languages provide routines which have a fixed set of input parameters and compute thereof the output values. One possible solution to combine the two worlds is to write the compiled functions as standard routines producing output for one set of input parameters only and have them fail in all other cases. It is then possible to wrap these into a more general Prolog program which contains as a first rule a check for the binding of the arguments, a cut and a call to the compiled rule, and a Prolog formulation of the same operation as the second rule. If the first rule fails because it got the wrong input parameters, the second rule executes and uses backtracking to generate all possible answers. If the first rule has the correct arguments, the 'cut' excludes the second rule from later being tried by backtracking.

The Prolog interpreter implemented on the Lilith personal computer realized a comfortable and efficient interface to procedures written in Modula 2 using a precompiler. The necessary additional definitions are specified in a simple language and used to produce the procedure heads for the Modula 2 procedures and also to extend the Prolog interpreter with the new built in predicate [Schorn 1984].

10. Conclusion

We have augmented a network database system with an inference mechanism, interpreting Prolog. This raises a number of new questions, which were presented in this paper:

- checking user input of facts and rules, and applying consistency constraints in a Prolog system.
- optimizing the order of selecting goals to prove,
- extension of meta information in the database and diminishing the difference between meta data and data.

Assuming that the methods relational databases use to optimize query

execution can be transported to a Prolog interpreter, it should be discussed, what additional performance gains are to be expected from the approach of coupling an existing database with a separate inference mechanism. The translation from the sequential processing of the inference mechanism to the set oriented formulation in a query language may well result in better clustered access to disk storage and run faster. On the other hand, the same effect can be attained by increased buffer space for the database.

Finally we found that a typing mechanism may benefit different areas (consistency of the data collection, execution speed and the user interface). It seems worthwile to continue investigating this possibility.

The implementation of the inference mechanism is complete and we expect to attain about 1000 inferences per second, and future optimization will be possible. We presently work on the mechanism to let the user extend the built in predicates and study the order of executing goals.

References

Astrahan, M.M. et al., System R: Relational Approach to Database Management, ACM TODS, Vol.1,No.1, p, 97,1976

Clocksin, W.F., Mellish, C.S., Programming in Prolog, Springer Verlag, 1981

Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.Y. 1976

Fagin, R., Multivalued dependencies and a new normal form for relational data bases, ACM TODS, Vol. 2, No. 2, p. 262, 1977

Frank, A. , PANDA, A Pascal network database system,Proceedings ACM SIGSMALL conference, Colorado Springs Co. ,1982

Gallaire, H. Minker, J. (eds.), Logic and Databases, Plenum, 1978

Jarke, M., Clifford, J. Vassiliou, Y., An optimizing front-end to a relational query system, in: B. Yormark (ed.), Proceedings ACM SIGMOD '84, Boston, SIGMOD Record, Vol. 14, No.2, 1984

Kent, W., Data model theory meets a practical application, in: C. Zaniolo, C. Dlobel (eds.) Proceedings 7th Internat. on Conf. Very Large Data Bases, Cannes (France) 1981

Kowalski, R. Logic for Problem Solving, North Holland, 1979

Motro, A., Browsing in a loosely structured database, in: B. Yormark (ed.), Proceedings ACM SIGMOD '84, Boston, SIGMOD Record, Vol. 14, No.2, 1984

Mylopoulos, J. An overview of knowledge representation, Procedings of the workshop on data abstraction, data bases and conceptual modelling, Pingree Park, Co. , SIGMOD Record, Vol. 11, No.2, 1981

Schorn, P., Schulz, T. Description of LOGULA (in german) Swiss Federal Institue of Technology, Institute for computer scienc, Zurich (Switzerland) 1984

Turner, S.J. W-Grammars for logic programming, in: J.A. Campbell (ed.), Implementations of Prolog, Chichester, England, 1984 1984

van Emden, M.H., An interpreting algorithm for Prolog programs,

vanWijngaarden, et. al, Revised report on the algorithmic language ALGOL 1968, Acta Informatica, Vol. 5, p1, 1975

Vassilou, Y., Clifford, J., Jarke, M., How does an expert system get its data?, in: M.Schkolnick, C.Thanos (eds.) Proceedings 9th Internat. Conf. on Very Large Data Bases, Florence (Italy), 1984

Warren, H.D.H., Efficient processing of interactive relational database queries expressed in logic, in: C. Zaniolo, C. Dlobel (eds.) Proceedings 7th Internat. on Conf. Very Large Data Bases, Cannes (France) 1981

REPORT: Surveying Engineering Publications and Reprints
The following reports were published and are available upon request

1. Defining the Celestial Pole, A. Leick, Manuscripta Geodetica, Vol. 4, No. 2.

2. A New Generation of Surveying Instrumentation, A. Leick, The Maine Land Surveyor, Vol. 79, No. 3.

3. The Teaching of Adjustment Computations at UMO, A. Leick, The Maine Land Surveyor, Vol. 79, No. 3.

4. Spaceborne Ranging Systems - A useful tool for network densification, A. Leick, The Maine Land Surveyor, Vol. 80, No. 1.

5. Potentiality of Lunar Laser Range - Differencing for Measuring the Earth's Orientation, A. Leick, Bulletin Geodesique.

6. Crustal Subsidence in Eastern Maine, D. Tyler, J. Ladd and H. Borns; NUREG/CR-0887, Maine Geological Survey, June 1979.

7. Land Information Systems for the Twenty-First Century, E. Epstein and W. Chatterton, Real Property, Probate and Trust Journal, American Bar Association, Vol. 15, No. 4, 890-900 (1980).

8. Analysis of Land Data Resources and Requirements for the City of Boston, Epstein, E.F., L.T. Fisher, A. Leick and D.A. Tyler, Technical Report, Office of Property Equalization, City of Boston, December 1980.

9. Legal Studies for Students of Surveying Engineering, E. Epstein and J. McLaughlin, Proceedings, 41st Annual Meeting, American Congress on Surveying and Mapping, Feb. 22-27, 1981, Washington, D.C.

10. Record of Boundary: A Surveying Analog to the Record of Title, E. Epstein, ACSM Fall Technical Meeting, San Francisco, Sept. 9, 1981.

11. The Geodetic Component of Surveying Engineering at UMO, A. Leick, Proceedings of 41st Annual Meeting of ACSM, Feb. 22-24, 1981.

12. Use of Microcomputers in Network Adjustments, A. Leick, ACSM Fall Technical Meeting, San Francisco. Sept. 9, 1981. (co-author: Waynn Welton, Senior in Surveying Engineering).

13. Vertical Crustal Movement in Maine, Tyler, D.A. and J. Ladd, Maine Geological Survey, Augusta, Maine, January 1981.

14. Minimal Constraints in Two-Dimensional Networks, A. Leick, Journal of the Surveying and Mapping Division (renamed to Journal of Surveying Engineering), American Society of Civil Engineers, Vol. 108, No. SU2, August 1982.

15. "Storage Methods for Space Related Data: The FIELD TREE", A. Frank in: MacDonald Barr (Ed.) Spatial Algorithms for Processing Land Data with a Minicomputer. Lincoln Institute of Land Policy 1983.

16. "Structure des données pour les systèmes d'information du territoire", (Date Structures for Land Information Systems), A. Frank in: Proceedings 'Gestion du territoire assistee par ordinateur's, November 1983, Montreal.

17. "Semantische, topologische und räumliche Datenstrukturen in Landinformationssystemen (Semantic, topological and spatial data structures in Land Information Systems) A. Frank and B. Studenman, FIG XVII Congress Sofia, June 1983. Paper 301.1.

18. Adjustment Computations, A. Leick, 250 pages.

19. Geometric Geodesy, 3D-Geodesy, Conformal Mapping, A. Leick.

20. Text for the First Winter Institute in Surveying Engineering, A. Leick, D. Tyler, 340 pages.

21. Adjustment Computations for the Surveying Practitioner, A. Leick, (co-author: D. Humphrey, Senior in Surveying Engineering).

22. Advanced Survey Computations, A. Leick, 320 pages.

23. Surveying Engineering Annual Report, 1983-84.

24. Macrometer Satellite Surveying, A. Leick, ASCE Journal of Surveying Engineering, August, 1984.

25. Geodetic Program Library at UMO, A. Leick, Proceedings, ACSM Fall Convention, San Antonio, October, 1984.

26. GPS Surveying and Data Management, A. Leick, URISA Proceedings, Seatle, August, 1984.

27. Adjustments with Examples, A. Leick, 450 pages.

28. Geodetic Programs Library, A. Leick.

29. Data Analysis of Montgomery County (Penn) GPS Satellite Survey, A. Leick, Technical Report, August, 1984.

30. Macintosh: Rethinking Computer Education for Engineering Students, A. Frank, August, 1984.

31. Surveying Engineering at the University of Maine (Towards a Center of Excellence), D. Tyler and E. Epstein, Proceedings, MOLDS Session, ACSM Annual Meeting, Washington, March, 1984.

32. Innovations in Land Data Systems, D. Tyler, Proceedings, Association of State Flood Plain Managers, Annual Meeting, Portland, Maine, June 1984.

33. Crustal Warping in Coastal Maine, D. Tyler et. al., Geology, August, 1984.

34. St. Croix Region Crustal Strain Study, D. Tyler and A. Leick, Technical Report submitted to the Maine Geological Survey, June 1984.

35. Applications of DBMS to Land Information Systems, A. Frank, in: C. Zaniolo, C. Delobel (Ed.), Proceedings, Seventh International Conference on Very Large Databases, Cannes (France), September, 1981.

36. MAPQUERY: Database Query Language for Retrieval of Geometric Data and Their Graphical Representation, A. Frank, Computer Graphics Vol. 16, No. 3, July 1982, p. 199 (Proceedings of SIGGRAPH '82, Boston).

37. PANDA: A Pascal Network Data Base Management System, A. Frank, in: G.W. Gorsline (Ed.). Proceedings of the Fifth Symposium on Small Systems, (ACM SIGSMALL), Colorado Springs (CO), August, 1982.

38. Conceptual Framework for Land Information Systems - A First Approach, A. Frank, paper presented to the 1982 Meeting of Commission 3 of the FIG in Rome (Italy) in March 1982.

39. Requirements for Database Systems Suitable to Manage Large Spatial Databases, A. Frank, in: Duane F. Marble, et. al., Proceedings of the International Symposium of Spatial Data Handling, August, 1984, Zurich, Switzerland.

40. Extending a Network Database with Prolog, A. Frank, in First International Workshop on Expert Databases Systems, October, 1984, Kiawah Island, SC.

41. The Influence of the Model Underlying the User Interface: A Case Study in 2D Geometric Construction, W. Kuhn and A. Frank.

42. Canonical Geometric Representations, A. Frank

43. Computer Assisted Cartography - Graphics or Geometry, A. Frank, Journal of Surveying Engineering, American Society of Civil Engineers, Vol. 110, No. 2, August 1984, pp 159-168.

44. Datenstrukturen von Messdaten, A, Frank and B. Studemann, paper presented at IX International Course for Engineering Surveying (Graz, Austria) September 1984.