Panoramas and Grammars: A New View of Data Models

Kenneth Baclawski[*]

April 21, 1993

Abstract

We show that grammars can be regarded as defining a kind of data model and that one can automate the translation from semantic data models to grammars and vice versa. The semantic and structural distinctions between grammars and data models are discussed in detail. In general, only part of the data in a database can be represented in grammatical terms. We introduce the concept of a *panorama* for the subset of the database that can be represented grammatically. The computation of a panorama is a key step in the translation from a data model to a grammar, and an algorithm for computing it is presented. One consequence of the automation of the translation between grammars and data models is to automate the development of symbol table software.

1 Introduction.

At first glance there appears to be little similarity between data models and grammars. Data modeling is a technique for organizing the data in a database. A grammar, on the other hand, is a way of defining a language. The purpose of this paper is to show that there are, in fact, many similarities between grammars and data modeling. Indeed, grammars can be regarded as defining a certain kind of data model, and one can automate the process of translating between data models and grammars.

The idea that a data model can be grammar-based is not new. In any data model, after all, it is necessary to have a language for expressing the structure or *schema* of a particular database. However, the development of this language is not usually considered to be a data modeling task. The language is only a means of expressing a schema. Similarly, the development of a schema for a database is not usually regarded as a linguistic task. One exception is the "p-string data model" of [10] which is an important precursor to our own

[*]Northeastern University, College of Computer Science, Boston, Massachusetts 02115. Keywords: Data model, semantic data model, context-free grammar, data model translation, symbol table, class library.

work. Another exception is the Demeter^{TM} system [11], [12] which uses a grammar-based approach to designing the classes of an object-oriented programming system.

In this paper we discuss grammars (or more precisely *context-free grammars*) from a data modeling point of view. We show that it is possible to treat a grammar as a schema, and to some extent, schemas as grammars. Furthermore, we discuss how to automate the process of translating from a grammar to a schema and vice versa as well as from a sentence of a language to a database state and vice versa.

The semantic data model that we will use is the ER model, extended to included inheritance. Grammars will be written using the same style as the yacc compiler-compiler. Both of these are discussed in section 2.

The relationship between grammars and schemas is discussed in detail in section 3. For each major linguistic concept, there is a corresponding data modeling concept. At some point, of course, the analogies begin to break down. Much of the discussion concerns distinctions between grammars and schemas.

For example, grammars implicitly specify a great deal of sequential information, while schemas are more flexible in this respect. One can include sequential information or not in a schema, but one cannot define an unordered set of items in a grammar. One step in translating from a schema to a grammar is to introduce linearity if it is not already specified by a process known as *linearization.*

Another distinction between grammars and schemas is that grammars have a preferred starting point which corresponds to the root node in the parse tree of a sentence. Most data models do not have such a distinguished entity. For a schema to be translatable to a grammar, one must choose this preferred starting point. We call this the *center* or *focal point* of the schema. Given a center, it is possible to determine how much of the schema can be viewed in a "grammatical" fashion from this point. Such a view is called a *panorama* or *panoramic view* of the schema. It is the panoramas that can be translated to grammars. Panoramas and the algorithm for computing them are discussed in section 4.

Still another distinction between grammars and schemas is the concept of a key or identifier in a schema. This concept is not expressible in a context-free grammar, but is nevertheless an important aspect of most languages. It is generally implemented in an ad hoc manner using a "symbol table." The translation from a schema to a grammar must therefore automate the production of symbol table software. In our system the symbol table software is a flexible and powerful C++ class library that can be used for much more than just parsing sentences.

Some of the other aspects of data models and grammars are discussed in section 5, and some of the directions for future work are mentioned in the concluding section.

2 Context-Free Grammars and Semantic Data Models

The notation for grammars that we will use is a simplified form of the standard yacc grammar of the Unix^{TM} Operating System. Since we do not use any of the special features of yacc,

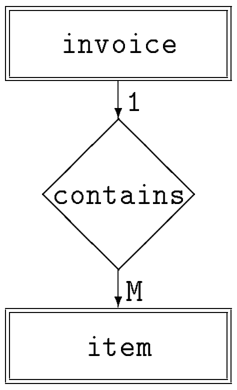

Figure 1: `Invoice` ER Diagram

our discussion should apply to any grammar notation.

A **context-free grammar** [1] consists of a set of symbols divided into the **nonterminal** symbols and the **terminal** symbols, and a set of **rules** (or **productions** or **rewriting rules**) each of which specifies how a nonterminal symbol can be expressed in terms of a sequence of symbols. In addition, one nonterminal symbol is designated as the **start** symbol. Each nonterminal symbol must have at least one rule, and we will assume that the grammar is unambiguous. In a compiler-compiler like yacc, one also specifies an **action** to be performed whenever a rule is used. The actions will not be shown explicitly in any of the examples, but the role of actions will be discussed.

A **data model** is an intellectual tool for interpreting data and the interrelationships among items of data. To keep from becoming unwieldy and inefficient, most data models restrict the scope of what kind of data can be represented and interpreted. **Semantic networks** were developed by researchers in artificial intelligence as a means of representing and organizing general knowledge of the world. **Semantic data models** incorporate ideas from both traditional data models and semantic networks.

The data model we will use is the ER model [6], extended to include inheritance. Our implementation of this data model is part of the `nu&` data modeling tool [4], [2]. This system is a vehicle for both research and education on such tools. It is concerned with studying ways to enhance the modeling power of a transaction management system while maintaining its high-performance characteristics. The name is a reference to Northeastern University where most of the work is being carried out.

Objects in the ER model are called **entities**, and they are grouped into classes called **entity sets**. Objects can be related to one another by links called **relationships**, which are grouped into **relationship sets**. An ER schema is usually written using a diagram called the **ER diagram** in which entity sets are drawn using rectangles and relationship sets are drawn using diamonds.

In figure 2, we show an example of an ER diagram. The diagram represents the schema of a classic business application: the storage of invoices. In this case there are two entity sets: invoice and item. Each invoice consists of one or more (line) items, and each item is used in exactly one invoice.

In the ER model, one can restrict the functionality of relationships to be one-to-one, one-to-many, many-to-one or many-to-many. These constraints are indicated using decorations of the basic ER diagram. For example, the one-to-many relationship set linking invoice with item in figure 2 is decorated with “1” and “N” to indicate its functionality.

In addition, an entity set is called a **weak entity set** if the existence of each entity in this entity set depends on the existence of another entity in the database. Although, strictly speaking, this is a property of relationship sets and not entity sets, the ER diagram indicates this constraint by using a “double” rectangle for the entity set. For example, in figure 2, both invoice and item are weak entity sets because both depend on the other for their existence.

Note that an ER diagram is not a complete schema for a database. In particular, the attributes of the entity and relationship sets are not normally shown in the ER diagram nor are the constraints presented in a precise manner (the weak entity constraint being an example). For this reason, we distinguish between the ER schema and the ER diagram. The syntactic details of the schema are beyond the scope of this paper, and we refer the reader to [2].

3 The Relationship between Grammars and Schemas

In this section the relationship between grammars and schemas is introduced. An example is used throughout the section to illustrate how linguistic concepts have analogous data modeling concepts and vice versa. As the example is examined in more detail, the distinctions between grammars and schemas emerge, leading to the concepts of linearization, delinearization and panoramas.

The following grammar defines a language in which a sentence specifies zero or more invoices, each of which consists of one or more items. In other words, essentially the same information content as in the ER diagram of figure 2.

```
%start invoice_list
invoice_list : invoice_list invoice | ;
invoice : string '{' item_list '}';
item_list : item_list ';' item | item;
item : string;
```

Each invoice has a “header” consisting of a string, and each item consists of a string. The braces are punctuation marks that determine which rule is to be used, but that convey no additional information once it has been determined which rule to use.

This example suggests that one can relate grammars to schemas as follows:

1. Nonterminals of the grammar correspond to entity sets of the schema.
2. If a terminal of the grammar has a value associated with it, then the terminal corresponds to a domain or built-in type of the schema. The use of the terminal in a rule

is represented as an attribute of an entity set. However, an attribute in a schema is identified by its attribute name, while a symbol in a grammar rule is identified by its position in the rule.

3. If a terminal of the grammar is just a punctuation mark, then the terminal corresponds to a domain with just one value. The use of the terminal in a rule need not be represented as an attribute.

4. If a nonterminal has several rules, then each rule represents an entity set that inherits from the entity set for the nonterminal. The entity set of the nonterminal has as its attributes the common attributes of the alternatives.

5. When a nonterminal is used in a rule, then this use is represented by a relationship set in the schema. As with attributes, relationship sets in a schema are identified by their names, while symbols in a grammar rule are only identified by their position.

6. A nonterminal can appear on both sides of a rule. In some cases, such as "left recursion," this represents ordering or sequential information. Such information may not be needed and can be omitted from the schema.

7. A sentence (or equivalently, a parse tree) of the grammar corresponds to a state of the database.

8. The action performed by the compiler when each rule is invoked corresponds to the action (constructor) to be performed when an entity is constructed. The symbols used in the rule represent the entities passed as parameters to the constructor of the entity represented by the left-hand side of the rule.

The process of translation between grammars and schemas will include provisions for improving (or "optimizing") the translation. Optimization for data model translations is similar to optimization for programming languages: if an optimization pattern is observed, then the schema is transformed in a specified way. Ideally an optimization should not alter the semantics of the schema. However, there are circumstances where optimizations that alter the semantics are desirable.

In the case of the Invoice grammar, the translation has performed two optimizations. The first is the elimination of sequential ordering information. This optimization changes the semantics. The second is the elimination of two grammar symbols so that there are only two entity sets in the schema, not four as one would expect. This does not change the semantics because the following grammar defines the same set of sentences as the one above:

```
%start invoice_list
invoice_list : invoice_list string '{' item_list '}' | ;
item_list : item_list ';' string | string;
```

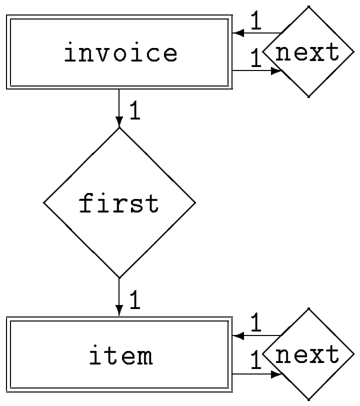

Figure 2: `OrderedInvoice` ER Diagram

As already noted, the Invoice schema and grammar above differ in an important respect: there is no ordering among the invoices (or among the items of one of the invoices) in a database state. Figure 3 is the ER diagram of a schema that maintains sequential information in the invoice database by using two one-to-one relationship sets named `next`. In addition the one-to-many `contain` relationship set becomes the one-to-one `first` relationship set. In the OrderedInvoice schema, the `item` entities are grouped into linked lists, and only the first `item` in each list is linked with an invoice. There are several constraints of the schema that do not appear in the ER diagram. These constraints ensure that there is just one list of invoices and that each item is contained in the list of exactly one invoice. These constraints are called disjointness and covering constraints.

The implementation of the ER model in the `nu&` system is rich enough to allow one to express constraints of the kind required above. Yet all constraints are built from just a half-dozen elementary constraints. The ER model of `nu&` is called the *alpha model.* In this model one can express data structures that use pointers, garbage collection, encapsulation, overloading and identifiers that depend on a local scope. For more information about the `nu&` system see [2].

Another important distinction between grammars and schemas is that grammars have a preferred starting point which corresponds to the root node in the parse tree of a sentence. For a schema to be translatable to a grammar, one must choose this preferred starting point. This choice leads naturally to the concept of a panorama which is developed in the next section.

4 Panoramas

A **view** (or **derived entity set**) of a schema consists of:

1. A subset of an entity set (or another view). This entity set is the **center** or **focal point** of the view.

2. A subset of the attributes of the focal point.

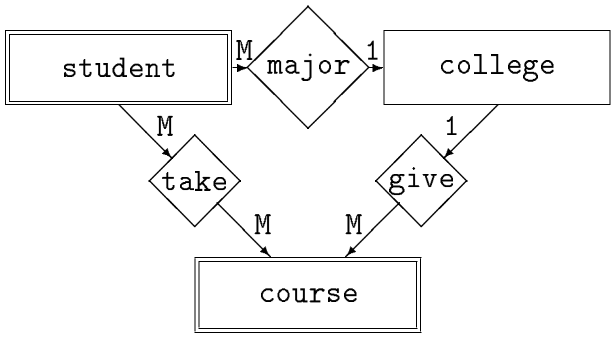

Figure 3: `University` ER diagram

3. Any number of **derived attributes** computed using relationship sets and attributes of other entity sets (or views).

A **panorama** is a maximal nonredundant view of the database. In this section we give some examples of panoramas and discuss how a panorama is computed.

Perhaps the most common example of a panorama is a set of data items stored as a flat ASCII file and not tied to any particular database management system. Such a database will be called a *peripatetic* database. Address lists, BibTEX files and structured text as in a dictionary are examples of peripatetic databases. Translating a schema to a grammar allows the database to be "dumped" as an ASCII file and loaded into another database which might use a different schema, data model and database management system. Having a peripatetic form makes a database more easily transportable. As we will see, a panorama is essential for translating a schema to a grammar and hence for making a database peripatetic.

The p-string ("parsed string") data model [10], [14] develops a grammar-based data model for structured text. This model is similar to ours in that grammars are regarded as defining a data model. Furthermore, the p-string query language has an operator that reparses a string. In other words, the operator dumps a database state and reloads it using another schema. However, the p-string model differs from our work in not considering the issues of identifiers and binding.

The schema of figure 4 will be used to illustrate the computation of a panorama. The computation of a panorama for the University schema presents a number of problems. Suppose that we use the student entity set as the center. Each student majors in exactly one college. However, a given college can have any number of student majors. If we view the college entity set from the student entity set, then information about each college will be replicated for each student record. Furthermore, there might not be any student majors in some college, in which case the information for that college would not be represented at all. Similar remarks apply to the course entity set relative to the student entity set. It follows that the panoramic view centered at the student entity set consists of just the student entity set itself. The panoramic view from the course entity set is also just the entity set itself.

Finally consider the college entity set as the focal point. The college entity set is linked with the student entity set via the `majors` relationship set and with the course entity set via the `give` relationship set. In both cases the relationship set is single-valued and mandatory.

In other words, a course *must* be given by exactly one college, and a student *must* major in exactly one college. Therefore both students and courses can be represented fully and nonredundantly from the point of view of the colleges. One can translate everything in the University schema except the `takes` relationship set into the following grammar:

```
%start college_list
college_list : college_list ';' college | ;
college : ... '{' student_list '}' '{' course_list '}';
student_list : student_list ',' student | ;
student : '(' ... ')' ;
course_list : course_list ',' course | ;
course : '(' ... ')' ;
```

It remains to consider the `take` relationship set. To represent this relationship set, either the courses taken by each student must be listed or the students taking each course must be listed. However, it would be highly redundant (and very error-prone) to require that all the information about a course (or student) be repeated in these lists. The usual solution to this problem is to use an identifier for an entity in lieu of repeating all the information about the entity. This will only work if there is such an identifier among the attributes of the entity. Assuming that this is the case for the course entity set, then the University schema can be translated into following grammar:

```
%start college_list
college_list : college_list ';' college | ;
college : ... '{' student_list '}' '{' course_list '}';
student_list : student_list ',' student | ;
student : '(' ... '[' course_title_list ']' ')' ;
course_list : course_list ',' course | ;
course : '(' ... ')' ;
course_title_list : course_title_list ',' string | ;
```

An attribute (or set of attributes) of an entity set that uniquely determines each entity is called a **key** or **identifier** of the entity set. Data models without object identity (such as the relational model) require the introduction of identifiers for every concept represented in the database. Object-oriented data models are more reasonable in this respect, but still require identifiers in some situations.

Unfortunately, the introduction of identifiers takes one beyond the range of what can be expressed entirely within a context-free grammar since the grammar itself does not specify that the strings occurring in `course_title_list` have any connection with the strings identifying courses. For that matter, there is no way to express within a context-free grammar that an attribute uniquely identifies an instance of a nonterminal. Both of these features of the schema are considered "semantic" rather than "syntactic" features of a language.

The use of identifiers in a grammar is called *binding* in the linguistics literature [7], [8]. Binding involves three word-classes: anaphors (including self-references like the word "itself"), pronomials (including pronouns like the word "them"), and referring-expressions (including proper nouns). All of these occur in programming languages. For example, the "this" variable in C++ can be regarded as anaphoric. However, the only strategy that we will discuss is the use of referring expressions.

The linguistics literature deals very well with binding, especially with the way that one looks up or "indexes" a reference in a natural language. However, there is no consideration of why the identifiers exist in the first place and when one might be able to do without them. The relationship between grammars and schemas discussed here presents a simple model that explains the circumstances that lead to the introduction of identifiers.

In the case of yacc or Demeter^{TM}, looking up an identifier is handled by the actions of the grammar. They are traditionally implemented by using a data structure called a *symbol table*. The translation of a schema to a grammar will produce a lexical scanner, grammar, the grammar actions, and a C++ class library that implements a symbol table and provides facilities for enforcing the constraints specified by the schema. The C++ classes come complete with facilities for garbage collection, tracing, displays, persistent object storage, and so on. The class library can be used as the foundation for an object-oriented database management system.

If a grammar uses identifiers, then the requirements for these identifiers are buried in the action code. As a result, when translating a grammar to a schema, it is currently not possible to translate these uses of identifiers into relationship sets in the schema. The need for identifiers is part of the more general issue of "object sharing" (or "instance sharing"). Objects can be shared in a number of ways:

1. A many-to-many relationship set between entity sets A and B permits many entities of B to be related with (and hence shared by) a single entity of A and vice versa.

2. Two relationship sets having a entity set in common can result in sharing even if the relationship sets are not many-to-many. In the OrderedInvoice schema, the `first` and `next` relationship sets would potentially allow sharing of `item` entities. Since this does not occur in the grammar, it was necessary to impose a constraint to prevent it.

The rest of this section is devoted to describing how panoramas can be computed. This is done in several steps. In the first step, the schema is translated to a more elementary data model called the *mu model* [5]. In this model, relationship sets, inheritance, attributes and even key constraints are all represented using binary relationship links. This model is not very useful for data modeling by a person, but it is very useful for implementing algorithms because of its simplicity. For a succinct description of the mu model see [3]. In the mu model, the schema is a directed graph in which both vertices and edges are labeled.

The next step in computing a panorama is a "greedy algorithm" in which one starts with the central entity set of the panorama, and one adds edges and vertices whenever certain criteria are satisfied. The key fact is that whenever it is observed that an edge or vertex should be added to the panorama, there is never an advantage to delay this operation. The examples and discussion above illustrate the kind of criteria that must be satisfied.

When no more edges or vertices can be added to the panorama, the results are translated back into the terminology of the alpha model. They are then used to define a view that can be translated to a grammar. The entire process of translating a schema to a grammar can be summarized in the following steps:

1. Choose an entity set to be the center.
2. Compute a panorama at that center:
   (a) Translate to the mu model.
   (b) "Greedy algorithm."
   (c) Translate back to original model.
3. Linearize and optimize the panorama.
4. Express the linearized panorama in lexical and grammatical terms.
5. Produce a C++ class library for each entity set.

5 Query Languages

There are still other ways in which data models and grammars differ. These are not so much formal distinctions as they are differences in the way the concepts are generally used.

One distinction is that data models generally have well developed query languages, while grammar-based models generally have limited support for queries. Most grammar-based systems (such as yacc) only provide for an action to be performed whenever a rule is used. Generally the actions construct the symbol table, along with a list of statements in an intermediate language. Compiler-compilers like yacc generally offer little help with the symbol table. The programmer typically constructs the symbol table in some ad hoc manner rather than utilizing standard tools like database management systems.

One exception is the p-string data model [10] which includes a query language that has powerful operators for manipulating and querying parsed strings. However, the model is not concerned with issues of type and consistency constraints since the operators can manipulate the parsed strings in essentially arbitrary ways. Nevertheless, one of the directions for our future research is to study the extent to which one can express the p-string operators in our framework.

Another exception is the ACORN system of Reiss, described in [15]. Our work is similar to that of Reiss in that we automate the production of symbol table software. However, we can *start* with the structure of the symbol table and produce the grammar for it, rather than just starting with the grammar and producing a symbol table structure for it. On the other hand, the symbol table in the ACORN system is just a means to the ultimate goal of translation to the target language, whereas in our case, the symbol table software is our ultimate goal.

In most compilers the relationship between the program being translated and its symbol table is very complex and difficult to extract from a compiler. As a result the interfaces to standard libraries must be recompiled from scratch for each program, a task that can dominate the compile time in a language like C++. It would certainly make more sense to store standard library interfaces in persistent symbol tables which can be queried as needed rather than continually recompiling the interfaces. Perhaps the reason why this has not been done is that data models have not been rich enough to express the semantics of symbol tables in a way that made a significant improvement over ad hoc techniques. This may change with the development of object-oriented data models.

Another way in which grammars differ from schemas is the fact that database management systems typically deal with very large databases that are incrementally modified rather than being continually reconstituted. Compilers, on the other hand, generally compile a program as a single unit each time the compiler is invoked. Linkers do improve the situation to some extent, but this technology is old, and the granularity of compilation is still relatively large.

The fact that databases are generally modified in small units while compilers deal with large programming units has led to different strategies for specifying input and output. Compilers utilize lexical scanners and can specify punctuation marks in the grammar. Database management systems, on the other hand, have elaborate tools for designing screens for input and reports for output. The `nu&` system helps to bridge the gap between the services provided by database management systems and compiler-compilers.

6 Conclusion

We have shown that context-free grammars and schemas have a close connection, and we have discussed several strategies for translating between schemas and grammars. Some of these strategies have already been implemented in the `nu&` data modeling tool.

Although schemas in general cannot be translated to grammars, even when one is allowed to reference identifiers, it is possible to characterize how much of a schema can be so translated. We introduced the concept of a *panorama* for a maximal view that can be translated to a grammar. We also discussed how panoramas can be computed, and how panoramas can be translated to grammars. We introduced the idea that translations can be optimized and gave some examples of optimizations.

A significant application of this work is the connection with symbol table software. One product of translation from a schema to a grammar is a class library that implements a symbol table for the language defined by the grammar. This has advantages for compiler technology by making it possible to use database management systems for storing interface information.

The `nu&` system itself takes advantage of the automation of symbol table development. It was bootstrapped by building a minimal system that provided only some of the capabilities that were planned. This was written using standard yacc and lex techniques. This early system was used in several courses and by a series of graduate students who did projects using this system. A great deal was learned about how to proceed with the system from this early experience. The early system was then used to build the yacc grammars and C++ class libraries for more sophisticated models as well as other kinds of translators.

Most of the current work on the `nu&` system involves the development of translators of various kinds both at the data level and at the schema level. Languages being studied include the p-string query language mentioned earlier, Demeter^{TM}, SGML (Standard Generalized Markup Language) [9], SQL, LEGO (a proof checking system) [13] and BibTEX. The ability to construct complex C++ class libraries quickly and accurately has the potential for simplifying and improving the development of software for translation.

7 Acknowledgements

The author would like to thank Raoul Smith and Markku Sakkinen for their help with references to the literature on linguistics and grammar-based data models. The author would also like to thank the many graduate students, too numerous to list, who contributed to the `nu&` project.

References

[1] A. Aho and J. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.

[2] K. Baclawski. The `nu&` object-oriented semantic data modeling tool: intermediate report. Technical Report NU-CCS-90-18, Northeastern University, College of Computer Science, 1990.

[3] K. Baclawski. The structural semantics of inheritance. Technical Report NU-CCS-90-23, Northeastern University, College of Computer Science, 1990.

[4] K. Baclawski, T. Mark, R. Newby, and R. Ramachandran. The `nu&` object-oriented semantic data modeling tool: preliminary report. Technical Report NU-CCS-90-17, Northeastern University, College of Computer Science, 1989.

[5] K. Baclawski and D. Simovici. An algebraic approach to databases with complex objects. *Information Systems*, 17(NU-CCS-90-14):33–47, 1992.

[6] P. Chen. The entity-relationship model – toward a unified view of data. *ACM Trans. on Database Systems*, 1:9–36, 1976.

[7] N. Chomsky. *Lectures on Government and Binding.* Foris, Dordrecht, 1981.

[8] V. Cook. *Chomsky's Universal Grammar: An Introduction.* Basil Blackwell, Oxford, UK, 1988.

[9] Anonymous et al. Information processing – text and office systems – standard generalized markup language (SGML), 1986. ISO 8879-1986 (E).

[10] G. Gonnet and F. Tompa. Mind your grammar: a new approach to modelling text. In *Proc. 13^{th} VLDB Conf.*, pages 339–346, Brighton, UK, 1987.

[11] K. Lieberherr. Object-oriented programming with class dictionaries. *J. on Lisp and Symbolic Computation*, 1:185–212, 1988.

[12] K. Lieberherr and A. Riel. Contributions to teaching object-oriented design and programming. In *Proc. OOPSLA*, October 1989.

[13] Z. Luo, R. Pollack, and P. Taylor. *How to Use LEGO (A Preliminary User's Manual).* Edinburgh, UK, October 12, 1989.

[14] H. Mannila and K. Raiha. On query languages for the p-string data model. In Ohsuga Kangassalo and Jaakkola, editors, *Information Modelling and Knowledge Bases.* IOS Press, 1990.

[15] S. Reiss. Automatic compiler production: the front end. *IEEE Trans. Software Engineering*, SE-13:609, 1987.