01 Jan 1996

A User Interface for the Visualization and Manipulation of Arrays

Jennifer Leopold
Missouri University of Science and Technology, leopoldj@mst.edu

A. Ambler

Recommended Citation

J. Leopold and A. Ambler, "A User Interface for the Visualization and Manipulation of Arrays," *Proceedings of the IEEE Symposium on Visual Languages, 1996*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1996.
The definitive version is available at https://doi.org/10.1109/VL.1996.545267

A User Interface for the Visualization and Manipulation of Arrays

Jennifer L. Leopold and Allen L. Ambler
Department of Electrical Engineering and Computer Science
The University of Kansas
Lawrence, KS 66045

Abstract

The success of spreadsheets has shown that a visual representation of a two-dimensional array greatly facilitates solving certain problems. However, spreadsheets are not a general-purpose programming environment and are not suited to many problems that might naturally be solved using multi-dimensional arrays. Furthermore, spreadsheets employ a textual notation for cell references in formulas. This notation, which adds to the programmer's burden by distinguishing relative and absolute addressing, can be difficult to understand and is error-prone even for the most experienced users.

In this paper we present a user-interface for multi-dimensional arrays within Formulate, a form-based visual programming language. This implementation avoids textual array notation and supports the application of formulas to logical regions of an array, rather than just to individual elements.

1. Introduction

An alternative to textual languages and spreadsheets is Formulate, a form-based visual programming language that supports arrays and the application of formulas to logical parts of an array, called regions. In this paper we discuss in more detail the Formulate user-interface for multi-dimensional matrices and its effectiveness in the creation and manipulation of regions, as well as referencing them within expressions.

2. Array Representation in Formulate

An array cell in Formulate can be of any number of dimensions of arbitrary size. Array elements can be of any simple data type or can be the composition of primitive graphical objects (i.e., ovals, lines, etc.). Array elements do not have to be homogeneous in type. To facilitate the manipulation of the two-dimensional display, the user has the option of displaying scroll bars to scroll horizontally and vertically through the elements, and dimension selectors in the lower left and upper right corners to choose which dimensions are displayed horizontally as "rows" and vertically as "columns." Figure 1 shows an array of two dimensions.

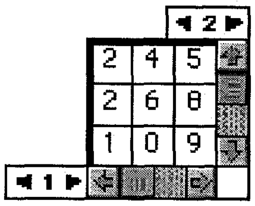


Figure 1. Array of two dimensions

3. Region Representation and Manipulation

An array can be divided into logical parts, called regions. Regions can be used both to describe the construction of other arrays and to describe the region's use in subsequent computations [1]. A region is visually represented in Formulate by dark lines outlining the rectangular area that encloses the elements of the region. When an array is created, it initially has only a single region which contains all the elements of the array. New regions are formed by splitting other regions; this guarantees that an array is always completely covered by its regions and that regions do not overlap. To create a new region, the user clicks on a region boundary line and drags it.

An existing region can be resized in a similar manner. The user holds down the shift key and clicks on a region boundary line, then drags the line. Boundary lines along the first or last rows or columns of the matrix cannot be dragged for resizing as that would leave elements not belonging to any region. Therefore, the user will always be resizing at least two regions at a time. To resize or split regions of more than two dimensions, the user must use the dimension selectors to display two of the dimensions and drag the region boundary lines within those dimensions. Another dimension can then be selected and the procedure repeated as necessary.

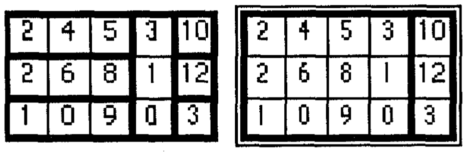
Common functions like + and *, as well as user-defined functions, can be applied to array regions. A reference to a region can be inserted into the expression

0-8186-7469-5/96 $05.00 © 1996 IEEE

for another region by clicking on the region to be referenced and selecting the desired attribute (such as the size of one of the dimensions) from a pop-up menu. An important distinction from spreadsheets is that an expression for a region is associated with the region as a whole and not individual cells. Thus the expression is never copied altering internal references and the user never has to understand the distinction between relative and absolute addressing.

Keeping expressions associated with regions, avoids the relative-absolute referencing problem of spreadsheets. However, there is an analogous problem created. If we wish to increment a region by the value of a single cell, then we are applying "+" to a region and a single cell. On the surface this would appear to be a type mismatch, yet the user's intention is pretty clear. To handle these cases, and many more complex ones, expressions involving regions are checked for consistency and applicability by an intelligent assistance system within Formulate [2, 3, 4].

It is possible that an array may need to be subdivided differently for subsequent computations. The various subdivisions of an array are called partitions. An array can have any number of partitions, one of which, the base, actually defines the array. The others are called views and are only used to access it [1]. At any time after an array has been created, the user can create a view by selecting the array and then clicking on a "create view" icon. An array view cell will be created with the same display characteristics (size, font, color, etc.) as the array cell. Regions can be created and resized by dragging region boundary lines within the view just as for the original base partition. Array view cells are distinguished by a double-line, rather than single-line, border. An array cell and its view cells are associated by having the same (system-generated) color for the cell border. Figure 2 shows an array cell and one of its view cells.



2	4	5	3	10
2	6	8	1	12
1	0	9	0	3

2	4	5	3	10
2	6	8	1	12
1	0	9	0	3

Figure 2. An array and a view

4. User-Defined Functions for Arrays

Formulate allows the user to define functions (themselves forms). Consider the problem of multiplying a single row by a single column. A function for this can easily be created in Formulate as in Figure 3 where the expression for the output cell (with the criss-cross pattern on its arrow) is specified as the sum of the product of the two input vectors. Particularly important is that (1) this solution does not require any knowledge of referencing representations for individual cells, (2) there is no relative-absolute referencing concept to confuse the user, (3) the number and sequence of steps required to create the solution is independent of the size of the arrays involved, and (4) the resulting function is applicable to arbitrary sized arrays.

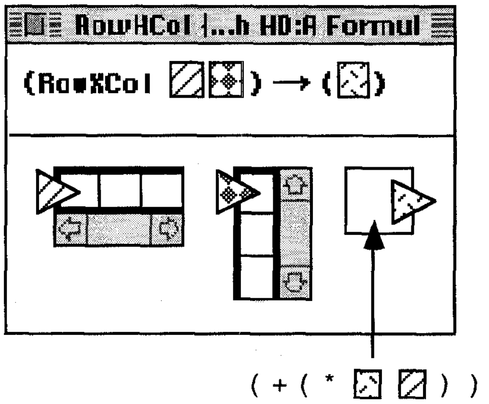


Figure 3. RowXCol function in Formulate

5. Discussion and Conclusions

An advantage to Formulate is its ability to specify arrays of arbitrary size that can be used in user-defined functions. In an imperative language like C, the array size must be known at compile-time or else the user must develop a pointer-based implementation that dynamically allocates memory for the array data. This may be rather difficult for a novice programmer to do. Spreadsheets, as well, must specify the maximum dimensions for cell blocks used in formulas. In Formulate, the user can drag lines to define regions or can specify an expression for each dimension of a region. Such expressions can reference the dimension expressions for other regions and will be computed dynamically.

6. References

[1] Gerhard Viehstaedt and Allen Ambler, "Visual Representation and Manipulation of Matrices", in Journal of Visual Languages and Computing, Volume 3, 1992, pp. 273-298.

[2] Guijun Wang and Allen Ambler, "Applicability Checking in Visual Programming Languages", in Proceedings of IEEE 10th Symposium on Visual Languages, 1994, pp. 31-38.

[3] Guijun Wang and Allen Ambler, "Invocation Polymorphism", in Proceedings of IEEE 11th Symposium on Visual Languages, 1995, pp. 83-90.

[4] Guijun Wang and Allen Ambler, "Solving Display-Based Problems", in Proceedings of IEEE 12th Symposium on Visual Languages, 1996.