Report Number: WUCS-88-25

1988-09-01

Relational Completeness of Show and Tell Visual Programming Language

Takayuki Dan Kimura

In this paper we present the database applications of the Show and Tell Language (STL) and demonstrate the relational completeness of the language. STL is a visual programming language designed for novice computer users who are not familiar with keyboarding. A program can be constructed by using only a pointing device, except for textual data entry. A program can be constructed by using only a pointing device, except for textual data entry. Various programming concepts such as subroutine, iteration, recursion, concurrency, exception, and so forth are represented by two-dimensional graphic patterns and icons. The language is used to test... **Read complete abstract on page 2.**

Relational Completeness of Show and Tell Visual Programming Language

Takayuki Dan Kimura

Complete Abstract:

In this paper we present the database applications of the Show and Tell Language (STL) and demonstrate the relational completeness of the language. STL is a visual programming language designed for novice computer users who are not familiar with keyboarding. A program can be constructed by using only a pointing device, except for textual data entry. A program can be constructed by using only a pointing device, except for textual data entry. Various programming concepts such as subroutine, iteration, recursion, concurrency, exception, and so forth are represented by two-dimensional graphic patterns and icons. The language is used to test the feasibility of keyboardless programming, Currently the language is implemented on the Apple Macintosh personal computer. In this paper we will present, first, the Show and Tell language primitives, then simple database applications through examples, and finally the representation of the five basic operations in relational algebra; difference, union, Cartesian product, projection, and selection; all using the Show and Tell visual constructs. This demonstrates that STL is a visual relational data query language which is complete in the sense of Codd.

RELATIONAL COMPLETENESS OF SHOW AND TELL VISUAL PROGRAMMING LANGUAGE

Takayuki Dan Kimura

WUCS-88-25

September 1988

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

This research was funded by Computer Services Corporation (CSK) of Japan.

1. Introduction

In this paper we present the database applications of the Show and Tell™[1] Language (STL) and demonstrate the relational completeness of the language. STL is a visual programming language designed for novice computer users who are not familiar with keyboarding. A program can be constructed by using only a pointing device, except for textual data entry. Various programming concepts such as subroutine, iteration, recursion, concurrency, exception, and so forth are represented by two-dimensional graphic patterns and icons. The language is used to test the feasibility of keyboardless programming[2]. Currently the language is implemented on the Apple® Macintosh™ personal computer.

With Show and Tell we try to integrate the computer capabilities for managing computation, communication, and database, into a single conceptual framework. From the end user's point of view, it is desirable to learn only one language, instead of three languages for three different areas of application. In order to achieve this goal we use the notion of *completion* as a mechanism for integrating computation with database applications.

To end user the STL system is a tool for defining and solving a kind of puzzle, called a Show and Tell (ST) puzzle. The system provides tools for solving, saving, modifying, and sharing with other users ST puzzles and solutions. The Show and Tell language is a specification language for ST puzzles.

A ST puzzle consists of boxes connected by arrows. The arrows define the neighborhood relationship among the contents of the boxes. Some boxes may be empty in the problem puzzle. The puzzle is solved when the empty boxes are filled with data objects satisfying the constraints imposed by the neighboring boxes. The puzzle is *consistent* if such a solution exists, and it is *inconsistent* otherwise. A ST puzzle defines a *completion problem* of filling the missing portions of an incomplete pattern, in the same way a jig-saw puzzle defines a completion problem. A puzzle is solved when the puzzle is completed. The STL system is a system with completion capability.

The system solves a puzzle either by computation or by database search of existing solutions. A puzzle is said to be solved by computation when the empty boxes are filled by transfer of data objects from the neighboring boxes. The computation model for STL is similar to the dataflow model[3]. A database search in STL consists of selecting a solution from the database that is consistent with the partial information given in the remaining part of the puzzle. From this point of view the STL system functions as an associative memory. It is possible to combine computation and database search in solving a single ST puzzle.

A solution for a ST puzzle is a tuple of data objects to be assigned to the empty boxes of the puzzle, and a set of solutions constitutes a relation. The puzzle defines what tuples could be contained in the relation, i.e., it defines the schema of the relation.

In this paper we will present, first, the Show and Tell language primitives, then simple database applications through examples, and finally the representation of the five basic operations

---

[1] Show and Tell is a trademark of Computer Services Corporation.
Macintosh is a trademark of Apple Computer, Inc.

[2] Kimura, T.D., Choi, J.W., and Mack, J.M., "A Visual Language for Keyboardless Programming," Technical Report WUCS-86-6, Department of Computer Science, Washington University, St. Louis, March 1986.

[3] Ackerman, W.B., "Data flow language," *Computer* **15**,2 (1982), pp. 15-25.

in relational algebra; difference, union, Cartesian product, projection, and selection; all using the Show and Tell visual constructs. This demonstrates that STL is a visual relational data query language which is complete in the sense of Codd[4].

2. Show and Tell Primitives

This section presents the basic language constructs through a sequence of examples based on the factorial function. The next section will present the constructs necessary for database applications.

A Show and Tell puzzle consists of three components; name, background, and boxgraph. Figure 1 shows a screen display of the system containing a puzzle that defines the factorial function recursively. A name is a rectangular bit image at the upper-left corner of the Edit window, and is used to identify the puzzle. The name is translated into an icon of standard size displayed on top of the trash can. The icon can be dragged into any box in the boxgraph representing the entire boxgraph itself. A background can be any bitmap picture or text, commenting about the puzzle like the equation in Figure 1. The boxgraph is the main component of a puzzle that the STL system interprets. It can spatially overlap with the name and/or background. The editing tools are provided in the horizontal menu on the left of the screen.

A boxgraph consists of one or more boxes connected by a set of arrows. An arrow connects one box to another, defining a flow of data, either scalar or vector, from the originating source box to the destination box. Figure 2 illustrates simple dataflow. Figure 2(a) is for computing factorial 5 and 2(b) for factorial 4. Two boxes may be connected by more than one arrow. An arrow may intersect with other arrows and boxes. While no two boxes may overlap with each other, nesting of boxes is allowed, i.e., one box may contain other boxes. The boxes and arrows may not form any cycle or a loop; a boxgraph is a partially ordered set of nested boxes. It is characterized formally as a directed acyclic multi-graph5. The acyclicness is required for making computation history independent and deadlock free.

There are many different types of box frames (Figure 3), but there is only one kind of arrow, a solid line with an arrow-head. The top four types of Figure 3 are used primarily for computational problems and will be illustrated in this section. The bottom three types are used for database applications and will be explained in the next section.

A **closed box** may contain nothing (empty), a data object (number, text, or picture), an icon (32X32 fixed-size picture) naming a boxgraph, or another boxgraph; representing a local variable, a constant, an operation, and a block structure, respectively. An open box may contain an icon or another boxgraph. A boxgraph can be recursively defined by having its name icon inside the boxgraph as in Figure 1.

A closed box and an **open box** differ in the ways the inconsistency is propagated to the outside of the box as illustrated by Figure 4. When a data value, 2, in Figure 4(a), is transferred to another box containing a different value, 3, the smallest boxgraph containing the destination box will become *primitively inconsistent*, i.e., it contains a conflicting dataflow. A boxgraph is defined to be *inconsistent* if and only if it is primitively inconsistent or it has an open box containing an inconsistent boxgraph. If an inconsistent

---

[4] Codd, E.F., "Relational completeness of data base sublanguages," in *Data Base Systems* (R. Rustin, ed.) Prentice Hall, 1972, pp. 65-98.

boxgraph is contained in a closed box, The smallest boxgraph containing the closed box may or may not be inconsistent. A closed box confines the inconsistency inside the box while an open box does not.

An inconsistent boxgraph is non-existing by definition. Any dataflow passing through a box containing an inconsistent boxgraph will be terminated with the box. For example, in Figure 4(a) the constant data object 1 cannot reach the destination box. In (b) the inconsistency is contained inside the smaller closed box and the larger closed box is consistent, therefore the constant 1 can reach the destination box. In (c) the inconsistency propagates out of the smaller open box, and the larger closed box also becomes inconsistent. Note that the Show and Tell system hatches all inconsistent boxgraphs.

A **base box** may contain nothing or a datum. It represent an input/output parameter in a procedure definition. In Figure 1 the puzzle has one input and one output parameter.

An **iteration box** may contain only a boxgraph. The iteration box represents an array of identical spatially spreading boxgraphs. There are two different forms of interaction between the components of the array and their environment; serial interaction and parallel interaction.

A *serial iteration*, represented by a pair of small triangles (*serial port*), provides a serial communication among the components. Figure 5 shows the syntax and semantics of serial iteration. The system dynamically creates a new copy of the boxgraph inside the iteration box, and transfers the data from the latest component to the new one. The process terminates when the newly created boxgraph is evaluated as inconsistent. The puzzle in Figure 6 is equivalent to the puzzle in Figure 2(b). Note that it does not terminate because the component boxgraph will never be inconsistent. Figure 7 presents a terminating version. Note that the data value 5 flows into every component of the iteration as a global constant, since there is no serial port designation on the arrow from the constant 5 into the iteration box.

A *parallel iteration*, represented by a striped rectangle (*parallel port*) on the iteration box, provides a parallel communication between two arrays of boxgraphs. Figure 8 shows the syntax and semantics of a simple parallel iteration. The source iteration box has a serial port as well as a parallel one and its execution terminates when an inconsistent boxgraph is created. The number of iterations at the destination box is the same as that of the source box minus the number of inconsistent components created by the destination box. The arrow connecting the parallel ports transfers a vector of data values in parallel. The puzzle in Figure 9 is equivalent to the puzzle in Figure 2(a). The definition of the operation [1,2,...] is given in Figure 10. It generates, as output, a sequence of integers from one to the input number.

When an iteration box has two parallel ports as in Figure 11(a), the iteration is interpreted as two nested parallel iterations, generating the cross product of the two incoming sequences. The semantics of (a) is given in (b) and (c). The utility of this construct will be illustrated in the next section.

3. Database Applications

A *file* in STL is a sequence of known solutions for a puzzle that has at least one base box. A solution is a set of data values which makes the puzzle consistent when they are assigned to the base boxes. A solution is a tuple of data values, and a file is a set of tuples. A puzzle and its file can be saved together in a *drawer* . The *directory* of a drawer is a set

of icon names representing the puzzles contained in the drawer. When the puzzle consists of base boxes only, and no arrow appears, it defines a record structure of a traditional programming language, where each base box represents a record field. A set of files in a drawer represent a relational database and the corresponding puzzles define schemata of the database.

A partially filled schema puzzle represents a simple data query. The STL system completes the puzzle, i.e., fills the empty base boxes, with a solution in the file that is consistent with the information in the partially filled puzzle. More complex queries can be constructed using the file box and the structure box constructs, as shown below.

An example of simple file definition is given in Figure 12. The puzzle defines a schema for recording golf score. The user fills each base box with proper information through a keyboard and saves the record into the puzzle by selecting the **Save** menu command. The small box next to the name area indicates how many solutions (records) are currently saved in the file associated with the puzzle. Any text outside of the base boxes is a part of the background and should be considered as a comment. The puzzle and its six solutions are stored in the drawer named **golf**.

For retrieving a particular record, the user can modify the puzzle to include a query condition and order the system to complete the puzzle. Figure 13(a) gives an example of such query specification. The query is "When did Kathy Smith score less than 100?" After the user selects the **Find** menu command, the system completes the boxgraph, as in Figure 13(b), with the first solution in the file that makes the boxgraph consistent.

For more complex data query specifications, a file box and a structured box of Figure 3 will be used. A **file box** containing a name icon represents the file of the named puzzle as a sequence of tuples. In semantics it is equivalent to a parallel iteration that produces or accepts a vector of records (Figure 14). A **structure box** represents the record structure of the schema puzzle. It is used to compose and decompose a tuple data structure. A set of values contained in a structure box can be treated as a single value.

Figure 15 illustrates how a structure box can be used to decompose a record structure. The puzzle prints the only records whose score is less than 100.

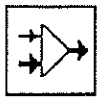
A parallel iteration can be used to construct a new file from an old file. For example, Figure 16 presents a program that constructs a file of good players based on the golf score file. The new file will contain the records of name and score, where the score value is less than 100. After the file is created, the user can browse through the good player file using the direct query method illustrated by Figure 13.

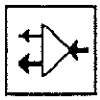
A parallel iteration box can also be used to combine more than one file to create a new file. The join operation, in the terminology of relational data model, can be implemented by the cross product mode of parallel iteration. Figure 17 illustrates the join operation in STL. We assume that the two schema puzzles of Figure (a) and (b) are already defined, and that the score file of Figure 12(a) and the player age file, a set of records consisting of name and age, were already created. The puzzle of Figure 17(c) constructs a new file, age score, which is a set of records consisting of age and score. For each record of score, and for each record of player age, the two name fields are compared to test the consistency. If there is no conflict, i.e., the two names are identical, then the entire boxgraph inside the iteration box is consistent and a new record of age score will be created. If two names are different, then the inside boxgraph will be evaluated as

inconsistent and no record will be generated out of the iteration box for this particular combination of input records.

Figure 18 demonstrates another example of database application in STL, involving image data objects. Figure 18(a) defines the schema of the Show and Tell personnel database, and (b) defines a query procedure of finding the profile of a record whose name field contains an occurrence of the input text. The procedure is used to construct a simple query, (c), to find a person's profile.

For manipulating a sequence of tuples, two primitive operations are made available in the STL system whose icon names are given below.


construct


decompose

The **construct** operation appends a tuple in front of a sequence, and the **decompose** operation produces the head and tail of a sequence.

4. Relational Completeness

A relational database query language is complete if the language can simulate relational algebra that is defined by the five basic operations on sets of tuples; difference, union, Cartesian product, projection, and selection[5] . In this section we will show that these five operations can be represented in STL. This demonstrates the adequacy of STL as a relational query language. We assume that the reader is familiar with the definitions of the basic operations.

Membership: Before we construct the union operation, we need an operation for testing the membership of an object in an array of objects, so that we can eliminate duplicate elements in the set union. Figure 19 defines the membership operation with the single input a and the array input A. If a is contained in A, then some copy of the boxgraph inside the parallel iteration is consistent and data object 1 flows into 0, causing the entire boxgraph inconsistent. If a is not in A, then all the copies of the iteration boxgraph are inconsistent and no communication between the constant 1 and the constant 0 occurs, leaving the boxgraph consistent.

Difference: Figure 20 defines the set difference operation, C = A - B.

Union: Figure 21 defines the set union operation with two array inputs, A and B, and one array output C, where C = $A \cup B$. Each component of B is tested for the membership in A, and if it is not contained in A, then it is appended in front of the resulting array.

Cartesian Product: Figure 22 defines a Cartesian product of the set of 3-tuples, A, and the set of 2-tuples, B, producing the set of 5-tuples, C, i.e., $C = A \times B$. Note that a parallel iteration box with more than one parallel ports generates the cross product of the input arrays.

---

[5] Ullman, J.D., Principles of Database Systems, Computer Science Press, 1980.

Projection: Figure 23 defines a projection operation from 4-ary relation, A, to binary relation, B, i.e., $B = \pi_{1,4}(A)$.

Selection: Figure 24 defines a selection operation on a relation A, selecting every tuple of A that satisfies the predicate F, and producing a new relation B, i.e., $B = \sigma_F(A)$, where F is the selection criteria. For example, using the golf score file of Figure 12, a selection of good score can be displayed by the procedure given in Figure 25(a). The selection criteria is specified in Figure 25(b).

5. Conclusion

We have shown that keyboardless programming and keyboardless data query are possible in Show and Tell. Also we demonstrated that, as a query language, Show and Tell is theoretically as much powerful as any other relational query language. It remains to be seen, however, whether the concepts introduced in Show and Tell are viable or not, in large scale practical applications.

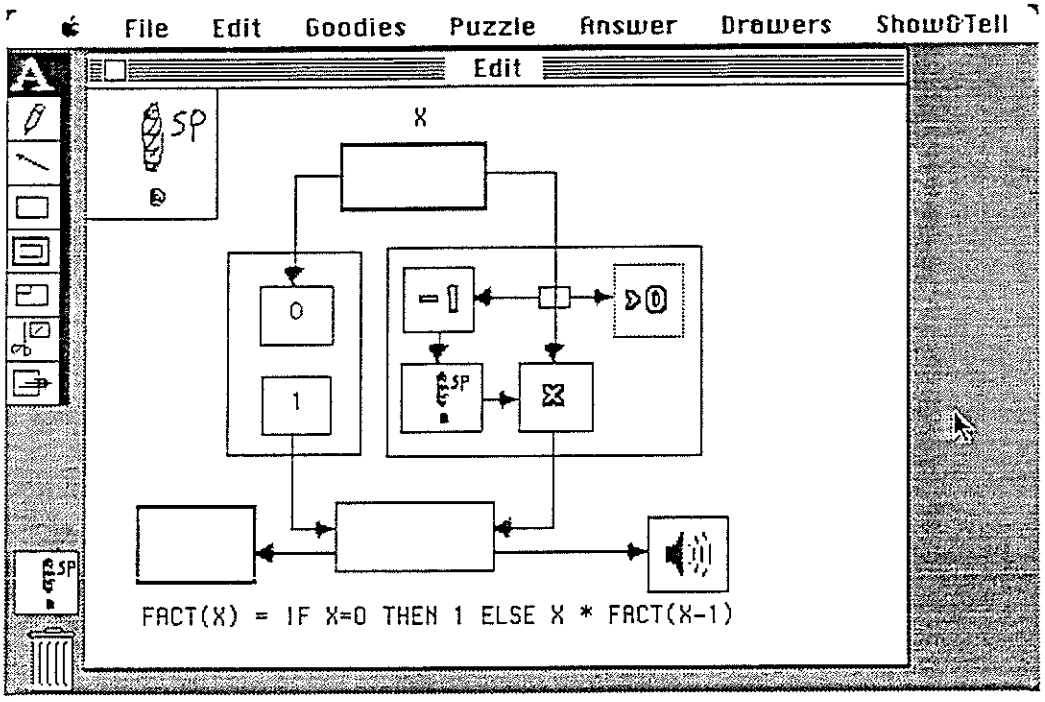
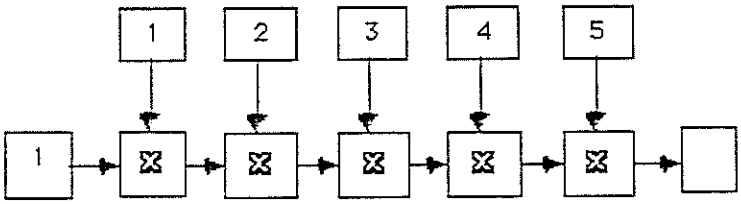
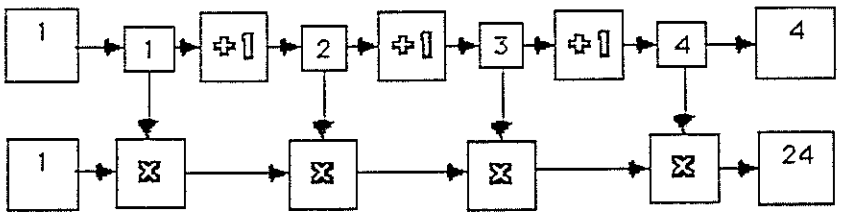


Figure 1: Show and Tell Screen Display



(a) Factorial 5



(b) Factorial 4

Figure 2: Dataflow

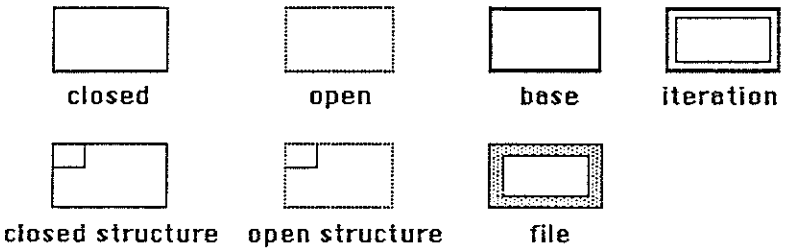


Figure 3: Types of Box Frames

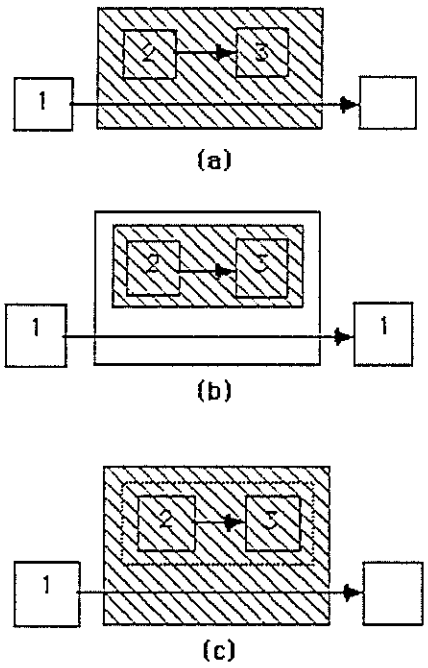
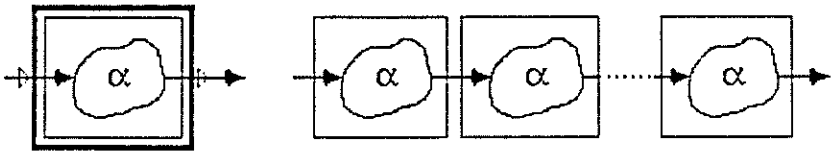


Figure 4: Inconsistency in Closed Box and Open Box



(a) Syntax (b) Semantics

Figure 5: Serial Iteration

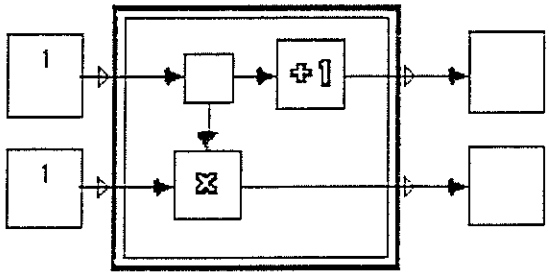


Figure 6: Nonterminating Serial Iteration

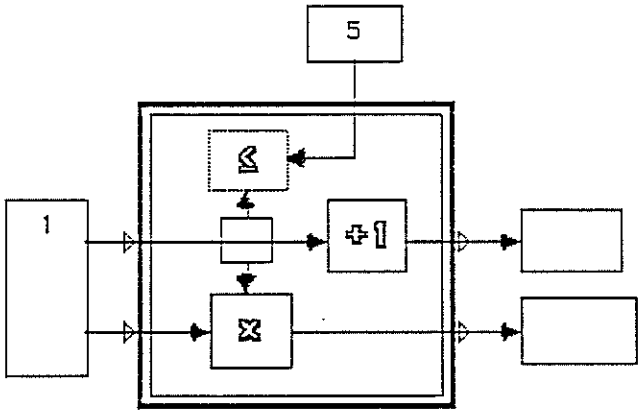
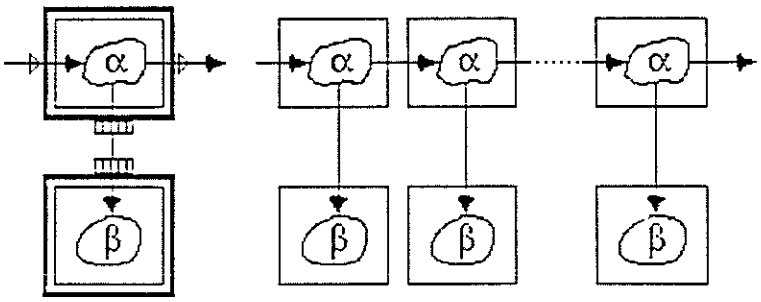


Figure 7: Terminating Serial Iteration (Factorial 5)



(a) Syntax (b) Semantics

Figure 8: Parallel Iteration

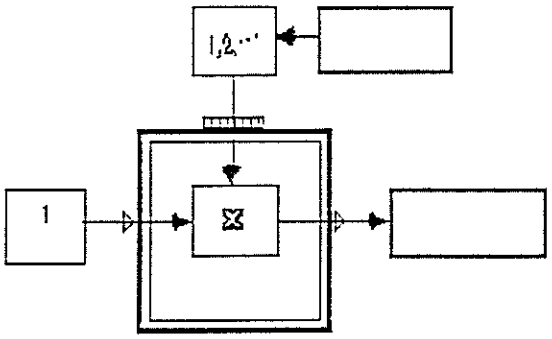


Figure 9: Factorial Function with Parallel Iteration

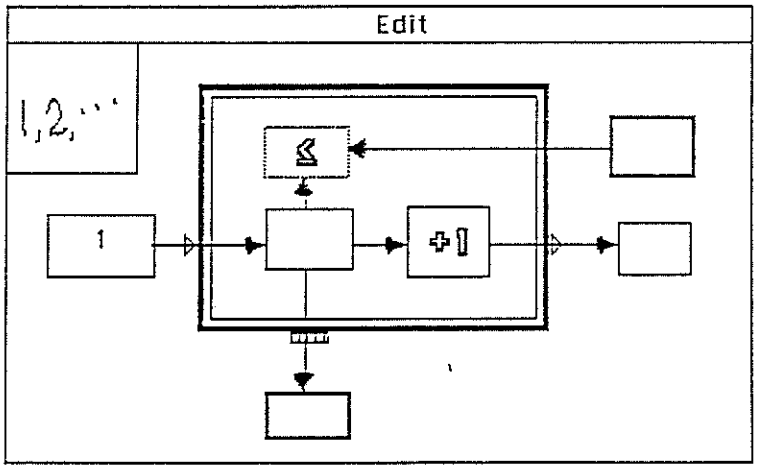
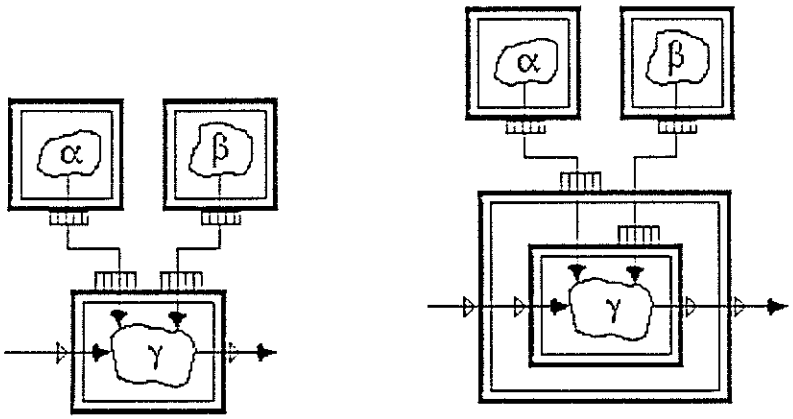
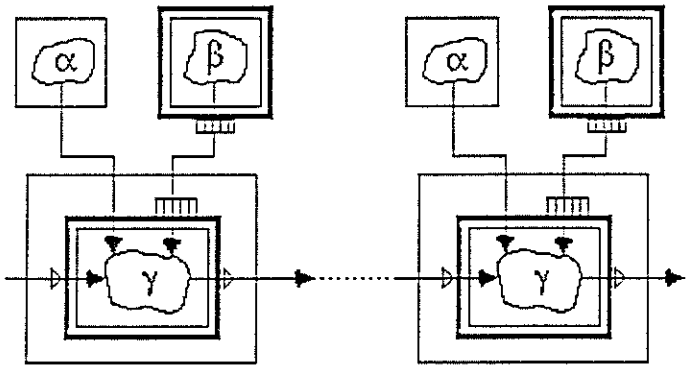


Figure 10: Function for Integer Sequence Generation



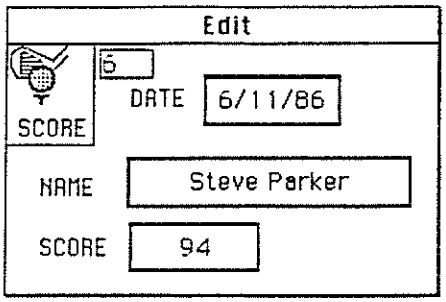
(a) Syntax

(b) Semantics



(c) Expansion of Semantics

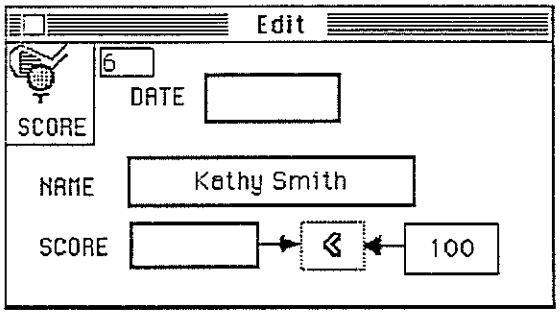
Figure 11: Cross Product Mode of Parallel Iteration



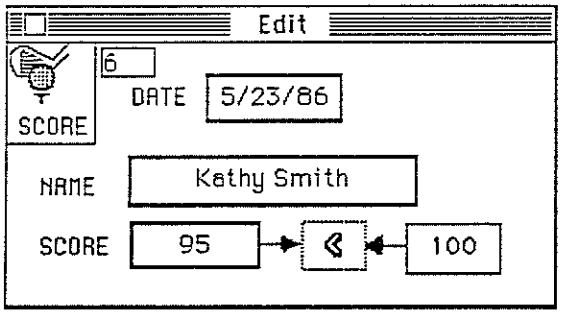
(a) Schema

(b) Directory of **golf** Drawer

Figure 12: File Definition for Golf Score

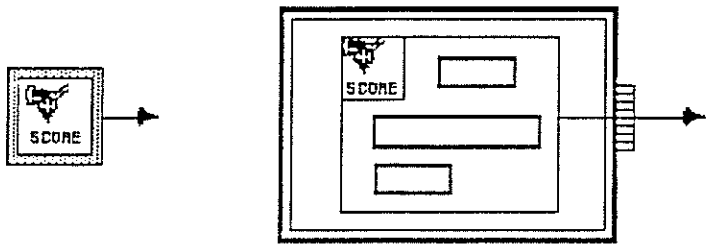


(a) Specification



(b) Solution

Figure 13: Direct Database Query



(a) Syntax (b) Semantics

Figure 14: File Box

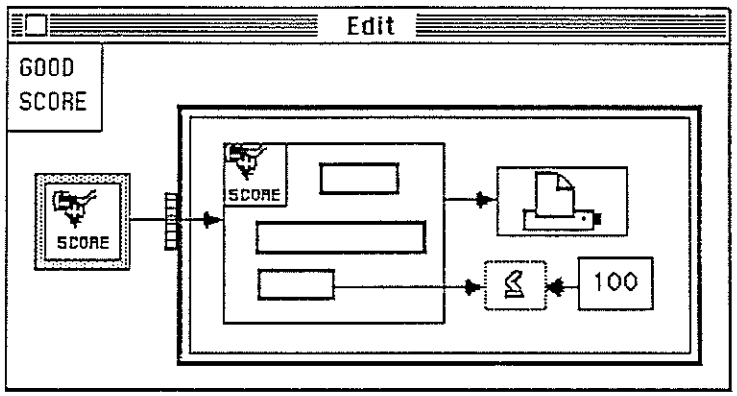


Figure 15: Structure Box for Record Decomposition

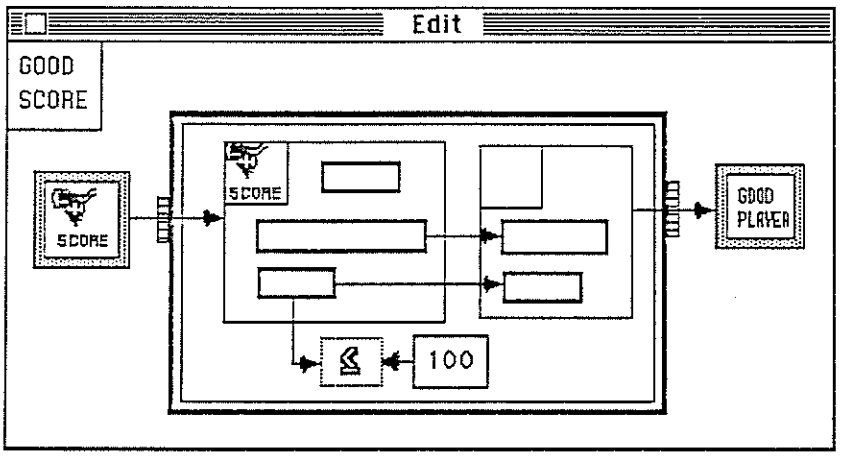
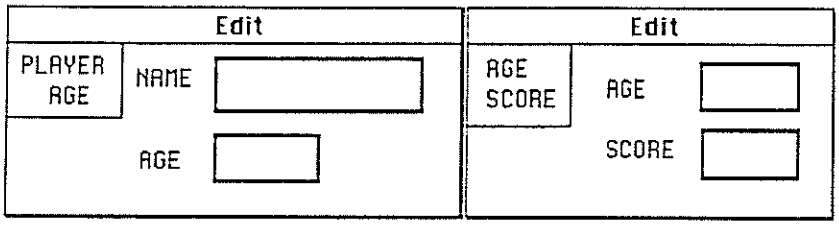
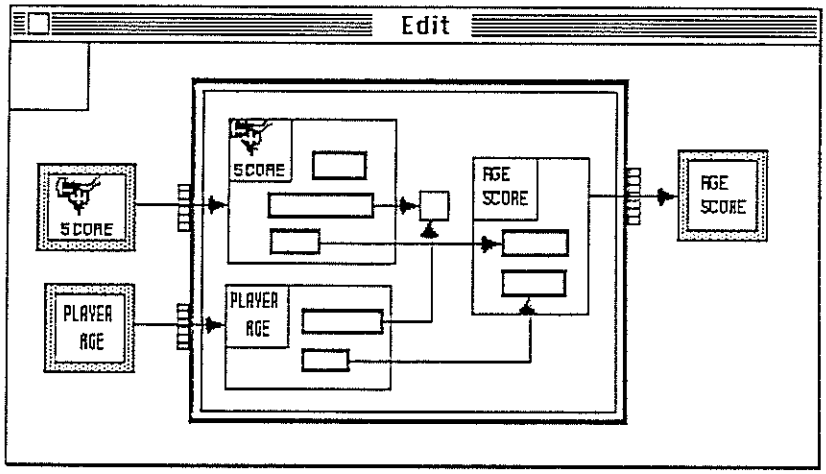


Figure 16: Construction of New File

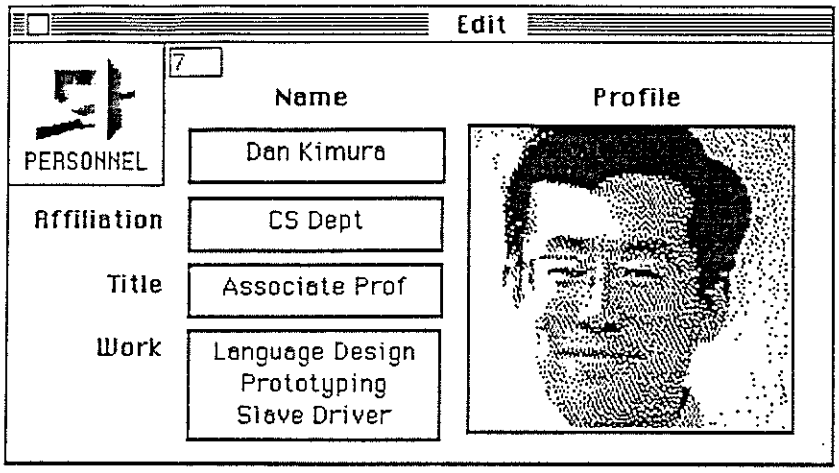


(a) Schema of Player Age File (b) Schema of Age Score File

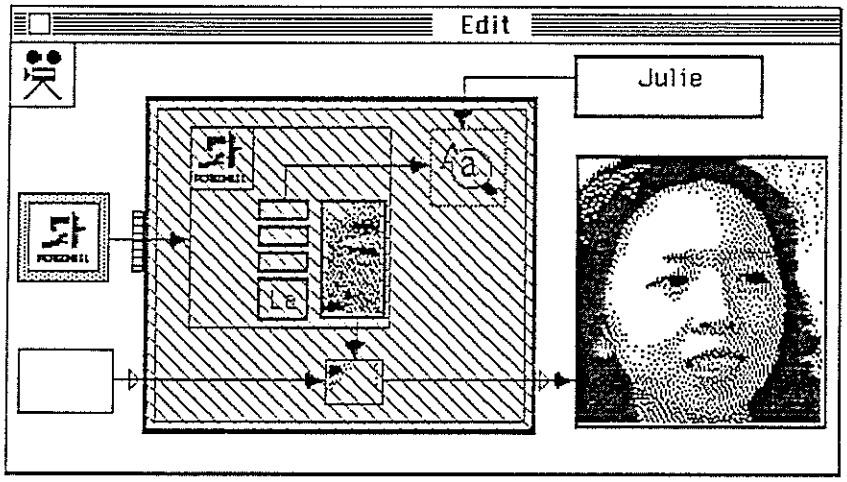


(c) Join Operation

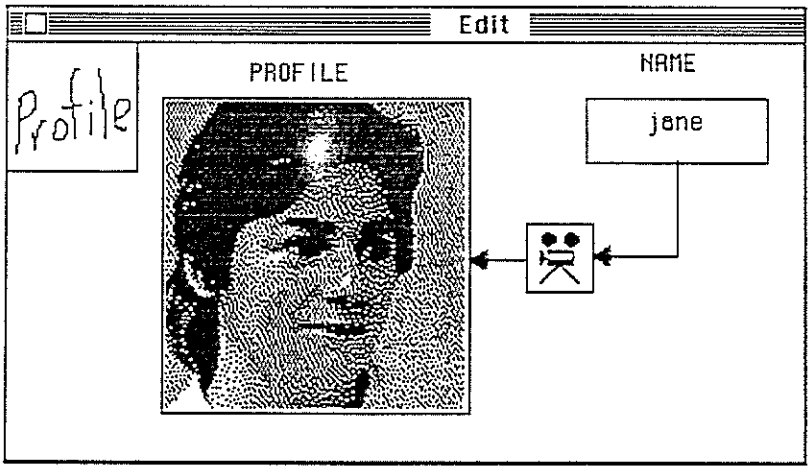
Figure 17: Combining Two Files



(a) Schema for Show and Tell Personnel File



(b) Picture Data Query Program



(c) Profile function

Figure 18: Picure Database

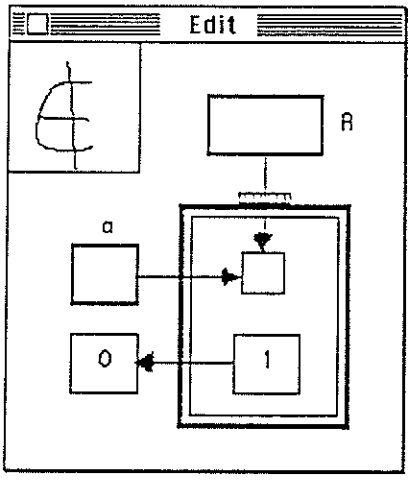


Figure 19: Membership Predicate

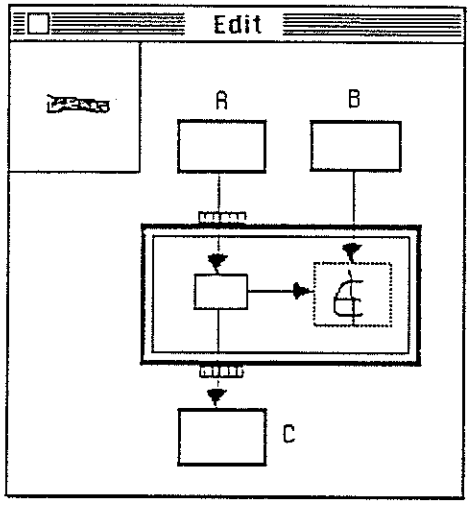


Figure 20: Difference Operation

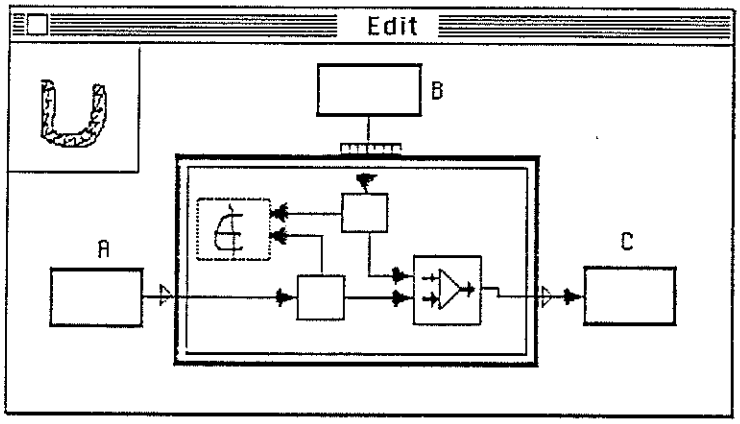


Figure 21: Union Operation

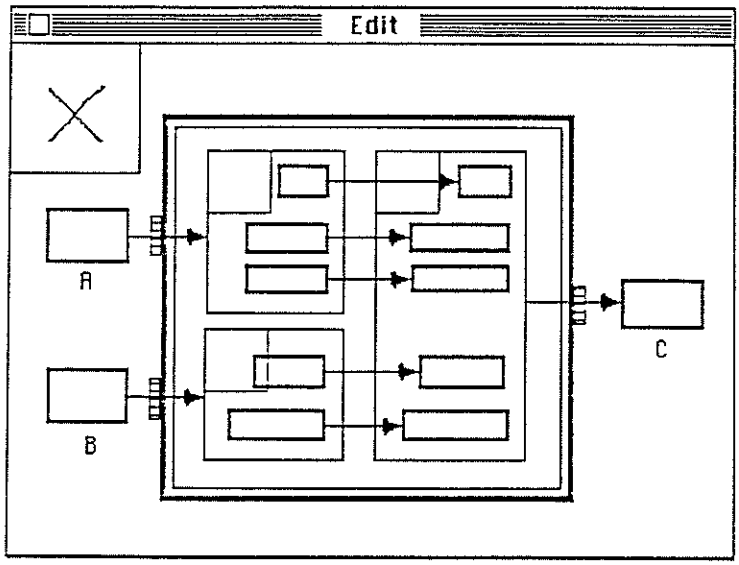


Figure 22: Cartesian Product ×

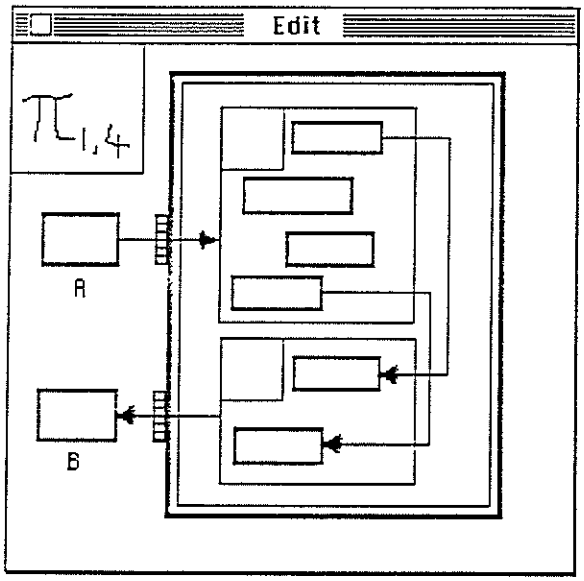


Figure 23: Projection Operation $\pi_{1,4}$

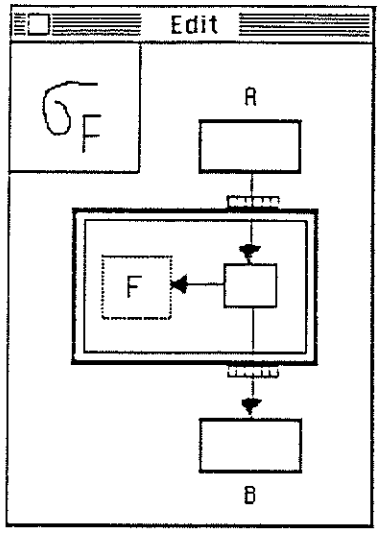
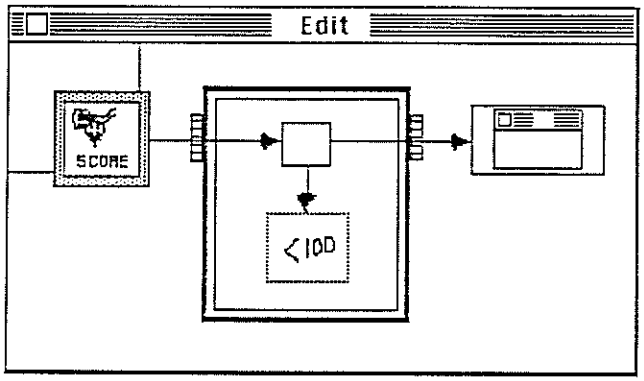
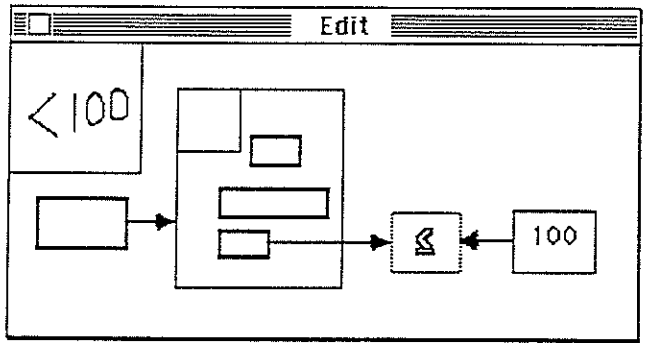


Figure 24: Selection Operation σ_F



(a) Displaying Selection of Good Score



(b) Selection Criteria

Figure 25: Example of Selection Operation