Using Learned Dependencies to Automatically Construct Sufficient and Sensible Editing Views

Jeffrey C. Schlimmer (schlimmer@eecs.wsu.edu)

School of Electrical Engineering & Computer Science, Washington State University, Pullman, WA 99164-2752, (509) 335-2399, (509) 335-3818 FAX

Received: 16 March 1993
Revised: 29 April 1993

Keywords: functional dependency, determination, relational key, electronic form, database.

Abstract

Databases sometimes have keys besides those pre-planned by the database designers. These are easy to discover given functional dependencies in the data. These superfluous keys are convenient if a user wishes to add data to a projection of the database. A key can be chosen that minimizes the attributes the user must edit. In a list format view, enough attribute columns are added to those specified by the user to ensure that a key is present. In a form view, enough extra text boxes are added. In this latter view, functional dependencies may also be used to visualize the dependencies between attributes by placing independent attributes above dependent ones. This paper briefly reviews an algorithm for inducing functional dependencies, and then it demonstrates methods for finding keys, constructing list views, and laying out form views.

1. Introduction

The relational database model requires a key to uniquely identify entries in the database. Typically, during database design, one or more attributes are specified that will serve as the key. This paper explores the possibility that there may be multiple keys, some unanticipated by the database designers. Like the pre-specified key, these additional keys may be useful for database indexing or for conversion to a normal form. These redundant keys are also convenient if the user wishes to add entries to a projection of the database. By searching for a key that most closely matches the attributes requested in the projection, the system allows the user to edit a view that includes the minimum number of additional attributes. In the slowly emerging field of electronic forms, users may wish to automatically generate a form corresponding to a projection of a database.

Identifying all keys in a database is straightforward given information about functional dependencies between attributes. A functional dependency simply states that it is possible to compute values of a datum for a set of attributes (the function's range) given values of that datum for another set of attributes (the function's domain). Trivially, any attribute is functionally dependent upon itself. If a set of functional dependencies includes all defined attributes in a domain or range (or both), then the union of their domains is a key. Values for all other attributes can be computed from the key.

When a user wishes to add entries to a projection of the database, functional dependencies can assist in a second way. In addition to identifying a minimal key that includes the user-specified attributes, dependencies can be used to construct a form view so that independent attributes appear above those that depend upon them. This simple data visualization is easily computed by

topologically sorting attributes using the dependencies. If the new view is represented as an electronic form, it naturally displays dependency information from the underlying database.

This paper describes algorithms to support these types of information processing. First, it reviews an algorithm for inducing functional dependencies directly from data. Second, it describes a search process for finding a minimal key that includes a set of attributes. Third, it describes an algorithm for sorting the attributes and constructing a form view. These ideas are demonstrated in an implemented system using sample databases.

2. Learning Functional Dependencies

Determinations are one kind of functional dependency that has been studied in artificial intelligence (Davies & Russell, 1987; Russell, 1986). The prototypical example determination is that the country in which a child is raised determines their first language spoken. While not universally true, this example illustrates the basic functional nature of a determination. It is a many-to-one mapping. Each value of the domain element (in this case, possible countries in which a child could be raised) maps to at most one range element (a language that could be spoken). Children raised in France speak French; children raised in England speak English. Generally, the reverse is not true — several domain elements may map to the same range element. Children in many countries speak French.

As an example, consider the database listed in Table 1. The attribute A determines B because each value of A maps onto a single value of B. The reverse is not true because the value of i for B maps onto both i and iii for A (tuple numbers 1 and 3). For an example of a larger determination, note that attributes B and C in Table 1 conjointly determine attribute E; every pair of values of $B \times C$ maps to exactly one value for E.

Table 1. Simple example database. Rows in the table correspond to data. Columns correspond to attributes. Values appear in cells. In this data, A determines B, and $B \times C$ determines E.

	Attributes				
Tuple Number	*A*	*B*	*C*	*D*	*E*
1.	i	i	i	i	i
2.	ii	ii	i	ii	i
3.	iii	i	ii	iii	ii
4.	iv	ii	ii	iv	iii
5.	i	i	i	iv	i
6.	ii	ii	i	iii	i
7.	iii	i	ii	ii	ii
8.	iv	ii	ii	i	iii
9.	i	i	iii	i	i
10.	ii	i	iii	ii	i

Nothing is particularly difficult about discovering determinations in a database; the only obstacle is that there are so many possibilities. Because any combination of attributes could determine a single

attribute, if there are m attributes, there are 2^{m-1} possible determinations. A new algorithm for finding all determinations searches breadth-first from smaller to larger domains (Schlimmer, 1993) as follows: Given a single attribute as the proposed range, the search begins by testing the empty domain. This corresponds to hypothesizing that the range has a constant value. If this is not the case, the search considers all domains of size 1 then domains of size 2 and so on as necessary. The search is iterated in turn for all attributes, each serving as a singleton range. After the searches are complete, determinations with identical domains are combined by taking the union of their ranges.

This search algorithm relies on three strategies for coping with the exponential number of possibilities. First, search is constrained to consider only domains of k or fewer attributes, where k is some user-supplied constant. This reduces the number of possibilities to $(m-1)^k$. Second, breadth-first search is used because by starting with the empty domain, it finds smaller determinations first. This is important because any determination can be extended by adding attributes to its domain. Inference methods that use determinations benefit if the determinations have minimal domains and do not require unnecessary attributes. Third, when a determination is discovered, the algorithm avoids examining any states that are supersets of this determination's domain. If a simple determination is found, then the savings from this third strategy can be significant.

Unlike the first two strategies of this algorithm, it is considerably more difficult to implement the third strategy. First, the algorithm relies on a pruning rule: if a determination is found with domain D, there is no need to search any set of attributes that is a superset of D. (They will all be determinations.) Second, to capitalize on this pruning information, the algorithm uses a perfect hash function to record pruning advice when a determination is found. When a new candidate determination is to be considered by the search, the hash function is consulted. If this state has been marked as pruned by any of its subsets (because one was a determination), the state is discarded. Otherwise, the state should be examined. The space complexity of the hash function and breadth-first queue is $O(m^k)$, and the overall time complexity of the search is $O(n \cdot k \cdot m^{k+1})$, where n is the number of tuples (data), m is the number of attributes, and k is the user-supplied bound on the size of a determination's domain. This non-trivial process can be precomputed for a relatively static database. A dynamic database would require incremental updates for a set of determinations. For a ideas of how to do this, and for details of the generation of states and the hash function, see (Schlimmer, 1993).

This algorithm was applied to the four databases described in Table 2. For each database, the table lists the number of attributes, the number of entries (or tuples), and the number of determinations learned by the algorithm with a size bound of $k = 4$. To give an intuitive sense of the complexity of the result, the final column lists the time required for a Lisp implementation of the algorithm on a MC68040-based machine.

Table 2. Summary of four databases and number of determinations learned with $k = 4$.

Database	Attributes	Tuples	Determinations	Time (min.)
Breast Cancer	11	699	31	74
Bridges	13	108	23	13
CPU Performance	10	209	38	17
Zoological	18	101	63	39

3.1. Related Work on Learning Functional Dependencies

Ziarko (1992) discusses a representation idea identical to determinations in the context of database dependencies. Using set-theoretic notions, Ziarko develops the notion of a "full functional dependency" which is precisely a determination. Given a range of interest, Ziarko goes on to identify the usefulness of minimal domains termed "reducts." Ziarko also defines the "significance factor" of a particular database attribute as the relative degree of decrease in functional dependency as a result of removing the attribute from a domain. This paper utilizes Ziarko's observations about minimality and presents an efficient search algorithm to induce all reducts given a range of interest and a set of data.

The principal of minimality has also been examined by Almuallim and Dietterich (1991). They note that many learning situations involve a large number of irrelevant features. In these cases, it is preferable if the learning method uses a minimum of features. Similar to Occam's razor, this bias is remarkably difficult to implement correctly. Almuallim and Dietterich analyze the sample complexity of selecting a minimum of features, and they present an algorithm for iteratively searching larger feature sets. Crucial to their algorithm is the notion that for every *insufficient* feature set, there will be at least two data that have the same value for every attribute in the set but differ in their classification. This paper adopts their search idea and expands it by identifying pruning rules and techniques for efficiently implementing those pruning rules. The application of the idea is also slightly more general in this paper; instead of focusing on a fixed classification, the search identifies minimal domains for any range of interest.

Mannila and Räihä (1987) present an algorithm for inducing functional dependencies. Like Almuallim and Dietterich, their approach also uses disagreements between pairs of data. Basically, their algorithm computes all pairwise disagreements between tuples in a database. This takes $O(n^2)$ time. Then, it collects up all these disagreements as candidates domains for determinations. Like the pruning rules outlined above, their algorithm removes any supersets from these disagreements. This also takes $O(n^2)$ time for a total complexity of $O(n^4)$. A straightforward search is conducted from this point to find domains that include some attribute from each of the relevant disagreements. Asymptotically, both this algorithm and the one above are exponential in the number of attributes; however, the algorithm presented above can easily be bounded by limiting the maximum domain size. The algorithm presented above also has a time complexity that is only linear in the number of data. Kantola, Mannila, Räihä, and Siirtola (1992) identify a revision of their algorithm that makes use of an intermediate sort, lowering the time complexity to $O(n^2 \log n^2)$. They also point out specific ways dependencies can be used during the design of a database and demonstrate an implemented system.

Shen (1991) also explores learning regularities, but his work does not explicitly focus on minimality. The task is to discover regularities in a predicate knowledge base of the form $P(x, y) \wedge R(y, z) \rightarrow Q(x, z)$. His approach is to find all constants for which some predicate P is defined, then find some other predicate Q for which the relationship holds. No further search for a predicate R is necessary to discover a determination though it is for other regularities of interest. As is, the method is limited to finding determinations between pairs of predicates, but by searching for both a P and a Q, this method is searching for both a domain and a range. This paper instead assumes that a range is prespecified, and its methods then conduct a search for minimal domains, whatever their size. To find determinations for any range, the search can be repeated for each possible singleton range (as Shen does). Then the results of search can be combined by merging any determinations with a common domain.

4. Searching for a Minimal Key

Adding entries to a relational database requires entering values for a key. If a user wishes to add entries to a projection of the database, the projection must include a key. Given a number of keys, the editing system selects the key that most closely matches the user's requested projection. If the database contains keys not originally envisioned by the database's designers, then these keys may match some projections more closely, fitting naturally with the user's request.

Functional dependencies may be learned as determinations or specified by the database's designers. In either case, given a set of dependencies, it is easy to compute relational keys using the notion of transitive closure over dependency. This paper offers a combined algorithm for finding all keys and the closest matching key given a requested projection. Given a list of attributes to be included in a projection, and a set of determinations, the algorithm searches for a key that adds a minimum number of attributes to those the user wishes to edit.

The key-search algorithm performs a simple, breadth-first search for a key that is a superset of the requested attributes. The search is both complete and systematic. Unlike the search for determinations, the measure of a successful key search is a set of attributes whose transitive closure (given the determinations) includes all attributes. The worst case search requires $O(2^{m-r})$ space for the breadth-first queue and $O(2^{m-r} \cdot d^2)$ time, where m is the total number of attributes, r is the number of attributes in the requested projection, and d is the number of determinations. On average, the expected space and time is considerably less. For example, if there exists a single attribute key for the database, the space is $O(m)$, and the time is $O(m \cdot d^2)$. Even in this case, having multiple keys may mean that no additional attributes must be added to the user's request.

For a given set of attributes, construction of a list view consists of finding a key that adds a minimum of additional attributes, computing the width of each attribute's column in the view (by examining previous values), and filling in rows with previous data.

As an example, suppose the user wishes to add to a projection including $\{A, C\}$. By transitive closure with the two determinations $A \rightarrow B$ and $B \times C \rightarrow E$ in the example database of Table 1, B and E are also covered. Only D must be added to ensure that a key is represented. Figure 1 presents a more realistic example drawn from the zoological database. Here the user has requested a projection including aquatic and fins. The attributes animal name and venomous were automatically added so that the view includes a key. Using the databases in Table 2, random samples of 100 projection requests added an average of 2.2, 0.6, 0.4, and 1.0 attributes to cover a key, and took an average of 2.0, 0.1, 0.1, and 3.7 CPU seconds on a MC68040-based machine, respectively.

5. Using Dependencies to Complete Entries

When the user inserts an entry with an arbitrary key, the functional dependencies are used to complete the entry by computing values for attributes not present in the editing view. When there is a single-attribute key in the projection, there are two cases: either the value of the key is novel, or it has been observed in another datum. In the first case, the new entry can't violate the relational key. However, determinations cannot use novel values for prediction; in this case, they can't compute the value of any dependent attributes. This implies that the projection must expand to include these attributes so the user can enter their corresponding values. For example, if the attribute A was a single-attribute key, and $A \rightarrow B \times C$, then the attributes B and C would also have to be added to the

List View

animal name	feathers	eggs	aquatic	predator	toothed	backbone	venomous	fins
TORTOISE	*0*	*1*	*0*	*0*	*0*	*1*	*0*	*0*
TUATARA	*0*	*1*	*0*	*1*	*1*	*1*	*0*	*0*
TUNA	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*
VAMPIRE	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*
VOLE	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*
VULTURE	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*0*
WALLABY	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*
WASP	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*0*
WOLF	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
WORM	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*

Figure 1. Sample list view automatically generated from a request to edit feathers, eggs, aquatic, predator, toothed, backbone, and fins in the zoological data. Note that the attributes animal name and venomous were included to make up a unique relational key. Previously entered data appears in the uppermost rows in italics; new data appears in lower rows in normal typeface.

projection. In the second case, the new entry does violate the relational key because it duplicates a previous value. This is either an error or an indication that the user wishes to edit a previous entry. Which semantics are preferable depends upon the particular application.

In the more general case, there is a multiple-attribute key in the projection. There are three cases: one or more of the key attributes' values are novel, the key attributes' values are a novel combination of previous values, or the combination of key attribute values has been observed before. In the first case, the key is not violated, but the learned determinations cannot predict values for dependent attributes. There is no previous record of the key. For example, using the sample data in Table 1, and the projection $\{A, B, D\}$, the subset $\{A, D\}$ is a key. If the user entered a novel value of v for attribute A, then the determination $A \rightarrow B$ is unusable, and B must be added to the projection. In the second case, none of the key attribute values are novel, but their combination is. This too, does not violate the key. Following the same example, $\{i, i, ii\}$ for $\{A, C, D\}$ is a novel key, but the determinations can predict values for the remaining attributes B and E. In the third case, the combination of attribute

values has been observed before. Depending on the application, this is either an error or an implicit request to edit previous data.

6. Using Determinations to Topologically Sort Attributes

Finding a key that adds a minimum of attributes to the user's requested projection is sufficient for a list view. However, an additional opportunity to present information arises in the form view. A form view discards the tabular presentation of the data in favor of a view where only a single entry is visible. This allows the form view to arrange attributes in a two-dimensional layout. Thus, after finding a projection that includes a key, the attributes in the projection are sorted topologically so that dependent attributes appear after those which they depend upon.

A simple topological sort interprets dependencies as directed edges in a graph and attributes as nodes. First, all nodes are found that have a minimum of in-edges. These consist of the first subset of the topologically-sorted result. After removing all out-edges from these nodes, the process iterates until all nodes have been placed. For example, using the dependencies in Table 1, the attributes $\{A, B, C, D, E\}$ would be sorted as $(\{A, D\}, \{B, C\}, \{E\})$; neither A nor D appear in the range of any determination, so they appear in the first set. Table 3 lists the topologically sorted attributes for each of the four databases in Table 2.

Table 3. Attributes sorted based on the topology of learned determinations for each of the four databases listed in Table 2.

Database	Topologically Sorted Attributes
Breast Cancer	{{Sample code number, Clump Thickness, Bare Nuclei, Bland Chromatin}, {Uniformity of Cell Size}, {Marginal Adhesion}, {Mitoses}, {Uniformity of Cell Shape, Single Epithelial Cell Size}, {Normal Nucleoli}, {Class}}
Bridges	{{IDENTIF}, {LOCATION, ERECTED, LANES, T-OR-D, SPAN, TYPE}, {RIVER}, {LENGTH, CLEAR-G}, {PURPOSE}, {REL-L}, {MATERIAL}}
CPU Performance	{{Model Name}, {PRP, ERP}, {MMIN}, {CHMIN}, {MYCT, CHMAX}, {MMAX}, {vendor name}, {CACH}}

Table 3. Attributes sorted based on the topology of learned determinations for each of the four databases listed in Table 2.

Database	Topologically Sorted Attributes
Zoological	{{animal name, venomous}, {hair, airborne, aquatic, predator, legs, tail, domestic, catsize, type}, {fins}, {breathes}, {feathers, toothed}, {eggs}, {backbone}, {milk}}

An ordered list of sets of attributes maps naturally to a two-dimensional form-view layout. Attributes appearing in the first set are laid out in the first row; each in an editing box sized to fit the largest previous value. Attributes appearing in subsequent sets are laid out in subsequent rows. Figure 2 andFigure 3 depict form view layouts for requests to edit all the attributes in the databases of Table 2. These views were computed by topologically sorting the attributes. Note that attributes that appear in the same subset of the topological sort are laid out next to each other in the form view. If the user requests a projection that does not include all attributes, a search is first conducted to ensure that it contains a key, and the resulting set of attributes is sorted topologically. If several levels of dependent attribute are omitted (i.e., by omitting some attributes in the middle sets of the sort), then this many be visually indicated, perhaps by a horizontal line for each level omitted.

Figure 4 depicts two sample views generated in response to edit a subset of attributes. Rather than generating the view directly, these have been converted into a specification language for a commercial electronic forms package (Informed Designer), resulting in the electronic forms depicted. This represents only an initial step in the process of automatically designing electronic forms from a database; several other reasonable tasks include better use of the form's white space, automatically recognizing different types of fields, and adding default value or integrity constraints.

Like the list view, functional dependencies are used to compute values for dependent attributes not entered in the form view. If because of novel values a determination cannot predict a dependent value, the form view must expand to include the additional attributes. By definition of the layout, these new attributes will be added below those which they depend upon. If the user fills out the form view in a top-to-bottom manner, the form will appear to expand naturally as required.

7. Conclusions

This paper explores the possibility that a database may contain additional keys beyond those originally specified. It gives an algorithm for finding functional dependencies that is linear in the number of data. The paper then shows how these dependencies can be used to find additional keys. A second algorithm uses these keys to construct a sufficient list view given a user's request to edit a projection of the database. The view incorporates just enough additional attributes to include a key. A third algorithm uses the first two and a topological sort to construct a form view that helps the user visualize the data in a natural and unobtrusive manner. The resulting projections are dynamic and expand as required to include attributes that cannot be computed from those that are visible.

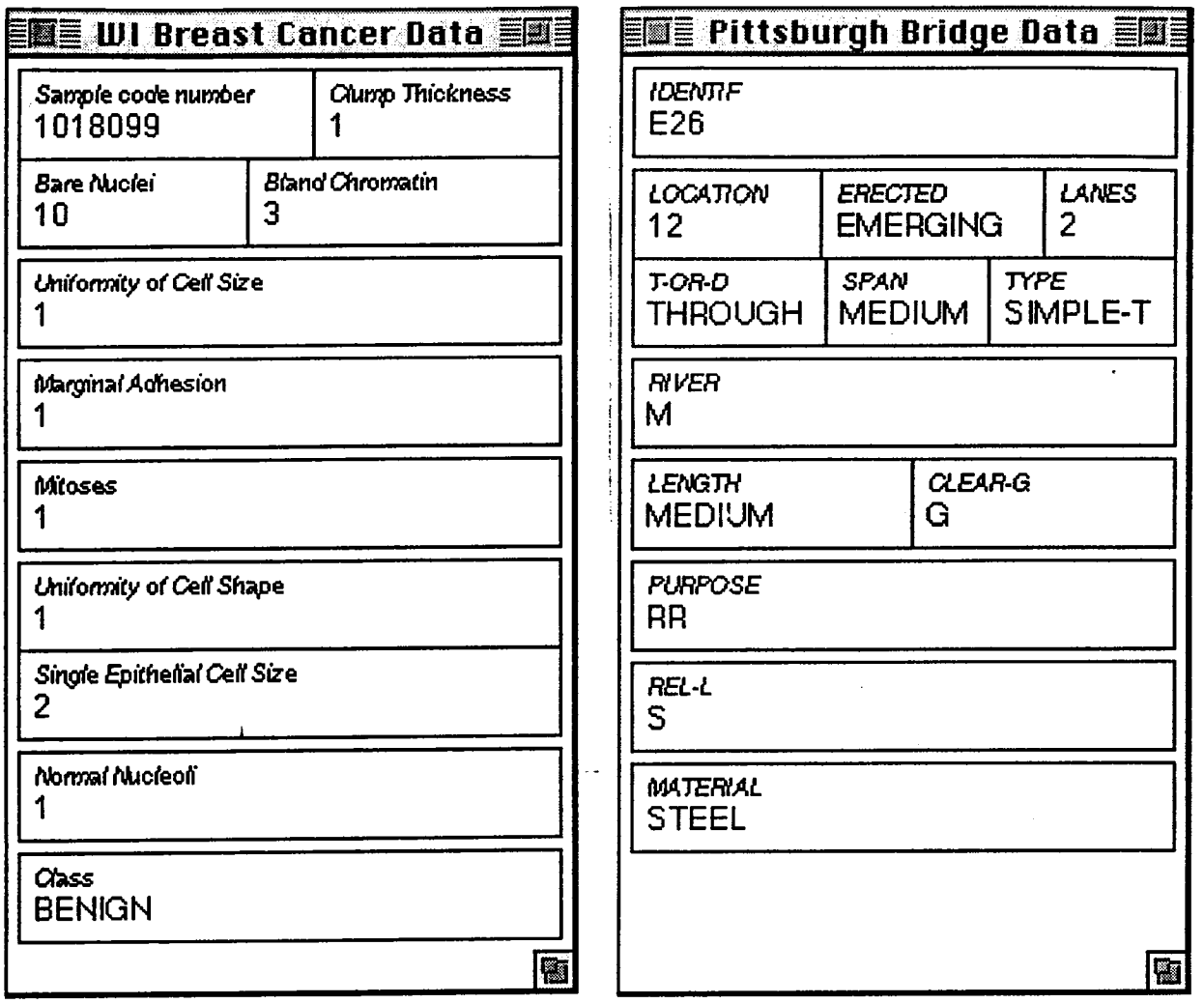

Figure 2. Form view for the breast cancer and bridges databases summarized in Table 2.

One limitation of this work is that the model for recognizing duplicate keys fits poorly with databases where determinations must be incrementally revised. If the determination learning algorithm is mislead by initial data, there will be too many hypothesized keys. This leaves the problem of whether a duplicate key indicates an error on the part of the data or with the determinations.

Acknowledgments

This research has been supported by Digital Equipment Corporation and grant IRI-9212190 from the National Science Foundation. Anthony Templar of AT&A suggested the idea of using learned information to generate a form view. This breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg (Mangasarian & Wolberg, 1990). The bridge database was obtained from Yoram Reich and Steven J. Fenves. The CPU performance database was obtained from Phillip Ein-Dor and Jacob Feldmesser. The zoological database was obtained from Richard Forsyth. Leonard Hermens and Jack Hagemeister made several useful comments on an earlier draft of this paper. Dongil Shin helped with early versions of the determination learning algorithm. Two anonymous reviewers pointed out some interesting related

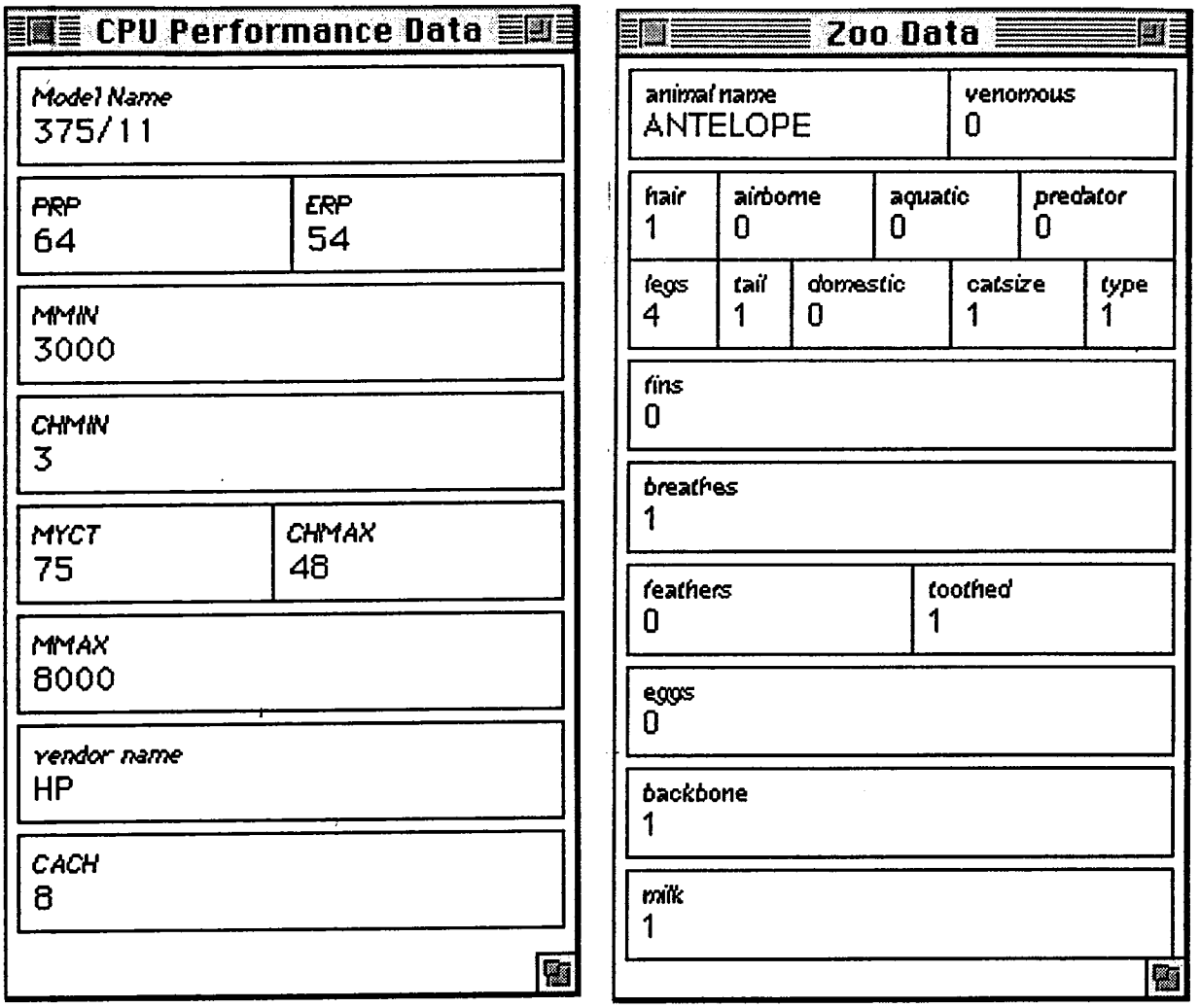

Figure 3. Form view for the CPU performance and zoological databases summarized in Table 2.

work.The faculty and students studying AI at Washington State University cultivate a stimulating intellectual environment. Jim Kirk, Mike Kibler, Karl Hakimian, and the facilities group provided a reliable computing environment. The MCL group at Apple provide an excellent Common Lisp development environment.

References

Almuallim, H., & Dietterich, T. G. (1991). Learning with many irrelevant features. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 547–552). Anaheim, CA: AAAI Press.

Davies, T. R., & Russell, S. J. (1987). A logical approach to reasoning by analogy. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 264–270). Milano, Italy: Morgan Kaufmann.

Kantola, M., Mannila, H., Räihä, K, & Siirtola, H. (1992). Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems, 7, 7.*

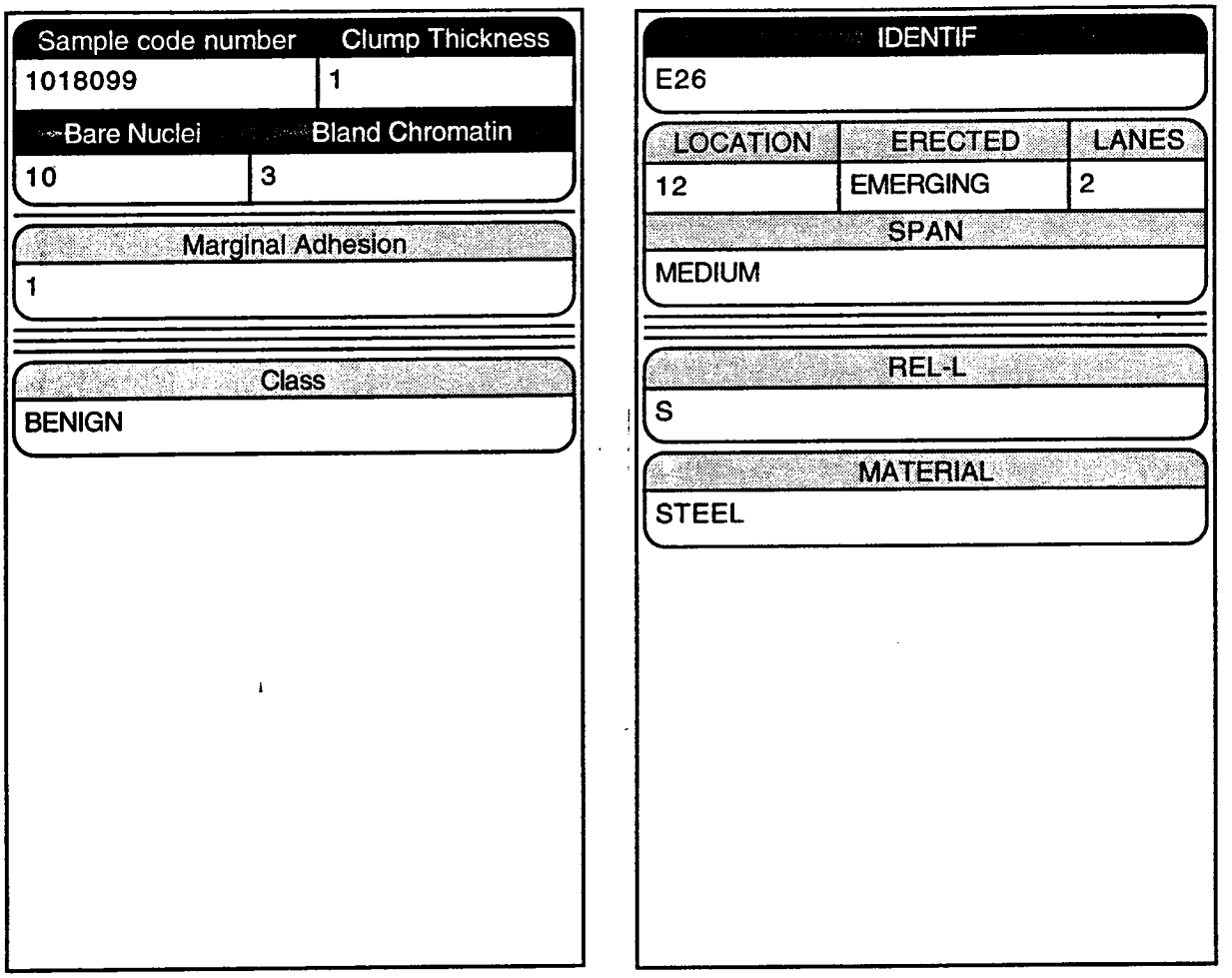

Figure 4. Sample, automatically generated electronic forms.

Mangasarian, O. L., & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News, 23*, 1 & 18.

Mannila, H., & Räihä, K. (1987). Dependency inference (extended abstract). *Proceedings of the Thirteenth Very Large Database Conference* (pp. 155–158). Brighton.

Russell, S. J. (1986). Preliminary steps toward the automation of induction. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 477–484). Philadelphia, PA: AAAI Press.

Schlimmer, J.C. (1993). Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. *Proceedings of the International Conference on Machine Learning*. Amherst, MA: Morgan Kaufmann.

Shen, W. (1991). Discovering regularities from large knowledge bases. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 539–543). Evanston, IL: Morgan Kaufmann.

Ziarko, W. (1992). The discovery, analysis, and representation of data dependencies in databases. In G. Piatetsky-Shapiro & W. Frawley (Eds.), *Knowledge Discovery in Databases*. Palo Alto, CA: AAAI Press.