Spreadsheets in Boxer: Controlling Complexity and Supporting Generalization in a Reconstructable Computational Medium

Jeremy Roschelle
Institute for Research on Learning*

*The research reported herein was performed while the author was a visiting scholar at the Sunrise Research Laboratory at the Royal Melbourne Institute of Technology.

CONTACT INFORMATION

address: Institute for Research on Learning,
2550 Hanover Street, Palo Alto CA 94304 USA
phone: 1 415 496-7974
fax: 1 415 496-7957
e-mail:- jeremy_roschelle@irl.com

Abstract

Boxer is a reconstructable computational medium which builds on and extends the accomplishments of Logo. Boxer's design principles of spatial metaphor and naive realism enable computational ideas to be implemented compactly, simply, and clearly. This article shows how these design principles enable fundamental computer science issues to be addressed in a real-world context. Controlling complexity and supporting generalization are two of the most powerful and central ideas in computer science. The article shows how building a spreadsheet, a complex, real-world application, becomes a manageable programming task in Boxer and how the resulting Boxer spreadsheet can be extended to other educationally relevant domains such as vectors, graphs and music. The discussion focusses attention on how Boxer's design principles provide educational advantages relative to Logo, Hypercard, and C and Pascal. Specifically, it is suggested that Boxer (1) supports a close connection between a conceptual model of a spreadsheet and a working implementation, (2) provides a concrete, visible, working model that can be inspected, explored, and modified and (3) eliminates distracting display and editting issues and concentrates cognitive load on the fundamental computer science issues.

Spreadsheets in Boxer: Controlling Complexity and Supporting Generalization in a Reconstructable Computational Medium

Jeremy Roschelle
Institute for Research on Learning
Palo Alto CA USA

Boxer is a computer language and system designed for educational use, especially in mathematics, science and technology studies. Boxer has been driven from the onset by a vision and set of principles based on educational research and educational goals (diSessa and Abelson, 1986). This article provides a compelling example of how Boxer's design principles — naive realism and spatial metaphor — make complexity manageable and generalization achievable.

Managing complexity and generalization are two of the most powerful and central ideas in computer science (Abelson & Sussman, 1985). The former refers to the mechanism by which computational processes and states can be described and specified. A medium should allow complicated behaviors to be composed from a basic set of processes and structures compactly, simply, and naturally. Generalization refers to the process of extending existing applications to a wide set of specific demands and custom uses. A well-designed language should make control of complexity and generalization easy.

This article shows how Boxer's design provides strong capabilities for control of complexity and generalization, and consequently it shows that Boxer can be valuable for teaching students about these important computer science ideas. The approach taken is to examine how Boxer enables a complex application, a spreadsheet, to be implemented elegantly and simply. In addition, three examples illustrate how the design principles employed in Boxer allow a spreadsheet to be generalized to an uncommon extent.

Most examples available for students to explore fundamental computer science concepts are toy problems. Writing a spreadsheet is not a toy problem, and yet in Boxer, it becomes a reasonable exercise. In Boxer, in contrast to other available tools and languages, implementing a spreadsheet is a real world challenge that can introduce students to fundamental principles of computer science.

Goals and Principles of Boxer

Although closely related to Logo, the goal of Boxer is more ambitious in scope. Logo (Papert, 1980), one of the most popular educational programming languages, is closely associated with teaching mathematics and computer science; in particular, Logo is associated with teaching geometry and recursion. Boxer retains these goals, but also seeks to provide general purpose text processing, calculation, graphics, database, and programming capabilities in a easy-to-learn framework (diSessa & Abelson, 1986). Since Boxer completely subsumes Logo capabilities, the main advances for a Logo user are in understandability and ease-of-use. These are described elsewhere (e.g., diSessa, Abelson, & Ploger, 1991; diSessa, 1986; diSessa & Abelson, 1986).

The more general goal of the Boxer design is best summarized by the concept of a "reconstructable computational medium" (diSessa & Abelson, 1986). The term "medium" indicates that Boxer is intended to support activities of constructing, expressing and communicating.

One reading of the term "computational medium" is a medium that exists on a computer. However, Boxer aims for a reading in which computation plays a more pervasive role (Abelson & diSessa, 1986); a computational medium is a material for constructing, expressing, and communicating *through the metaphor and actual processes of computation*. The contrast of a word processor and a spreadsheet may clarify the distinction. A word processor is a medium for writing on a computer, but computation on a word processor is largely hidden from view, and in most cases, completely inaccessible. In contrast, a spreadsheet is a medium in which computational ideas — active mathematical relationships — can be expressed and used to communicate and calculate. Boxer seeks be a computational medium in the spreadsheet sense; it is a system in which computational operations are central to the constructive activity.

The additional term "reconstructable" indicates that Boxer provides a computational medium that is susceptible to modification and re-use of ideas. Many programming environments make a strong distinction between the programmer and the user. The programmer may modify and re-use computational expressions, but the user may only do what the programmer allows. Boxer seeks to blur this distinction, making programming, and re-use of program ideas a central activity among Boxer users (Picciotto & Ploger, 1991). Reconstructability is demonstrated in this article in the ease of modifying and extending the basic spreadsheet.

The spreadsheet examples that follow will highlight Boxer's capabilities as a reconstructable computational medium. They will show, first, how Boxer provides an exceptional medium for constructing expressing ideas about the *computation of mathematical relationships* by building *computational structures and processes* in accordance with fundamental *computer science principles*. Furthermore, the examples will show how reconstructing a simple Boxer spreadsheet can generate a wide variety of useful tools.

Boxer derives these capabilities from a well-articulated and studied set of design principles. Among the most important principles in the design of Boxer are spatial metaphor and naive realism (diSessa & Abelson, 1986; diSessa, et al., 1991). Boxer uses the principle of spatial metaphor to tap the powerful intuitive understanding that people have of space. By tapping spatial intuitions, Boxer aims to make computational ideas like recursion and hierarchy more accessible. Boxer uses the principle of naive realism to enable the user to treat the visible screen as a concrete model of computation, obviating the need to imagine structures and processes that operate "behind the scenes." Thus Boxer's aim is to reduce cognitive load by directly supporting a user's mental model of computational data and process.

Through the examples, this article will demonstrate how the principles of spatial metaphor and naive realism provide leverage. In particular, it will be argued that the principle of spatial metaphor gives an unusually elegant expression of the computational structure of a spreadsheet. The principle of naive realism significantly reduces the gap between conceptual understanding and working implementation. In addition, it will be argued that the use of "boxes" as a basic building block provides powerful opportunities for supporting generalization.

Specification of a Spreadsheet

Spreadsheets can be a very powerful educational tools. Spreadsheets are finding application beyond business — in teaching mathematics (e.g. Arganbright, 1984; Healy & Sutherland, 1990; Visch, 1990), science (e.g. Goodfellow, 1990; Graham, 1987) and other topics at the primary level (e.g. Quinn, 1986) through the university level (e.g. Turner, 1988).

For the purposes of this article, a spreadsheet is taken to have these properties:

- a matrix of cells, each containing a value and a formula.
- references to other values in the matrix by relative reference.
- recalculation of each cell's value according to its formula.

Commercial spreadsheets (e.g. Microsoft Excel, Lotus 1-2-3) also have charting capabilities. Charting is a relatively independent module from the cell-based calculation facilities listed above, and thus, for the purposes of this article, is not considered part of the basic specification of a spreadsheet. It is nonetheless straightforward to produce most common chart types from spreadsheet values in Boxer, as will be described later.

Spreadsheets usually contain "scripting" or procedural programming capabilities as well. These are inherited directly from Boxer, and thus no specific scripting capability need be developed. Commercial spreadsheets further incorporate a large set of user-interface niceties, such as the ability to generate scripts by example, annotate cells, format results, copy and paste with a mouse, etc. Consideration of these ancillary features is also postponed until the discussion.

Major obstacles face a programmer who writes a spreadsheet in Logo, C, Pascal, or Hypercard, as will be discussed later. Similar obstacles face educators who try to explain how a spreadsheet works with only these conventional languages at their disposal. But in Boxer, both these constraints can be dissolved. A basic spreadsheet can be implemented in a matter of hours. Moreover, Boxer can express how a spreadsheet works, in terms of data structure and flow of control, in an unusually clear manner. Thus, with Boxer, teachers and their students can explore fundamental ideas in computer science — control of complexity and support for generalization — in the context of a familiar, real world application, a spreadsheet.

A Basic Boxer Spreadsheet

This section demonstrates how a spreadsheet can be written in Boxer in a relatively short time (from half a day for an expert, to perhaps a week for a student). In so doing, it illustrates how the data and procedural primitives provided by Boxer allow control of computational complexity.

The basic structure provided by Boxer for making things is the box.[1] Boxes are of two basic types, data and doit (pronounced "do it"). A data box contains data; doit boxes contain procedures. The boxes are distinguished by labels and the corners of their borders (data boxes have rounded corners). Both kinds of boxes can contain text and other boxes. Spreadsheets can be built in Boxer using these two box types.

Data structure

Conceptually a spreadsheet is a two dimensional matrix of cells. A spreadsheet in Boxer can be constructed from a data box containing other boxes, in row and column positions. Each of these inner boxes represents a cell.

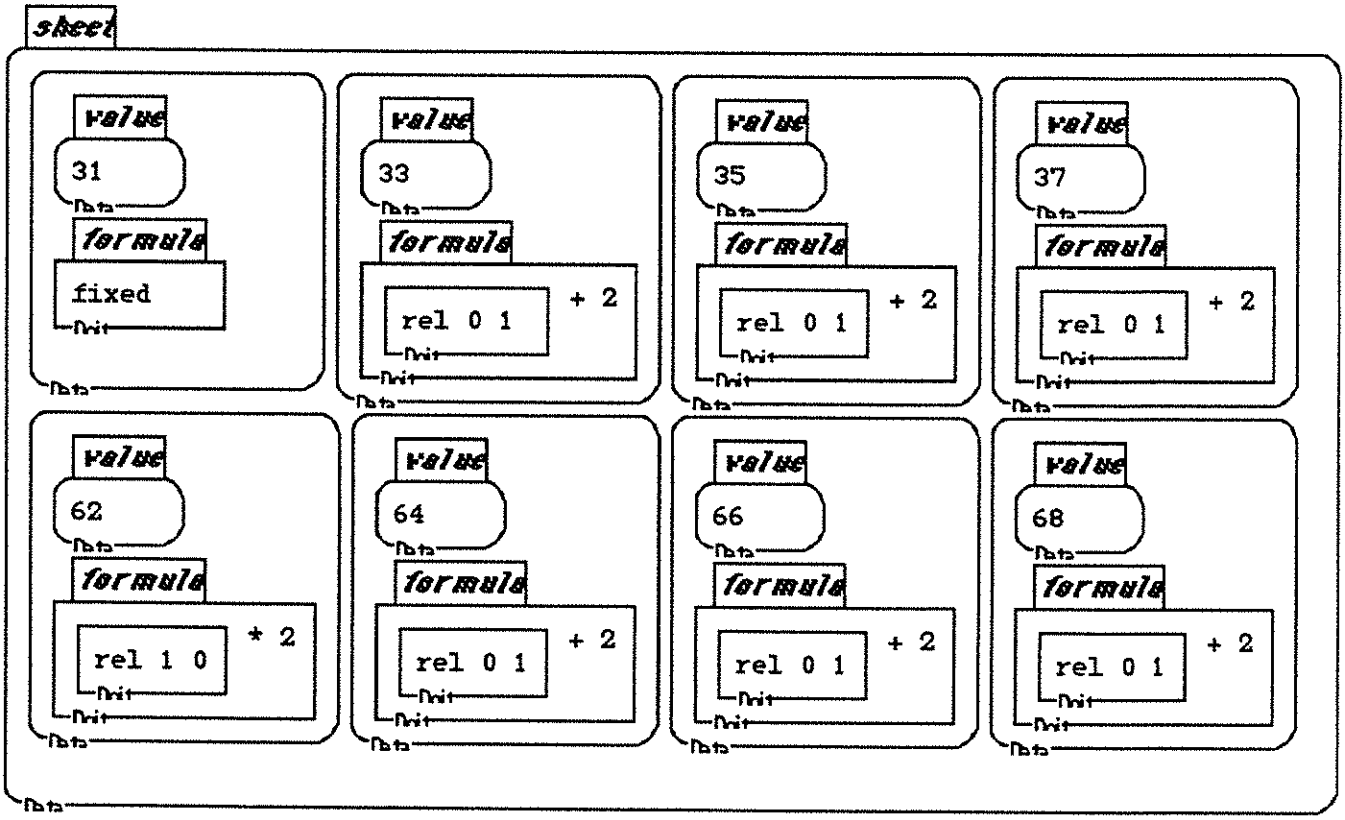

Figure 1: A Boxer Spreadsheet comprising a box of boxes, each containing a value and formula box

Cells have 2 properties, a value and a formula. Again this substructure is represented using boxes. A value is a data box and a formula (because it contains a procedure) is a doit box. Figure 1 depicts a Boxer spreadsheet.

[1] This article does not purport to offer a comprehensive introduction to Boxer. diSessa, et. al (1991) is a good reference for that purpose.

Name tags are attached to these boxes, allowing reference by name to each cell's "value" and "formula." Note that every box in Boxer contains a naming environment; therefore, multiple boxes can have the name "value" as long as each is within a different box. Thus each cell can have named value and formula boxes. This data structure, which supports a running spreadsheet, can be created in Boxer by direct manipulation WYSIWYG operations in a few minutes.

Note also that cells are unnamed. Boxes do not need to have names. The cell boxes in the spreadsheet will instead be referenced by Boxer's generic row and column accessing operations, and by the relative referencing operator "rel."

Program structure

The one major procedure necessary to make a Boxer spreadsheet is the recalculation procedure, herein called "Calc." Calc is written using programming language primitives provided by Boxer (Figure 2). Boxer is a procedural, interpreted, textual language.

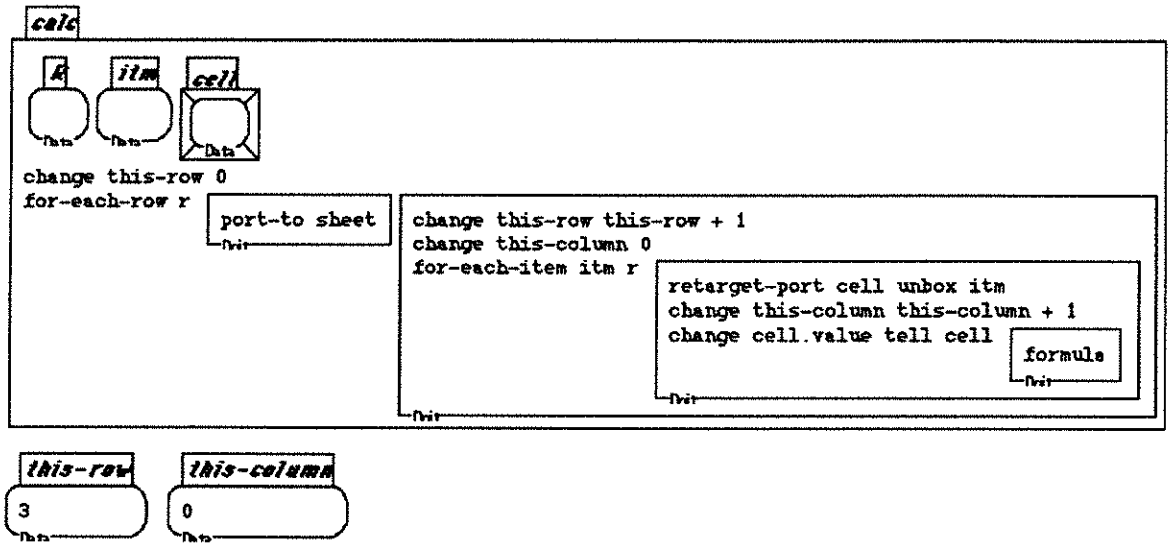

Figure 2: Calc Procedure, as it appears in Boxer

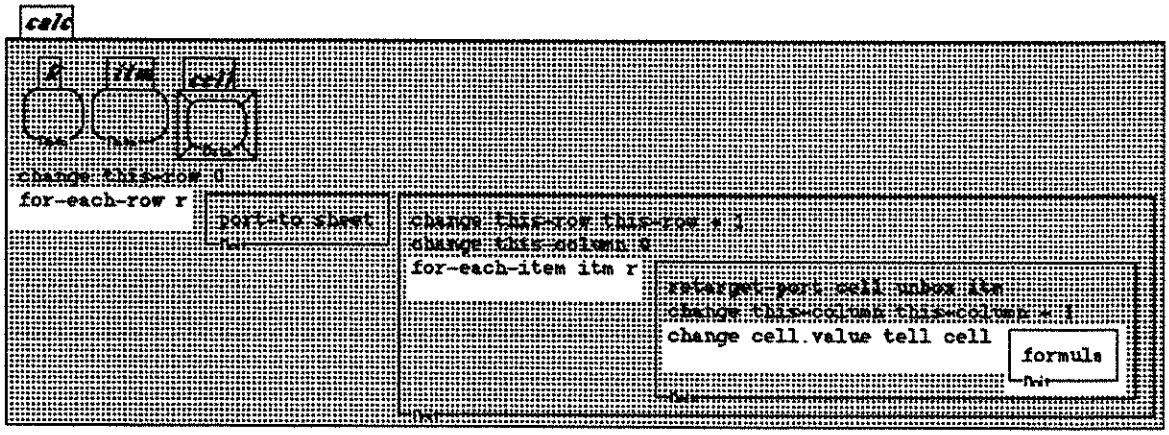

Figure 3: Calc Procedure, highlighted for readability

The heart of the calc procedure (Figure 3) is an embedded set of iterative loops. The outer loop, "for-each-row," iterates through the rows of the spreadsheet. The inner loop "for-each-item," iterates through the cells in the row. As each cell is accessed, the cell's value is changed to be the result of executing the cell's formula.

Two local variables are necessary. The variable "r" holds the contents of each successive row. The variable "itm" holds each successive cell in the row. Note that local variables in Boxer simply appear as named data boxes within the doit box. Also note how Boxer uses boxes to show the nesting of the loops. These are both examples of powerful use of a spatial metaphor to depict more abstract kinds of containment — containment of variable within a local environment and containment of a sub-process within a process.

A third local variable is introduced for convenience. The "cell" variable is a special kind of data box, called a port. A port is a pointer to a non-local data box; it is a "port-hole" or window looking onto another location. This is useful here because, in conjunction with the iteration commands, it allows Boxer to refer directly to successive cells in the spreadsheet. Each time through the loop, "cell" is a port to a different cell in the spreadsheet.

Within this iteration, recalculation is a one-line affair: "change cell.value tell cell formula." This changes the value of the cell to be the result of executing (hence the "doit" box) the formula procedure within the cell. The "tell" command tells Boxer to execute this cell's definition of "formula", and not one in another box.

The ancillary code in the calc doit box maintains two global variables, "this-row" and "this-column." These variables store the row and column number of the cell being calculated. This information is necessary for relative referencing. Note that a global variable in Boxer is just a named data box outside the boundaries of a doit box, such that it can be seen by other doit boxes. This is an example of the use of a spatial metaphor; containment in boxes is used to specify scope of variables.

Using the Boxer Spreadsheet

The spreadsheet is used by specifying values and formulas in cells. The user can create any number of cells (up to available memory) and fill them by using standard Boxer WYSIWYG operations. Formulas can use all Boxer primitives; Boxer includes all the standard functions of arithmetic, trigonometry, etc. Absolute references can be made either by naming a cell (attaching a name tag with WYSIWYG operations) or using the standard row and column accessing commands.

In addition, there is one new procedure added to enable relative referencing. This is the "rel" doit box. This three line procedure computes an absolute reference using the input values and the global state variables "this row" and "this column." Then the primitive "RC" is used to access a cell by row and column, and the "value" data box of that cell is returned.

Summary — Controlling Complexity

As this example demonstrates, the computational abstractions provided by Boxer are sufficient for building a spreadsheet. Indeed, the task can be accomplished readily and elegantly. Data and Doit boxes provide the necessary data structures for representing a spreadsheet, with cells containing values and formulas. The principle of spatial metaphor allows the conceptual containment of values and formulas within boxes to have a direct graphical representation — containment of data and procedures boxes within cells. Furthermore, spatial metaphor gives elegant expression to abstract senses of containment that are fundamental in computer science, subprocesses and local variables.

The iteration primitives provide a sufficient mechanism for controlling recalculation, and with the addition of the "rel" procedure (written in Boxer), Boxer provides mathematical primitives for specifying relationships among cells.

Displaying and editing of the spreadsheet structure is supported by Boxer without any additional code, an example of naive realism (concreteness). An addition aspect of naive realism is worth subtle, but noting. The encapsulation of two named boxes

for value and formula in a box for the cell is an intuitive way of depicting a conceptual model of a spreadsheet. Similar drawings could be produced in any number of applications. In Boxer, this *drawing* is also a full *computational instance* of a data structure; there is no need for declarations of the record structure, allocating memory, filling the record contents, etc. Thus, through the principle of naive realism a *graphical depiction* of a conceptual model and *program code* that implements that model are one and the same. Put simply, in Boxer, a picture of a spreadsheet's structure *is* the actual data structure *and* the user interface for the data structure.

Generality of the Boxer Spreadsheet

This section illustrates the generality of Boxer by presenting a number of extensions to the basic spreadsheet. The first example, a spreadsheet of vectors, illustrates the power of using boxes as a general building block for data structure. The second example, a spreadsheet of graphs linked by functional transformations, presents a pedagogically useful generalization of a spreadsheet that would be hard to replicate using commercially available technology. The third example, a spreadsheet of musical phrases shows that rather extreme (and useful) generalizations of the spreadsheet idea are made possible by Boxer's design.

Vector Sheets

Data boxes are not limited to containing single values. Therefore, unlike a traditional spreadsheet, the value of a cell can be a series of values. This can be used to create a spreadsheet that computes over vectors.

This extension is nearly trivial. The main addition is a set of "doit" boxes that provide operations on vectors. For the purposes of illustration, two such procedures are presented, "scale" and "dot-product" which carry out scalar and dot-product multiplication respectively. Other vector operation could be provided similarly. Figure 4 presents a vector spreadsheet that computes using scale and dot-product. (By extension, cells could easily contain matrices).

Figure 5 shows the dot-product doit box ("scale" is similar). Dot-product inputs two vectors, v1 and v2. It computes the dot-product recursively by a divide and conquer method: one pair of elements at a time is multiplied, leaving the rest of the pairs to the next call of the procedure. If v1 is empty, then the computation terminates and the empty box is returned. Otherwise, a new box is constructed using Boxer's "Build" primitive. Build constructs a new box from a template. In this case the template contains two parts, represented by the two boxes within the template. Figure 6 shows

the specification of the parts of the template (a Boxer user would see the parts by opening the boxes with the mouse). The first part of the constructed box is the result of multiplying the first element in each vector. The second part of the constructed box is the result of computing the dot-product of the rest ("butfirst") of the vector.

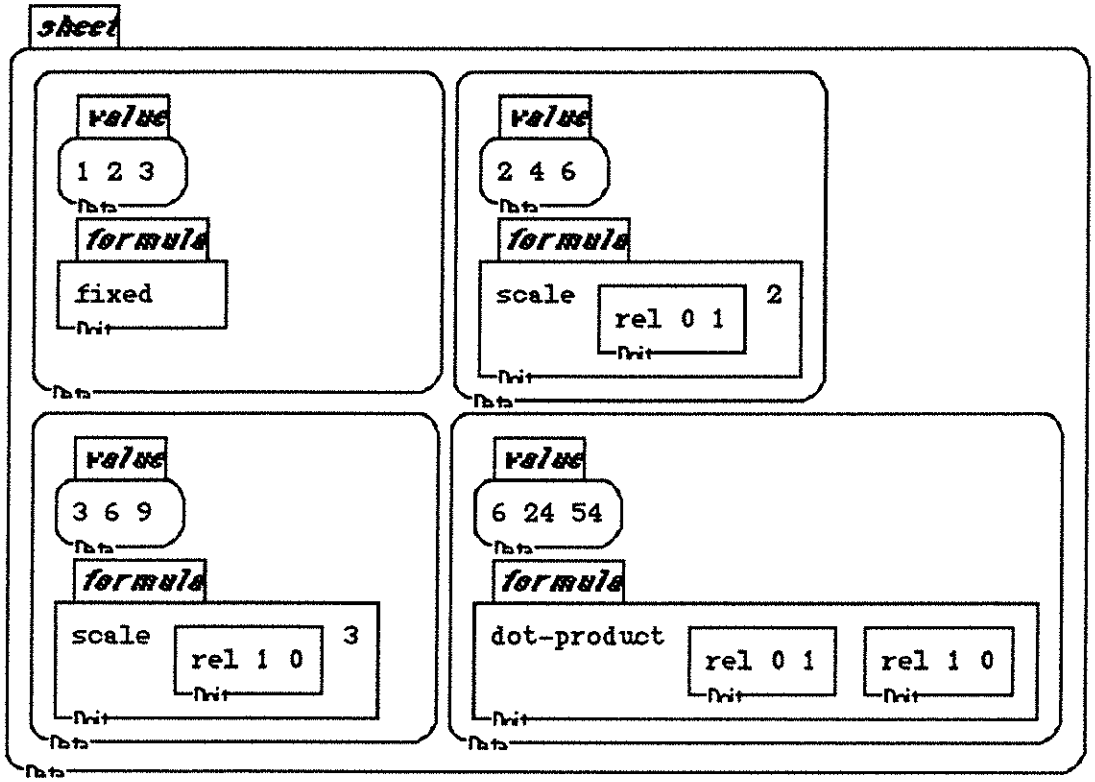

Figure 4: A spreadsheet of cells containing vectors.

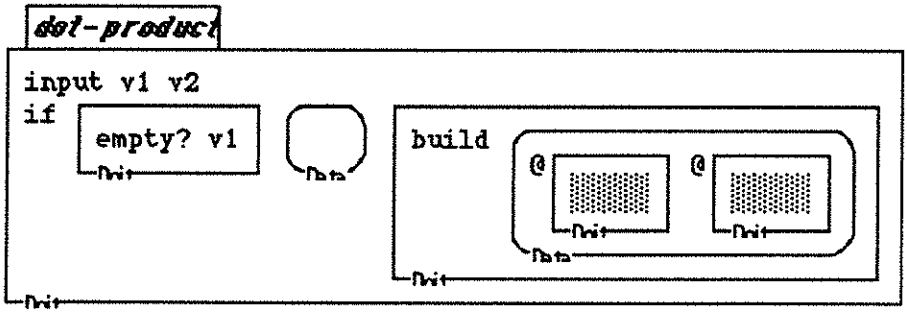

Figure 5: A procedure for computing the dot-product of two vectors

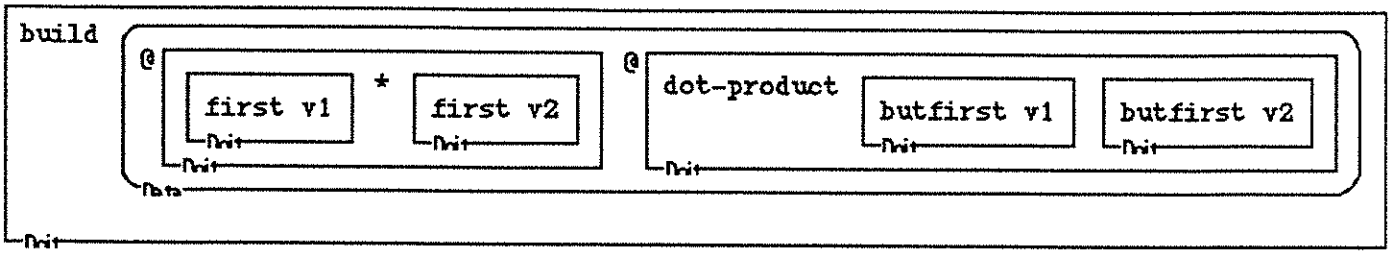

Figure 6: Detail of the dot-product procedure showing the Build template.

Graph Sheets

Commercial spreadsheets offer charting capabilities. Boxer, likewise, can produce bar, line, area, and pie charts. These capabilities are provided in Boxer by way of doit box procedures that move a "sprite" or pen on a drawing surface. Rather than show basic charting capabilities equivalent to those offered on commercial spreadsheets, this section presents a useful generalization: a spreadsheet in which each cell is a graph, and formulas express functional relationships between graphs.

This generalization could be useful because students often have great difficulties understanding the relationships between graphs. For example, in physics, students have great difficulty relating graphs of position, velocity and acceleration (Haertel, 19xx; van Zee & McDermott, 1987; McDermott, Rosenquist, & van Zee, 1987). Similarly in mathematics, students have difficulty relating symbolic, numerical, and graphical representations of a function (Kaput, 1992; Leinhardt, Zaslavsky, & Stein, 1990). A graph sheet enables students to provide different starting graphs (in the top left corner) and then to observe transformations that result from applying various functions to the starting graph. It also lets students play with the relationship between symbolic, numerical, and graphical depiction of a function. Figure 7 shows a graph sheet.

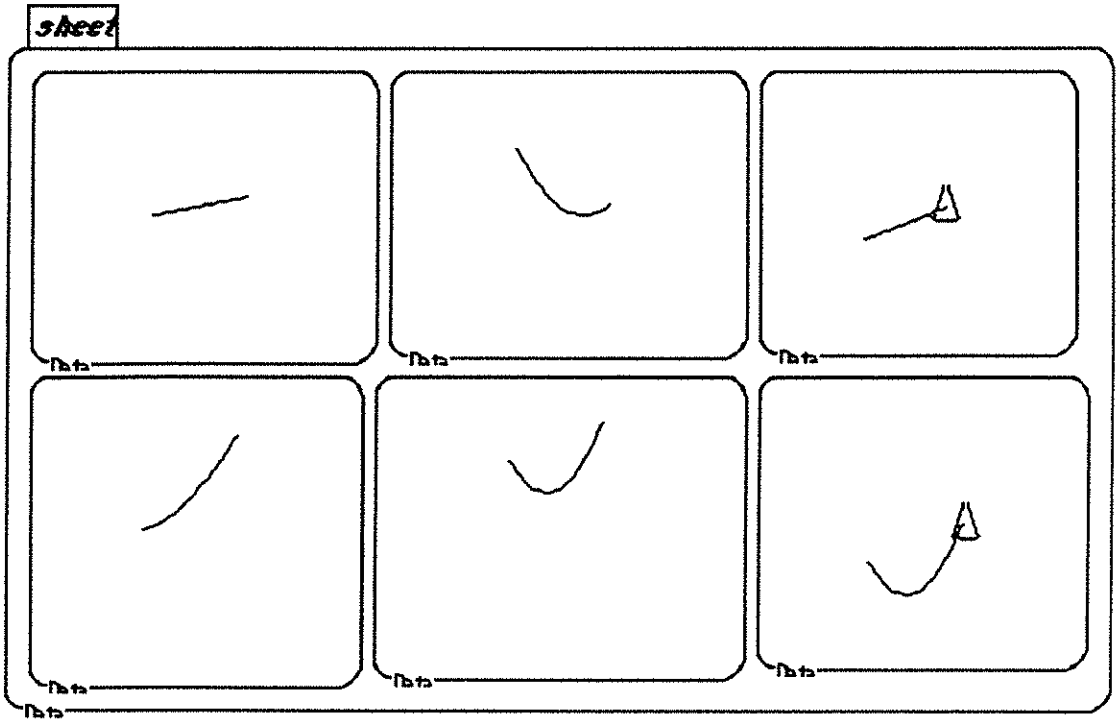

Figure 7: A Spreadsheet in which each cell is a graph

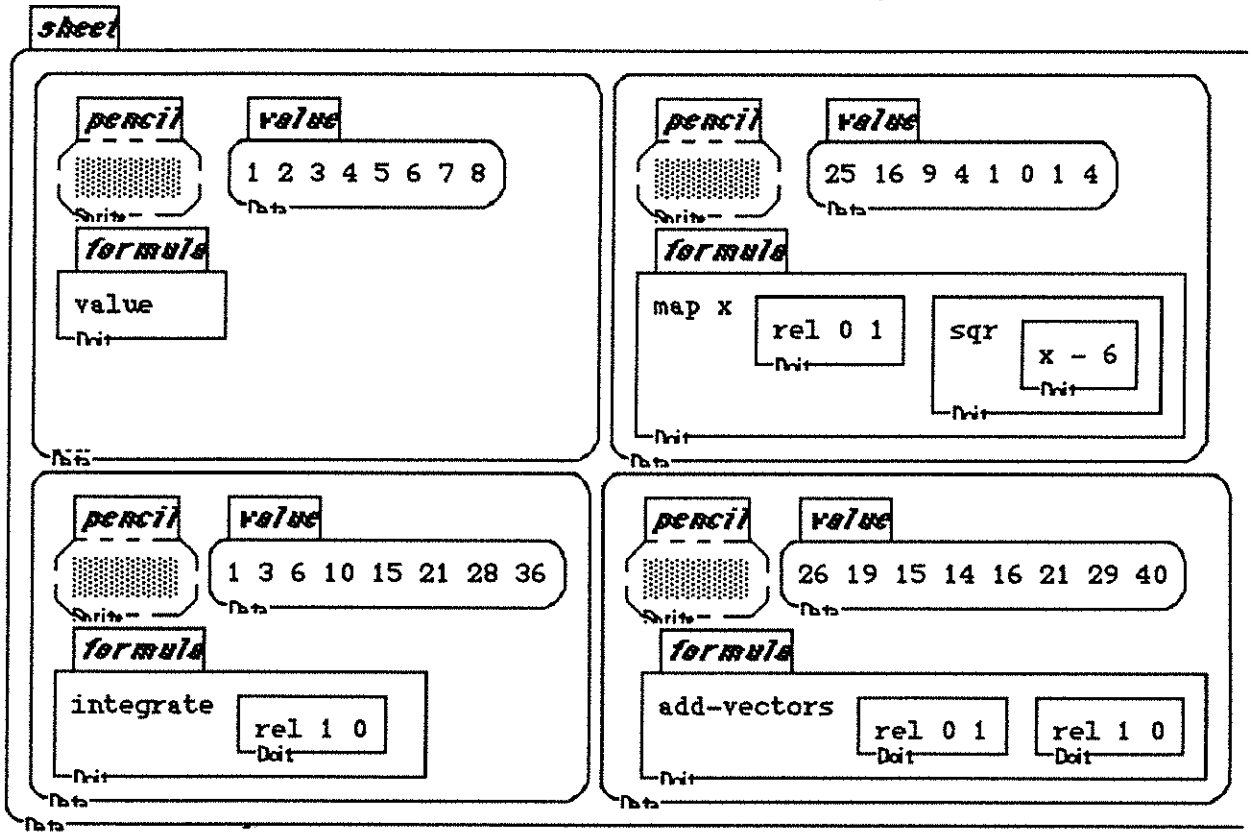

Figure 8: The Data Structure of a Graph Sheet

Figure 8 shows the implementation of the graph sheet. Data boxes in Boxer can have 2 "sides" — data structure and a drawing surface. To make a graph sheet, graph-data boxes are used for each cell. The data structure, as in previous spreadsheets, contains a formula doit box and a value data box in each cell. For the graph sheet, a "pencil" is added to each box. The boxed named pencil is a sprite box. Sprite boxes can draw on the drawing surface of the graph-data box in which they reside. Drawing is accomplished by moving the pen to successive x and y positions using the "setxy" primitive.

The value and formula boxes for the graph sheet are similar to the value and formula boxes for the vector sheet. Each value is the vector of y-values of the graph. Each formula is an operation on a vector. Three new vector functions are illustrated, integration, vector addition, and a generic mapping function. In this case, the generic mapping function is used to apply the square function ("sqr") to the graph in the adjacent cell. Each of these new functions (integrate, add-vectors, and map) are implemented as recursive doit boxes similar to the "dot-product" box.

Drawing the graphs is accomplished by adding a single line to the calc procedure. Immediately after calculating the value for a cell, the pencil in the cell is told to plot the value of the cell (Figure 9). Plot is a doit box written in Boxer that moves the pencil to successive x and y locations (Figure 10).

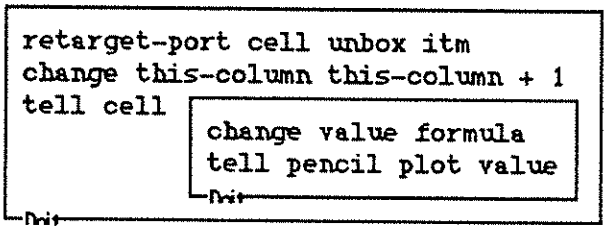

Figure 9: "Tell Pencil Plot Value" added to calc doit box to draw the graph

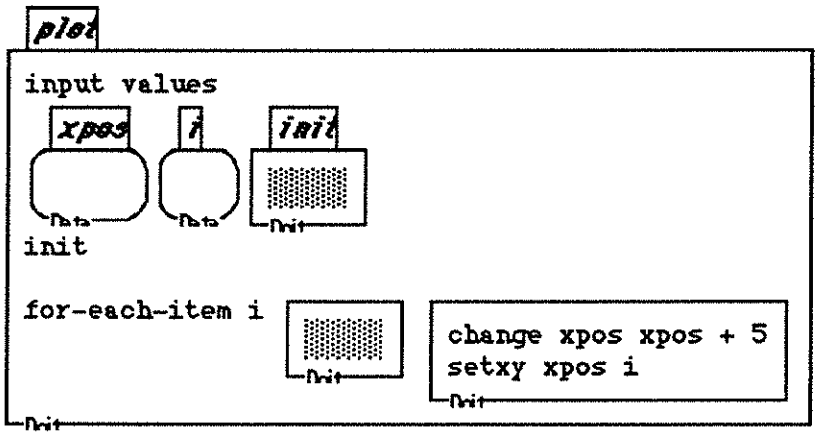

Figure 10: The procedure for plotting a graph.

To use the graph sheet, a student or teacher can make any number of graph-data cells. After formulas are typed into the "formula" doit box of each cell, the calc procedure is run, and the graphs are updated. Changes to the value of the initial graph will cause corresponding changes to all successive graphs. Thus the graph sheet maintains the graphical relationships specified by the mathematical function in the formula box of each cell.

Sheet Music

Sheet music is presented here as an example of a more extreme generalization of a spreadsheet. Sheet music is a spreadsheet in which each cell contains a musical phrase. Successive cells can generate additional musical phrases by transforming preceding musical phrases. Thus a large musical composition can be specified an additional motif, and a list of transformations.

The opening movement of Beethoven's Fifth Symphony is a well-known work susceptible to this sort of specification — Beethoven builds the movement from transformations of the initial 4-note motif ("da, da, da, dum"). Using Sheet music, a student could explore production of melodies modelled on Beethoven's Fifth, or could try their hand at modifying or extending Beethoven's work.

Sheet music is inspired by the work of Jeanne Bamberger (Bamberger, 1992). Professor Bamberger has drawn attention to the fact that we psychologically experience music in phrases, not individual notes. Moreover, melodies are heard in terms of relationships between successive phrases. Ordinary musical notation, which is based on notes in a rigid metric grid, does not highlight the psychology of music hearing, and can distract and alienate students from their own musical sense.

The purpose of sheet music is to allow students to focus on music in terms of phrases and relations between phrases.

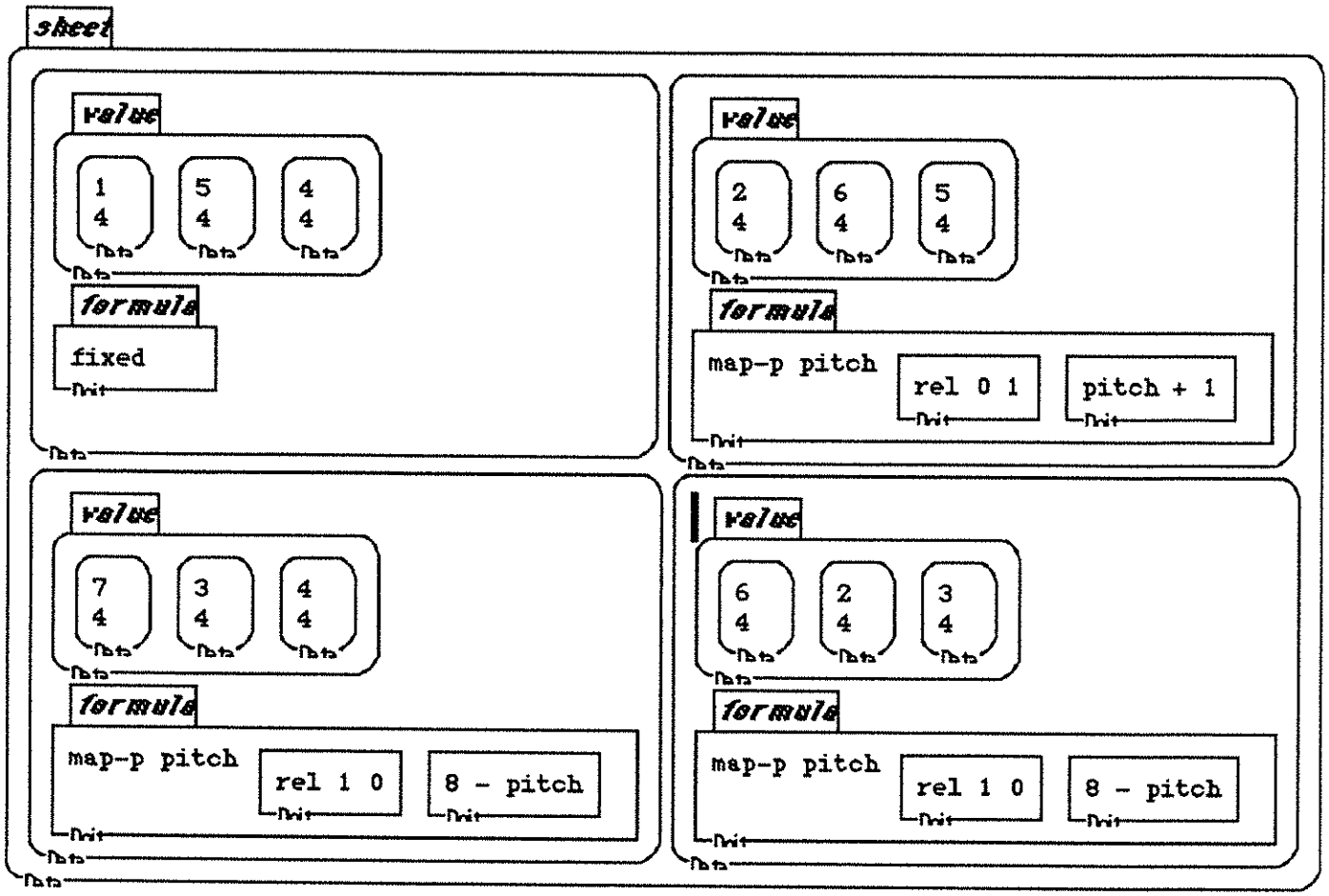

Figure 11: Sheet Music

Sheet music, like other spreadsheets, is constructed of cells, each with a value data box and a formula doit box. The value data boxes, in this case, illustrate a further generalization: instead of being single values or vectors, the value is a data structure constructed from data boxes. In particular, a melody is represented in Boxer as a box of notes. Each note is a box containing a pitch and a duration. Whole numbers specify pitches on a major scale, and also a length of time.

The initial phrase in the top-left cell of Figure 11 contains the musical notes "C-G-F," each played for a quarter note duration. The next phrase to the right, moves these pitches up 1 note to "D-A-G." This movement is specified by the formula of the cell, which applies the operation "pitch + 1" to each pitch of the adjacent cell's melody, via the mapping operation "map-p." Map-p is a simple Boxer doit box that applies a doit box to each each pitch of a melody, returning the transformed melody.

A similar doit box called "map-d" can be used to transform durations.

As in the graph sheet, the music sheet adds a line to the calc procedure. The added line in this case, "play cell.value" causes each cell's melody to be played as it is recalculated.

Additional Features

Several other features of Boxer are directly useful in spreadsheets. First, Boxer offers control of the display by means of shrinking and expanding boxes, or making them temporarily invisible. In all the examples presented herein, the value and formula boxes have been shown in their expanded state. With simple mouse clicks, a user can shrink and hide all the formulas (Figure 12). In addition, every box has a "closet" which can be used to hide sub-boxes from the user. The closet is opened and closed with a key or menu selection. Using this facility, the formula of each cell can be placed into the closet, thus hiding it from view. Although these mechanisms do not exactly parallel those of a commercial spreadsheet, they offer the same functionality — hiding of information and conservation of screen space.

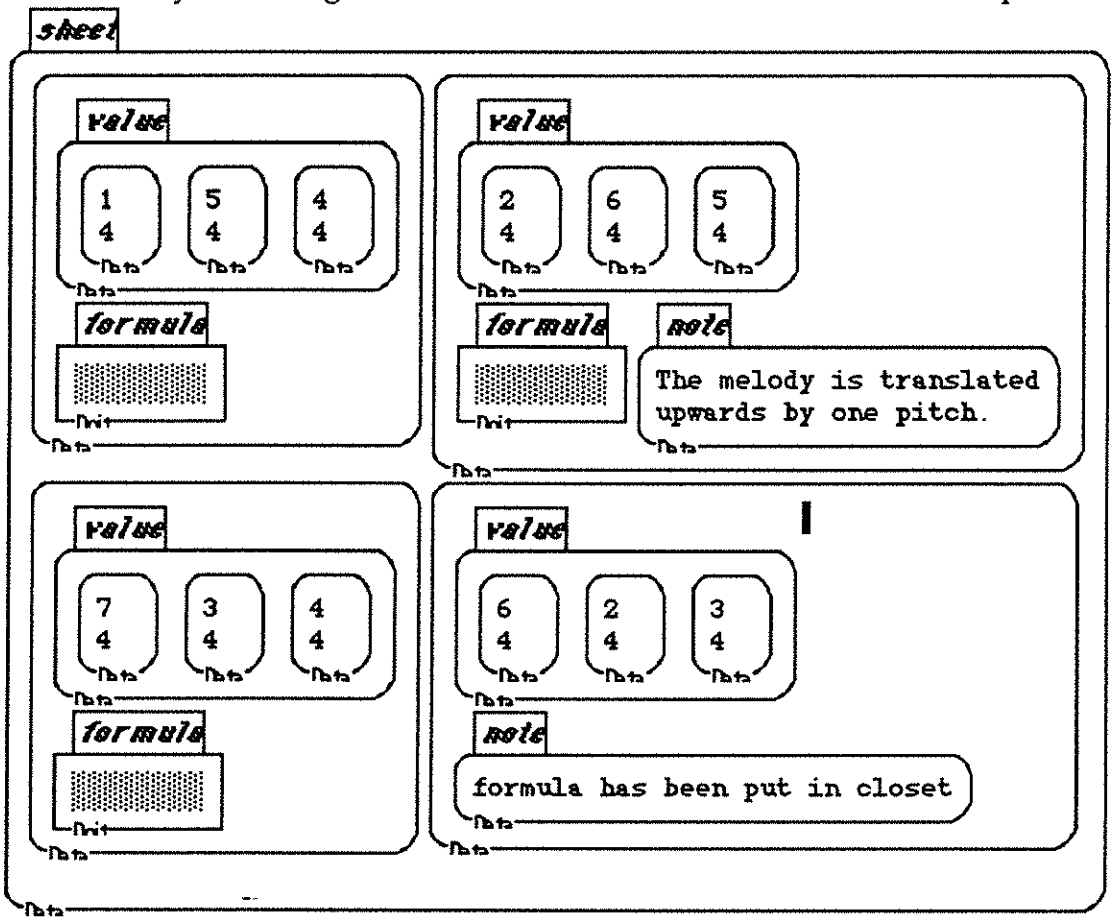

Figure 12: Shrinking Boxes and Adding Notes to a Boxer Spreadsheet

Commonly spreadsheets allow cells to be annotated with comments about their purpose or history. These notes are typically provided through additional user interface commands. In Boxer, no additional commands are necessary. To add a note, the user creates a data box named "note" (by WYSIWYG actions) and types a comment as appropriate. In Figure 12, two cells have notes added.

Another feature of commercial spreadsheets is the ability to write scripts. Typically two kinds of scripts are supported. A *function* script specifies a new mathematical function by composing other available functions. A *command* script automates a series of modifications to the spreadsheet, e.g. adding cells. In Boxer, an ordinary doit box is sufficient for writing new functions (as in the dot-product example). If the doit box is placed outside the spreadsheet, the new function will be globally available. Moreover, command scripts can be easily written using standard Boxer primitives. Figure 13 illustrates a "Fill Right" script, which copies the last cell of a specified row a given number of times. The doit box works by repeating the operation of appending a copy of the last cell in the row the given number of times.

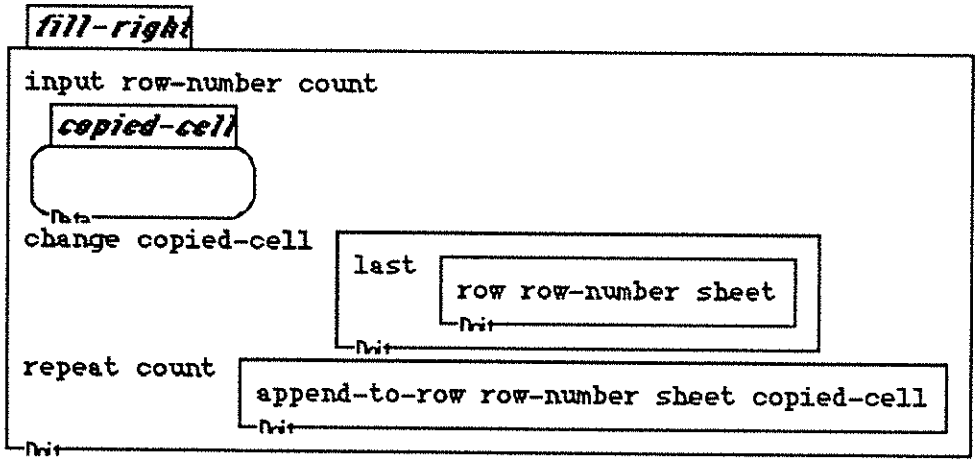

Figure 13: Fill-right script or command macro

Summary — Generality

The presented examples demonstrate that Boxer readily supports generalization of the spreadsheet concept.[2] The examples of vector, graph, and music cells would be

[2]Additional generalizations could take advantage of Boxer's hierarchical nature. For example, Joy Nunn of Ballarat University College, Australia, is working on an ecological spreadsheet in which different hierarchical levels of the food chain are representing by a hierarchically-structured spreadsheet.

hard to implement in any commercially available spreadsheet, as the value of cells are usually limited to single numeric values or text strings. Moreover, each of these generalizations has apparent pedagogic utility. For example, the production of linked representations among symbolic, numerical, and graphical representations is a important topic in mathematics education (Kaput, 1992). By allowing students to focus on relationships between adjacent cells, rather than calculation, generalized spreadsheets may help students build conceptual understanding of relationships. As the graph sheet shows, Boxer reduces the implementation time for linked representation software to an easy day's work.

The ease of constructing these examples in Boxer is directly due to the generality of basic Boxer structures. Every data box can contain a complex data structure consisting of text and boxes. Thus the "value" data box can readily be generalized from simple numeric values to extended data structures. Similarly, all data boxes can have a graphics drawing surface. Controlling the display, scripting commands and functions, and adding notes are all features that are supported by general-purpose Boxer data structures.

Limitations of the Boxer Spreadsheet

Some limitations of Boxer Spreadsheets, however, are not easily overcome. Principally, these limitations are in the area of user interface. Commercial spreadsheets have highly-tuned, special purpose user interface that are not easily replicated in Boxer. For example, one can usually specify a reference to an adjacent cell, or group of cells, by a click and drag operation. These is not easily duplicated in Boxer. Moreover, one can "freeze" a set of cells in place, while scrolling the others. This behavior would be almost impossible to implement directly in Boxer. In addition, commercial spreadsheets have a wide range of options for formatting the values of cells, including left, center, or right alignment, bold or italic typefaces, various monetary formats, etc. Boxer currently does not support these options.

One further feature of commercial spreadsheets is the ability to recalculate based on dependencies among cells, rather than in strict row and column order. The Boxer spreadsheets presented so far do not have this feature. However, it could be readily added to Boxer spreadsheets by writing a 2-pass recalculation procedure. In the first pass, the recalculation procedure could record the dependencies among the cells in a data box. These could then be sorted. In the second pass, iteration could proceed in the order listed in the sorted data box. Boxer can also attach a "modified-trigger" doit box to any data box. This could be used to record which cells have been modified, and thereby avoid total recalculation of the spreadsheet.

Discussion

The purpose of this article is to highlight the gains made possible by developing and using reconstructable computational media. Boxer offers control of complexity and enables generalization through principles of naive realism and spatial metaphor. The spreadsheet examples showed the Boxer's computational abstractions allow spreadsheets to be implemented elegantly and simply. The generality of Boxer's computational structures allow ready extension of the spreadsheet concept of additional kinds of quantities and presentations.

The uniqueness of Boxer in this regard can be highlighted by considering, briefly, the difficulty of writing a spreadsheet in other conventional computer languages. Four classes of competitors will be considered: Logo, C and Pascal, Hypercard, and "Object Oriented" programming.

Logo

Logo offers strong capabilities for educating students about computational concepts. Compared to Basic (the other educational language of its era), Logo offers two key features, recursion and list processing. Recursion is a powerful strategy for specifying complicated computations, and list processing is a very general data representation strategy. Boxer's copy and execute model makes recursion easier to understand (Leonard, 1991) and boxes are a convenient and powerful generalization of lists.

Although Boxer's language is modelled on Logo and subsumes its features, several capabilities unique to Boxer are essential for developing spreadsheets. The most obvious is Boxer's capability to display and edit boxes on the screen. Logo conventionally offers two display capabilities: turtle graphics or textual output. Writing code to edit spreadsheets using Logo would be quite difficult.

Several deeper issues also arise. First, Logo has a single, flat naming environment. Thus one cannot have a "value" and "formula" defined in multiple cells. Second, traditional implementations of Logo do not allow modification of the substructure of variables; all modification must be carried out by setting a global variable. This makes recalculation of a spreadsheet unnecessarily complex. Third, procedures cannot easily be manipulated in Logo — there is no system representation like a doit box that can be manipulated as part of a data structure.

Hypercard

Hypercard is often viewed as a competitor to Boxer, as both are object-based computational media. Hypertalk (along with spreadsheet scripting languages) has perhaps been the most significant commercial advance in making programming a common activity for computer users. However, Hypertalk is a scripting language, designed to automate tasks, whereas Boxer has been designed to make computation understandable. Thus Hypercard is more tuned towards rapid programming of simple processing tasks, whereas Boxer is more tuned towards conceptualizing, expressing and communicating computational ideas.

One could begin to implement a spreadsheet in Hypertalk using fields as cells.[3] Like boxes in Boxer, fields are self-displaying and allow user interaction. The value of a cell could be contained in a field's text, and the formula in a "script" attached to the field. Using iteration and send commands, a Hypertalk script could be written to recalculate each cell by sending a message to each field to run the formula script.

Beyond this, however, several difficulties become apparent, directly traceable to a less powerful use of spatial metaphors. First, Hypercard allows access to fields by numbering them in a list, in their order of creation. Thus there is no easy way to refer to fields by row and column indices. The only possibility to refer to the pixel position of each field's top-left corner, a cumbersome method for referring to spreadsheet cells. This is a fairly severe limit in computational expressiveness. Second, Hypercard fields can only contain text, and thus offer no substructure as boxes do. Indeed, not that the basic computational structure of a "card" is not helpful in building a spreadsheet, while a "box" is. This makes graph sheets and music sheets more cumbersome to implement, and is a restraint on generality. Third, Hypercard's display features are somewhat less suited to spreadsheets than Boxer is. It is difficult to see a set of formula scripts side by side, as scripts, by default, are very large windows occupying most of the screen. The fixed size of a card limits the number of fields that can appear on a card. This limits how well Hypercard communicates computational ideas. In general, Hypercard is more limited than Boxer in expressing hierarchy and structure, as well as in the display metaphor.

C and Pascal

Most commercial spreadsheets are written in either C or Pascal. Nonetheless, doing so is a daunting task best suited for a professional programmer over the course of a year or more. It is useful to ask "how does Boxer reduces this task to a matter of days or hours?"

[3] The alternative, using cards as cells, is unappealing because one can only see one card at a time.

Although C or Pascal provide very general capabilities, the abstractions provided tend to be at a very low level, and thus there are several gaps between the abstractions available and the desired functionality. First, whereas Boxer provides high-level display and editing capabilities in terms of boxes, C or Pascal provides neither. Display is in terms of characters and pixels, and editing in terms of keystrokes. Hence writing the interface portion of a spreadsheet is a major task in itself, finessed in Boxer by general interface principle of naive realism. Second, Boxer is an interpreted language, and thus inherently can evaluate text written by the user as mathematical functions. C or Pascal are compiled, and thus the programmer must write a language interpreter to translate user expression of mathematical functions into executable code. Third, the basic data structures of C or Pascal force early commitment to the contents of variables, which thwarts attempts to generalize the values of spreadsheets to be graphs, vectors, or musical data structures. Boxer instead provides a single, more general data type. Thus Boxer accomplishes a remarkable reduction in the programming task by providing suitable computational abstractions of a general nature.

It is worth briefly mentioning the emerging category of object-oriented languages, such as C++ and Object Pascal. Object-orientation is sometimes misleadingly used as a litmus test for accepting a new language. Although Boxer (like Hypercard) has some object-based features, it is not an object-oriented language. In particular, it does have classes or inheritance. Object-orientation, however, does not automatically lead to elegant or general programs. It is not much easier to write a spreadsheet in off-the-shelf C++ than it is in straight C. The reason is that object-orientation does not inherently overcome the low-level graphics and lack of a language interpreter in C or Pascal. For educational computing users, an object-based system like Boxer, which has appropriate computational abstractions like data and doit boxes, can be more powerful than a completely object-oriented system like C++, which is more general but offers computational abstraction at too low a level.

Conclusions

This article has illustrated how a reconstructable computational medium comprised of a general "box" data structure and a Logo-like language can offer means for controlling complexity and enabling generality. Boxer is not designed for spreadsheets — its target domains are primarily science and mathematics simulations, as well as introductory programming. Yet the primitives provided by Boxer allowed general and elegant implementation of spreadsheets. In conclusion, I

consider some reasons why Boxer is important and useful innovation.

First, Boxer is of interest in computer science instruction. The focus in computer science instruction, especially in the beginning, should be on general computational concepts like data structures and controlling computational processes. A spreadsheet offers an excellent opportunity to explore these issues in a context with obvious real-world applications. However, using a spreadsheet as an example with a conventional language would be impractical, as the details of providing a user interface and an interpreter would dominate. The Boxer spreadsheets show that Boxer eliminate these distracting issues, and allow students to focus on the heart of the computer science problem that spreadsheets present —how to control complexity and support generalization.

Second, a recurring theme of debate in educational computing is whether students should learn programming or not (Kaput, 1992; Kurland, Pea, Clement & Mawby, 1986). As the examples illustrate, Boxer can dissolve this debate because it blurs the boundary. At the onset, a Boxer spreadsheet, like other Boxer programs and microworlds, can just be used. However, for the inquisitive student or teacher, the workings of the spreadsheet are available to inspect and modify. Students can open up the structure of the tool, and explore how it works. Thus Boxer is an example of a glass box tool (Wenger, 1987): it is an object that allows learning about internal computational *means* while accomplishing useful computational *ends*. Glass box tools are important in education because they allow students to ask "how" and "why" questions in addition to asking , "what?"

In addition, as noted above, many problems which are conceptually amenable to spreadsheet analysis can not be implemented within commercial spreadsheets. For example, a science or mathematics teacher interested in focussing students attention on relationships among graphs can use a Boxer graph sheet in a way that commercial spreadsheets do not support. Similarly, Boxer makes spreadsheet concepts useful for domains such as music and ecology. Thus by reconstructing the boxes provided in Boxer, a teacher can make a spreadsheet to meet his or her own needs. Boxer's reconstructability should support the growth of a community in which computational ideas are readily expressed, explained, and extended.

Third, Boxer is a valuable example of the payoffs that can result from articulating principles for educational computing, such as spatial metaphor and naive realism. Boxer, unlike low-level languages such as C or Pascal, and unlike special purpose languages, such as those provided with databases and spreadsheets, offers an understandable, powerful metaphor — the box — which can be used to build an

astonishing variety of innovative tools, microworlds, and applications. Through the principles of spatial metaphor and naive realism realized in Boxer's structure, a complex application like a spreadsheet can be reduced to a compact, understandable, general description. This is not a result that can easily be replicated in available commercial products, and that is to be expected. Commercial products are driven by the goal of productivity, and attributes such as speed and infallibility are of the essence. However, for educational purposes, productivity is not enough. We want students to understand, communicate, and learn, not just use tools effectively. With respect to these ends, Boxer is an outstanding positive exemplar of the results of a sustained effort at the principled design of educational computing software.

Acknowledgements

This work was supported by and performed at Sunrise Research Laboratory, Royal Melbourne Institute for Technology, Melbourne, Australia. I am grateful to Liddy Nevile, David Williams, and Andy diSessa for help in the development of the spreadsheets and this article.

References

Abelson, H. & Sussman, G.S. (1985). *The structure and interpretation of computer programs*. Cambridge, MA: MIT Press.

Arganbright, D. (1984). Mathematical applications of an electronic spreadsheet. In V.P. Hanson (Ed). *Computers in mathematics education, 1984 Yearbook.* Reston, VA: National Council of Teachers of Mathematics.

Bamberger, J. (1992). *The mind behind the musical ear*. Cambridge, MA: Harvard University Press.

diSessa, A.A. (1986). From Logo to Boxer. *Australian Journal of Computer Education*, July, 8-15.

diSessa, A.A. & Abelson, H. (1986). Boxer: A reconstructable computational medium. *Communications of the ACM, 29*, 859-868.

diSessa, A.A., Abelson, H., & Ploger, D. (1991). An overview of Boxer. *Journal of Mathematical Behavior, 10*, 3-15.

Goodfellow, D. (1990). Spreadsheets: Powerful tools in science education. *School Science Review, 71*, 47-57.

Graham, I. (1987). The application of spreadsheets to data analysis in biology. *Journal of Biological Education, 21(1)*, 51-56.

Haertel, (19xx). wrote to Herman and waiting for updated reference.

Healy, S. & Sutherland, R. (1990). *Exploring mathematics with spreadsheets.*

London: Blackwell.

Kaput, J.J. (1992). Technology and mathematics education. In D. Grouws (Ed.), *Handbook on research in teaching and learning mathematics*. New York: Macmillan.

Kurland, M., Pea, R., Clement, C., and Nawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of educational computing research, 2,* 429-458

Leinhardt, G. Zaslavsky, O., & Stein, M.K. (1990). Functions, graphs, and graphing: Tasks, learning, and teaching. *Review of Educational Research, 60,* 1-64.

Leonard, M. (1991). Learning the structure of recursive programs in Boxer. *Journal of Mathematical Behavior, 10,* 17-53

McDermott, L.C., Rosenquist, M.L., & van Zee, E.H. (1987). Student difficulties in connecting graphs and physics: Examples from kinematics. *American Journal of Physics, 55,* 503-513.

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas*. New York: Basic Books.

Picciotto, H. & Ploger, D. (1991). Learning about sampling with Boxer. *Journal of Mathematical Behavior, 10,* 91-113.

Turner, S.C. (1988). The use of spreadsheets in teaching undergraduate mathematics. *Computers and education, 12(4),* 535-538.

Quinn, M. (1986). The use of spreadsheets in primary mathematics. In Salvas, A.D. & Dowling, C. (Eds.), *Computers in education*. North Holland: Elsevier, 207-214.

van Zee, E.H. & McDermott, L.C. (1987). Investigation of student difficulties with graphical representations in physics. In J. Novak (Ed.), *Proceedings of the second international seminar on misconceptions and educational strategies in science and mathematics*. Ithaca, NY: Cornell University.

Visch, E. (1990). Make ends meet with a spreadsheet. In McDougall, A. & Dowling, C. (Eds.) *Computers in education*. North Holland: Elsevier. 207-214.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communciation of knowledge*. Los Altos, CA: Morgan Kaufman.