Frank, A. U. "Combining a Network Database with Logic Programming." 14. Orono, Maine, USA: Department of Civil Engineering, 1985.

Department of Civil Engineering

SURVEYING ENGINEERING

Combining a Network Database with Logic Programming

Andrew U. Frank

Report No. 45

Abstract

Combining a Prolog interpreter with a database management system can be viewed either as providing Prolog with structured, permanent storage, or extending the database with a powerful query language. The simlarity in the structure of network databases and logic-programming (Horn-clause proovers based on the resolution principle) has recently been discovered. With a minimal extension of Prolog (an additional data type 'entity identifier') facts stored in the database can be accessed with regular (first-order) predicates. The proposed query language is therefore well based within the theory of logic but the resulting constructs are also easy for users. Experience with a working prototype using a regular, network database to store Prolog facts and rules, but also permitting access to structured database entities from writing Prolog is reported.

This manuscript is in a preliminary form and comments from readers are encouraged and very welcome.

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

1.Introduction

There is considerable discussion going on about how results and experiences from the two fields of database research and artificial intelligence could be combined. Reseachers in both fields try to solve essentially the same problems, namely modeling parts of reality as accurately as possible. The approaches on both sides are different as database management is mostly concerned with managing very large numbers of facts of few different types, whereas artificial intelligence is more concerned with smaller numbers of facts from a large diversity of types [Mylopoulos 1981].
In a recent discussion [Kerschberg 1984] the following points where mentioned as the major shortcomings of present systems:

- databases : -too rigid structures to truly model reality
  -inflexible to adapt to later changes
  -access to data is either thru an extended traditional programming language including database operators, or using a query language with limited expressive power
- artificial intelligence systems :
  -main memory bound (not permanent and limited address space)
  -no modularization to structure very large knowledge bases

The diversity of approaches from the two sides is partly motivated by a concern for the best use of expensive hardware; now that more processing power is available, a more flexible and better adapted solution may become possible.

This paper describes a working prototype combination of a Prolog-like language with a network oriented database system. Such a combination can be looked at from two different angles:
-first,the database can be used to store the Prolog facts and rules, and is therewith providing AI systems with transparent management of permanent data on disk.
-second, Prolog can be considered as a host language (similar to COBOL as a host for a CODASYL Data Manipulation Language [CODASYL 1978]), providing a higher level, multi-purpose programming interface to the database.

This paper deals with both of these aspects. Its main contribution is in a proposal for including database access operations into the framework of logic-programming with minimal extension. The motivation for our work comes from the area of CAD/CAE systems, where complex structured data

must be stored and retrieved in different forms [Stonebraker 1984], [Retifuss, 1984]. We assume, as others, that a Prolog-like method may help to build more intelligent systems [cad comp graphics][Palmer 1984]. Especially it would provide an uniform language to express data and rules for data manipulation in a conceptually simple integrated framework, without the limitations of present day database query languages which lack transitive closure and recursion operations that are very important in CAD/CAE systems.

2.Organization of the Paper

The next section shortly explains the framework of logic programming without going into details. The first main section describes a datastructure suitable for storage of Prolog rules and facts (Horn clauses) and discusses some of the performance issues. The second main section shows methods of using Prolog as a host language for a Data Manipulation Language and describes a first prototype implementation. The following section then discusses some other features that should be integrated to make such a system truly useful . The final section argues that using both extensions described together in one system results in synergetic effects.

3. Logic Programming and Prolog

The background of logic programming is twofold: construction of theorem provers based on the resolution principle [Rboinson 1965] [Kowalski 1979] and natural language understanding system [Colmerauer 1973]. It uses the special form of **Horn clauses** to express logical (first order predicate) clauses. In [Gallaire 1984] a comprehensive discussion of the special properties of this restricted form of first order logic and their (performance) advantages over unrestricted forms is given. We will restrict the discussion here to such Definite Deductive Databases which consist of general deduction methods, elementary facts and deduction rules. The deduction methods can be built in into the inference programs and are the same for all applications. The specific application oriented knowledge is stored in facts and deduction rules which both can be formulated as Horn clauses (A1 and A2 and ... and An implies B). Horn clauses can also be interpreted as procedures with a head, the atomic formula that stands for the consequent, and a body, the atomic formulae that form the antecedent.

The notation is typically

```
consequent :- antecedents... .
```

or more detailed:

```
predicate1 (arg1, ... argn) :- predicate2 (arg1, .. ), predicate3 (arg1,...).
```

Example:

grandfather (g, s) :- father (g, p), father (p,s).

which is to be read:

g is the grandfather of sif g is father of p and p is father of s.

Symbols starting with lowercase letters are taken as variables, such starting with uppercase letters as constants (some Prolog implementation use different rules).

All predicates in the antecedent are to be connected by AND, and the separator ':- 'stands for IF. A clause may have no antecedent, it is then said to be a fact as it is true without depending on other clauses. Clauses that have antecedents are commonly called rules.

Running a Prolog program is actually proving a clause, but due to the restricted form of facts and rules (Horn clauses) efficient methods can be applied. The resolution principle [Robinson] allows to combine two clauses to form a new one. From

Q(c, d) :- P (a, b, c) and

P(e, f, g) :- R (g)

follows

Q(g, d) :- R (g).

The resolution principle can be applied repeatedly. A resolution proof consists of applying the resolution principle to all atomic formulae in the initial set, adding those newly derived atoms to the set and iterating the process until the set is empty. During each step of resolution a selection of the next atom from the set to process must be made; standard Prolog selects atoms from left to right adding new atoms to the front of the set (this is equivalent to a depth first search). For each step of the resolution a clause to be used must be chosen. If this choice does not lead to success, other possible choices must be tried (backtracking).

The following example should help to understand.

Example:

father (Adam, Bert).

father (Adam, Brigit).

father (Bert, Cecil).

Using the above rule about the grandfather, we can ask:

? :- grandfather (Adam, x).

the first resolution step results in the new set:

? [grandfather (Adam, x)]:- father (Adam, p) and father (p, x).

select atom: ? :- father (Adam, p).

and resolve with fact:

father (Adam, Bert). - success

continue from above
? (grandfather (Adam, x):- father (Adam, Bert)), father (Bert, x).
select atom: ? :-father (Bert, x).
father (Adam, Bert). - not matching, backtrack
father (Adam, Brigit). - not matching, backtrack
father (Bert, Cecil). - success
continue from above:
? (grandfather (Adam, Cecil):- father (Adam, Bert), father (Bert, Cecil)).
no atom left, success.

Using the same facts and rules, we could also ask
? grandfather (x, Cecil).
or
? grandfather (x, y).
and receive the correct answers as Prolog rules express logic predicates which can be used with any variable instantiated and then returns with all combinations of variables, that follow from the database (invertibility).

This simplistic example should have made clear some of Prolog's most compelling features:
- no explicit control structure, only backtracking
- very high level, descriptive programming
Implementation of Prolog is not necessary slow; some advanced versions provide compilers and claims are reported that Prolog implemented programs may be faster than implementation using traditional programming languages [comp. graph. cad].
In the following we will assume the reader to be somewhat knowledgeable about the features of Prolog (for an excellent introduction to 'standard' Prolog see [Clocksin 1981), and how it might be applied to build expert systems. However, we do not think that all the extra-logical features of standard Prolog should be retained for the intended use of it. In fact, we will understand in the sequel by the word 'Prolog' or 'logic-programming language' (which shall be used as synonym), a pure, 'Horn clause only' logic based programming language, using the resolution principle and a depth-first, backtracking strategy. In [Turner1984][Brodie 1984] and [Frank 1984] a discussion of shortcomings of Prolog can be found; this is excluded from the present treatment of the topic.

4. Using a Database to Store Prolog Facts and Rules

Normally facts and rules are stored in the address space of the processor, either in real or in virtual memory (in the latter case possibly paged out on disk). Long term storage of facts and rules is in files that are read in on the users demand. Most actual systems are limited in the number of rules that can be used at a given time.
In order to build large systems, storage of rules on disk and transparent access to them from the Prolog interpreter are desirable. Using a database to provide these services is an obvious idea as dbms traditionally provide such functions to all types of programs. Not only would this free the user from distributing rules over different files and the obligation to read in the necessary rules for a problem, but also secure the data against accidential loss. Finally, most dbms permit concurrent access to data by several users at a time, a form of use highly desirable for building large expert systems, but presently (to my knowledge) not available on any knowledge system [Smith 1984].

This paper discusses a working prototype implementation of this idea. It does not implement the full Prolog language (as described in [Clocksin 1981), but a useable variant. The most important restriction is to single valued variables (no lists and structures, but including strings). There are several extensions, including built-in predicates for graphical output, and changes in the programmer interface to make the language more useful within the realm of programming large and complex software systems.

The goal of our work is very similar to work reported in [Sciore 1984], but the approach is quite different: we start with a dbms and augment it with a Prolog inference engine, wheras Sciore starts with a Prolog system that is to be expanded to provide typical dbms functions.

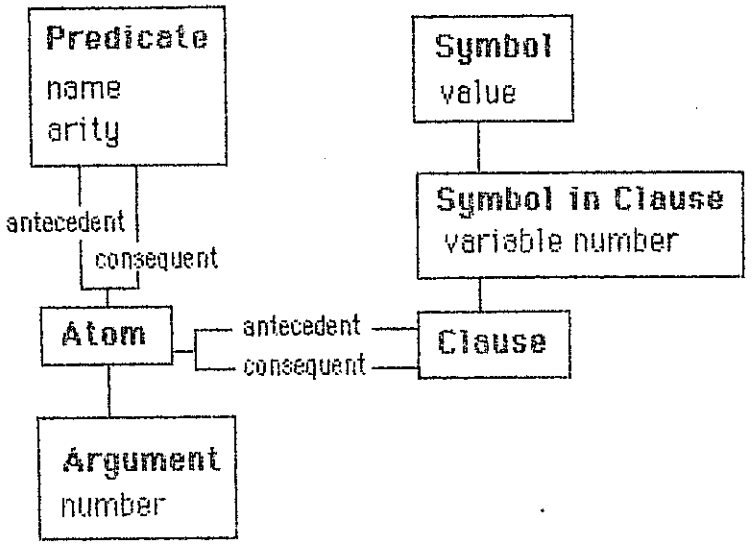
4.1 Datastructure for Storing Facts and Rules.

In order to store Horn clauses in a database we have to choose a datastructure. Eliminating repetitive groups and doing some additional normalization we can model them in the following relations:

PREDICATE: (predname, arity).
CLAUSE : (clau#).
ATOM : (atom#, predname, clau#, consequent/antecedent)
ARGUMENT: (atom#, NoInAtom, value, type).

There is obviously a problem with the attribute value in the ARGUMENT

navigational command to move from one entity to the next, is fairly simple. Writing recursive procedures - if the host programming language permits it - is simple as a present position in the navigation can be stored in a variable and be used for later continuation.
We have worked on the PANDA dbms [Frank 1982a] using an advanced type of network model, supporting generalization [Smith & Smith 1976] and data abstraction [Guttag]. It is completely written in Pascal and provides a Data Manipulation Language that is implemented as Pascal functions.



The data structure, equivalent to the relational schema given above, is now drawn as an extended box diagram.

Writing a Prolog interpreter is neither extremely difficult nor very simple [Battani 1983]. It contains a module for binding values from one clause to the next, which is of no interest to our present topic, and a module to traverse the and/or tree. Tree traversing routines are easily written in navigational data manipulation language. Our implementation concentrates all operations for data storage and retrieval in a few routines and could easily be adapted to other database management systems. Finally the traversal routines are either used recursively or an explicit stack management is necessary. We implemented the latter as it provides more flexibility as to what actions can be taken once a solution is found and how to continue to find the next.

The present prototype is of course slower than optimized, all in main memory systems (or compilers). It runs at several hundred inferences per

second on a heavily loaded, general purpose university IBM mainframe. Optimization of the code will certainly improve this figure, but is not of prime importance to us at the moment. Our interest is much more in highly structured code that is easy to extend and experiment with. We are especially interested in the requirements a Prolog interpreter demands from a dbms, and can report here some preliminary findings:

Requirements for a dbms suitable for a Prolog interpreter
- Fast access from one entity to another of a different type, as necessary for tree traversal operations
- Very large buffers for entities: an entity once used will be necessary much later (when recursion unrolls). The usual page oriented buffer strategy will not avoid reading in this entity from disk again. The PANDA dbms has a two level buffer: a first level managed in pages, the second in single entities. It is obvious that such a method can hold a vastely larger number of records of interest in a smaller space. Such a device was deemed necessary for use of the Panda dbms with geographical data and interactive graphical output, and generally for engineering [Frank 1982b].see also [Plouffe 1984].

5. Prolog as a Host Language for a DML

A different line from the previously discussed use of a dbms to store Prolog rules and facts is the use of Prolog as a host language for access to a database - Prolog as an intelligent front end to a database. The similarity between (relational) dbms and logic programming had been noticed in [Gallaire 1978] and [Kowalski 1979].
The problem is related to the previous one as in both cases a dbms is used to store facts that can be used from Prolog, but it is less general as it does not provide storage for Prolog rules.

5.1 Tight Coupling of Prolog Front-End with the DBMS

If a Prolog front-end is coupled with a dbms, either two separate systems may exchange intermediate results, or one integrated system may be produced.
Several implementations and projects are reported which combine Prolog with an existing relational database, most prominent the Japanese 5th generation project [Runifugi 1984] [Jarke 1984a] [Vassilou 1984] [Li 1984]

A method that is often proposed [Chang 1984] is to use the apparent similarity between the relational query languages and logic programming -

both are based on predicate logic, and treat data in relations as Prolog facts. A Prolog query could be executed by passing the relevant query to the dbms, a tuple request a time. This is clearly inefficient as Prolog's model of execution (depth first, backtracking) is the exact opposite of relational dbms's relation at a time logic. In order to improve performance and allow optimization, regrouping of antecedents in a Prolog query is proposed. This requires first that only pure logical predicates (without side effects, especially excluding the 'cut') are used. The Japanese fifth generation project plans to move all database predicates to the end of the query and execute them at the end - allowing for most extensive optimization. Unfortunately this does not always lead to correct results [Naqvi 1984].
Some systems use Prolog as an intelligent query optimizer [Jarke] [Vasilou] that uses semantic information about the database to improve a query that is ultimately passed to a standard dbms which then produces the answer. Similarly, Prolog can be used to provide more flexible and better adapted query languages which are then translated by Prolog into a form understandable for the dbms [Li 1984].

Combining two separate systems includes some overhead each time control passes from the one to the other. Our goal is to provide a fully integrated system where access to the database is part of and integrated into the execution of Prolog query. Ultimately, Prolog should then be used in our project for example for a reimplementation of MAPQUERY [Frank 1982c][Egenhofer 1984], a query language to retrieve geographic, spatial data from a database and produce map-like output. At present, however, we discuss only how a Prolog programmer can gain access to the database.

5.2 Dbms Access Operations from Prolog
[Zaniolo 1984b] observed the intimate similarity in execution strategy between a navigational dbms and Prolog. A query in Prolog proceeds from one antecedent to the next, using backtracking in case of failure. This is similar to writing a sequential procedural program in a traditional host language and using navigational DML statements.
A navigational interface to a dbms provides operations on two different levels, namely on single valued fields in the entities (records in CODASYL parlance), and on the entities as a whole, moving from one entity to the next in following one of the navigational paths (CODASYL sets). Any navigational language must in one form or another provide these two types of objects and operations on them.
Generally the entities are considered as Prolog facts with the entity name as a predicate and the appropriate number of constants. In order to move

between these facts seen as aggregation of values or seen as entities in their own right, Zaniolo proposes two different methods. The first using a binary predicate '@ ' to translate from the entity to a fact that is composed of the fields in the entity, or to translate in the reverse direction.
The second method is 'object-oriented' as there are methods defined that lead from one of the entities to anothe, using a navigational path. Syntactically he proposes an additional construction ':', that separates an entity predicate from a method, that unifies with another entity, related to the first by a path.

We opted for a method which needs less extension to the syntax of Prolog and stresses even more the navigational aspect of the data model. It may as well be claimed that navigation is a natural metaphor to express movement in the data space - human beings have been navigating in real space since their very early days, and it may be argued that they have a good sense for finding their way on road systems etc . This remark should not be taken too seriously - it is only to counter the abundant claims for 'naturality' in query language literature.

In order to accomodate an interface to the database, we extend the Prolog datatypes by a 'surrogate' datatype, i.e. a single value that stands for a dbms entity. There are no user available constants for such values, but variables may be used in the usual way.

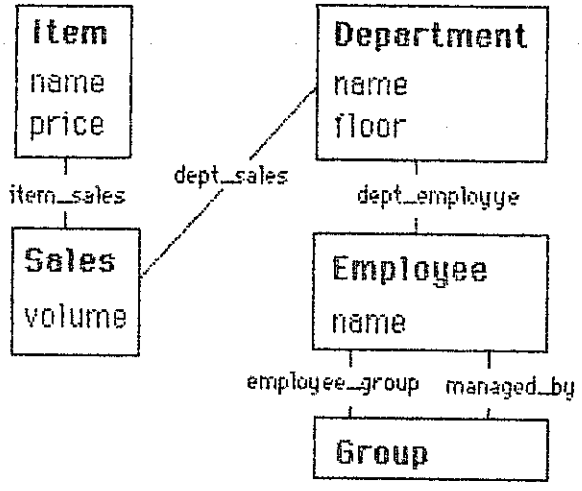
As a first level, which is not intended for human programmers, we implemented the following basic network DML predicates (very closely modelled after [CODASYL 1978]):

findMember (owner-entity, member-entity, set-name)
findOwner (member-entity, owner-entity, set-name).
findwithKey (entity, pathname, key-value).
findInSortedSet (member-entity, owner-entity, set-name, key-value).
giveField (entity, fieldname, value).

The use of these predicates is severely restricted as they work only if certain values are instantiated and they do not backtrack (except findMember which in turn produces all member-entities in the specified set). They are comparable to the basic computational predicates in Prolog which also have strict limitation to the number of instantiated variables present.

From this level we construct a second level of predicates which are better suited for programmers. They relate to the graphical description of the

database schema and provide navigation using the schema as a chart:



For each **field in an entity**, there is a predicate with this field name (attribute name) of the form

attribute-name (entity, value)

eg.

salesVolume (sales, number).

This predicate succeeds if the entity is instantiated and binds the value to the value of the respective attribute.
It also succeeds if the value is given and the attribute is defined as a key for accessing this entity. It fails if there is no entity with this value.

For each **set** a predicate of the form

set-name (owner-entity, member-entity)

is defined;

eg. dept_sale (dept, sales).

It succeeds, if any of the two entities is instantiated, or if both are instantiated and they are related in a set. If the owner is instantiated, it will succeed repeatedly if backtracked and produce all the members.
It fails if the instantiated entity is not member in any occurence of the named set or if none of the entities is instantiated.

Using this second level predicates, we show some of the standard examples of dbms literature using the schema given above.

Ex. 1: find all employees who work for departments on the first floor of ABC company.

References

Battani, G. et. al., mod-PROLOG, microcomputer oriented Prolog, in: Proceedings 1983 ACM Conference on Personal and Small Computers, SIGPC vol. 6, No. 2

Brodie, M., Jarke, M., Integrating Logic Programming and Databases, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Chang, C.L. and Walker, A. PROSQL : a Prolog programming interface with SQL/DS, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Clark, Gregory, PARLOG, Parallel Programming in Logic, Imperial College of Science and Technology. Research Report 844, London, 1984

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

CODASYL, Data Description Language, Journal of Development, 1978

Codd, E., Extending the relational database model to capture more meaning. ACM Transaction on Database Systems, Vol. 4, No. 4, December 1979

Colmerauer, A., Un système de communication homme-machine en français. Rapport, Groupe Intelligence Artificielle Université d'Aix-Marseille-Luminy, Marseille (France), 1973

Eggenhofer, M., Implementation of MAPQUERY, (in german),report No. ..., Institute of Geodesy and Photogrammetry, Swiss Federal Institute of Technology, Zurich (Switzerland), 1984

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

Frank, A., PANDA Pascal Network Database, (in german), report No. 62, Institute of Geodesy and Photogrammetry, Swiss Federal Institute of Technology, Zurich (Switzerland), 1982

Frank, A., Extending a Network Database with Prolog, in. L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Freeman, H., Ahn, J., Autonap - An expert system for automatic name placement, in: D. Marble et al. (Eds.), Proceedings of the International Symposium on Spatial Data Handling, Zurich (Switzerland), 1984

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

Gallaire, H., et al., Logic and Databases: a deductive approach, ACM

Computing Surveys, Vol 16, No. 2, June 1981
Goldberg, A., SMALLTALK-80: The Language and its Implementation . Addison Wesley, Reading (Mass.). 1983
Gonzalez, J., et al., Evaluation of the effectiveness of Prolog for a CAD application IEEE Computer Graphics. Vol. 4, No. 3, March 198434 .
Gougen, J., Thatcher, J., Wagner, E., An initial algebra approach to the specification, correctness and implementation of abstract data types in: Yeh, R. (Ed.) Current Trends in Programming Methodology, Prentice-Hall, Englewood Clifts, N.J. 1978
Gray, J., et al., The recorvery manager of the system R database manager, ACM Computing Surveys, Vol. 13, No. 2 , June 1981
Guttag, J; Horowitz, E., Musser, D., The design of data type specification, in: Yeh, R. (Ed.) Current Trends in Programming Methodology, Prentice-Hall, Englewood Clifts, N.J. 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
Jarke, M., and Koch, J,. Query Optimization in database systems, ACM COmputing Surveys, Vo.16, No. 2, June 1984
Kerschberg, L. (Ed.), Proceedings of the First International Workshop on Expert Database Systems , 1984 .
Kowalski, R. Logic for Problem Solving, North Holland, 1979
Kunifuji, S. and Yokota, H., Prolog and relational databases for 5th generation computer systems, Proc. Workshop on Logical Basis for Databases, Toulouse (France), 1984
Li, D. A PROLOG Database System, Research STudeis PRess, Letchworth, England/ John Wiley & sons, 1984
Meier, A., Lorie, R., A surrogate concept for engineering databases, in: Schkolnick, M., Thanos, C., (Eds.) Proceedings Ninth International Conference on Very Large Databases, Florenmce (Italy), 1983
Mylopoulos, J. An overview of knowledge representation, in: M. Brodie, S. Zilles (Eds.), Procedings of the workshop on data abstraction, data bases and conceptual modelling, Pingree Park, Co. , SIGMOD Record, Vol 11, No.2, 1981
Naqvi, S. A., Interfacing Prolog and relational databases: the problem of recursive queries, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984
Palmer, B., Symbolic feature analysis and expert systems, in: D. Marble et al. (Eds.), Proceedings of the International Symposium on Spatial Data Handling, Zurich (Switzerland), 1984
Parnas, D., A technique for software module specification with

examples, Commun. ACM, Vol 15, No. 5, May 1972

Plouffe, W, et. al, A database System for Engineering Desing, IEEE DATABASE Engineering, Vol. 7, No. 2, June 1984

Price, D., and Maier, D. Data model requirements for engineering applications, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems , 1984

Rehfuss, S., et al., Paritcularity in Engineering Data, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Robinson, J., A machine oriented logic based on the resolution principle, Journal of the ACM, Vol. 12, No. 1, Jan. 1965

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

Sciore, E. and Warren, D.S., Towards an integrated Database-Prolog system, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Smith, J., Smith, D., Database abstraction: aggregation and generalization, ACM Transaction on Database Systems, Vol. 2, No. 2, June 1975,

Smith, J., Expert database systems: a database perspective, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Stonebraker, M., Guttman, A., Using a relational database management system for computer aided design data - an update, IEEE Database Engineering, Vol. 7, No. 2, June 1984

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

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

Zaniolo, C., An implementation of GEM - supporting a semantic data model on a relational back-end, in : B. Yormark (ed.), Proceedings SIGMOD'84, SIGMOD Reecord, Vol. 14, No. 2, 1984

Zaniolo, C., Prolog: a database query language for all seasons, in: L. Kerschberg (Ed.), Proceedings of the First International Workshop on Expert Database Systems, 1984

Zaniolo, C., Object-oriented programming in Prolog, Int. Logic Programming Symposium, 1984

Publications

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 donnees pour les systemes 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 raumliche Datenstrukturen in Land-informations-systemen (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, (co-author: D. Humphrey, Senior in Surveying Engineering).

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, vol 12, pp 677-680, November, 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.

45. Combining a Network Database with Logic Programming, A. Frank.

46. Montgomery County (PA) GPS Survey, A. Leick and J. Collins, ASP/ACSM Annual Meeting, Washington, D.C., March 10-15, 1985.

47. Analysis of Macrometer Networks with Emphasis on The Montgomery County Survey, A. Leick and J. Collins, First International Symposium on Precise Positioning with the Global Positioning System, Rockville, Maryland, April 15-19, 1985.

48. Application of GPS in a High Precision Engineering Survey Network, R. Ruland and A. Leick, First International Symposium on Precise

Positioning with the Global Positioning System, Rockville, Maryland, April 15-19, 1985.

49. Graphics Programming in Prolog, R. Michael White and Andrew U. Frank

50. Instrumentation Needs (GPS and Related Matters), Alfred Leick. Workshop on Fundamental Research Needs in Surveying, Mapping, and Land Information Systems, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, November 18-20, 1985.

51. Surveying Engineering Annual Report, 1984-85.

52. Expert Systems Applied to Problems in Geographic Information Systems, Vincent Robinson, Andrew U. Frank, Matthew Blaze, presented at the ASCE Specialty Conference on Integrated Geographic Information Systems: A Focal Point for Engineering Activities, February 3-5, 1986.

53. Integrating Mechanisms for Storage and Retrieval of Data, Research Needs, Andrew U. Frank, challenge paper for the 'Workshop on Fundamental Research Needs in Surveying, Mapping, and Land Information Systems', held November 17-19, 1985, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061

54. Formal Methods for Accurate Definitions of Some Fundamental Terms in Physical Geography; Andrew U. Frank, Bruce Palmer, Vincent B. Robinson. Invited paper at the Second International Symposium on Spatial Data Handling, July 5-10, 1986, at Seattle, WA.