JOURNAL OF MATHEMATICAL BEHAVIOR 10, 3-15 (1991)

An Overview of Boxer

ANDREA A. DISESSA
University of California, Berkeley

HAL ABELSON
Massachusetts Institute of Technology

DON PLOGER
University of California, Berkeley

AN OVERVIEW OF BOXER

This issue of the *Journal of Mathematical Behavior* highlights research emerging from the Boxer group at the University of California, Berkeley, School of Education. Boxer is the name for a multipurpose computational medium intended to be used by people who are not computer specialists. Boxer incorporates a broad spectrum of functions—from hypertext processing, to dynamic and interactive graphics, to databases and programming—all within a uniform and easily learned framework. Most significantly, Boxer is intended to be a new interactive educational medium through which students and teachers can carry out activities spanning a wide range of areas.

When discussing education, the word "medium" often calls to mind books, newspapers, films, or other noninteractive media. This goes hand-in-hand with a model of education where teams of professionals—writers, software developers, content experts—produce materials that incorporate the "right" approach, and where teachers and students are passive consumers. Boxer embodies a very different premise: Students and teachers can shape an expressive computational medium for their own ends. In Boxer, a teacher can create an interactive "worksheet" that incorporates hypertext, graphics, and programs, and students can use and modify these elements and incorporate them into their own work. Research on Boxer is motivated by the hope that such an interactive computer medium could lead to major changes in how students learn, and what they learn.

This research is supported primarily by a grant from the National Science Foundation, NSF-MDR-88-50363, to Andrea A. diSessa. We also gratefully acknowledge a contribution to support students from Apple Corporation, and a contribution of equipment from Sun Microsystems.

Correspondence and requests for reprints should be sent to Andrea A. diSessa, School of Education, Tolman Hall, University of California, Berkeley, CA 94720.

BOXER'S ROOTS IN LOGO

Boxer grew out of Logo, a computer language developed primarily at MIT[1] by Seymour Papert and his colleagues beginning in the late 1960s (Papert, 1980). Logo was a significant advance in comprehensible programming languages: It was the first prominent attempt to apply constructivism[2] to educational computing, and introduced new content areas and modes of learning. Logo made computer programming easy enough so that young children could use programming as a vehicle for exploring geometry (Watt & Watt, 1986). It was also powerful enough so that high school and college students could use it to investigate such topics as differential geometry (Abelson & diSessa, 1981).

Logo has been available for use in schools since 1981. Serious work on Logo, however, preceded school-available technology by about 15 years. The first Logo implementations, made in the late 1960s, ran on research computers that cost over a million dollars. During the 1970s, while the cost of computer technology was falling, Logo researchers developed and refined the language, and carried out extensive pilot studies with teachers and students. These studies addressed two questions: (1) Can students and teachers master this medium and do interesting things of genuine educational value? (2) What can educational researchers learn about learning by watching people work in the Logo environment?

Yet, even before the first Logo implementations were available for schools, a few members of the Logo group at MIT were envisioning a new computer system to overcome what they saw as two major drawbacks in Logo.

The first drawback is that Logo is too difficult to learn. Children can write programs in Logo to do simple things, for example, to draw triangles, squares, and other shapes, but the mechanism by which the language actually works is not made visible and concrete either for students or teachers. It is easy to get started in Logo, but it is difficult to master the language.

The second major drawback is that Logo is essentially just a computer language, and Logo activities are closely tied to the ability to write simple procedures. It would be difficult for children to use Logo to construct a computer-based journal or to make extensive use of a database. Many teachers have been able to use Logo to create simple computer-based microworlds consisting of a few Logo procedures, but Logo's lack of structuring principles, beyond individual procedures, makes it difficult for teachers to organize these into flexible constructs, such as an interactive notebook that students can use and modify.

[1] *Editor's Note.* Although Seymour Papert has unquestionably been the genius behind LOGO—and in this case the word *genius* can be given its fullest meaning—it should be noted that the first NSF contract supporting the development of Logo was to Bolt, Beranek, and Newman, not to MIT, and the Project Director was Wallace Feurzeig.

[2] *Editor's Note.* For a discussion of "constructivism," see, for example, Davis, Maher, and Noddings (1990).

These considerations led to the formation, in 1981, of a new research group in the MIT Laboratory for Computer Science, under the direction of Andy diSessa and Hal Abelson. In contrast to previous work on Logo, their goal was to develop, not just a programming language, but a programming language that would be fully integrated into a multipurpose, flexible medium.

THE DESIGN OF BOXER

An underlying orientation of Boxer is this: If the match between the medium and the programming language is right, then many programming tasks can be automatically subsumed into the natural interaction with the medium, and will not require explicit programming. Consider, as a simple example, changing the value of a program variable. In traditional programming languages, a variable is an abstract association between a name and a value; a variable is changed by issuing an assignment statement. In contrast, if the variable is represented as a visible object on the computer screen, you can change the variable's value simply by editing the screen representation—just as you would edit a piece of text with a text editor—an activity that does not seem like programming at all. This style of interaction is often referred to as a "direct manipulation interface" (Hutchins, Hollan, & Norman, 1986). Boxer differs from most direct manipulation interfaces in that it incorporates a full-fledged programming language.

The Boxer group began by designing a new system based on two principles of learnability. The first one is called *concreteness* (or *naive realism*). Concreteness specifies that all mechanism in the system should be visible and directly manipulable on the display screen. The second underlying principle is that there is a uniform *spatial metaphor* for structure. The root form of Boxer is an object called a *box*. A box may contain text, graphics, programs, or other boxes. A box may be named, in which case, the name is a variable whose value is the box's contents. Named sub-boxes of a box serve as "local variables," their names are visible only inside the enclosing box. Boxes structure files, databases, programs, and everything else in the Boxer system. Boxes allow people to use ordinary spatial intuitions of *inside*, *outside*, and *next to* in order to understand a broad range of computational structures. An extensive discussion of the principles behind Boxer's design can be found in diSessa (1985).

In 1985 the locus of Boxer development moved from MIT to the University of California at Berkeley, where a scenario similar to the earlier Logo work is now being played out by the Boxer Research Group, under diSessa's direction. Like Logo, Boxer development is preceding practical school-available technology by 10 to 15 years. The first Boxer implementation, in 1981, was designed for computers that cost well over $100,000. Today, Boxer can run on a workstation that costs about $6,000, and the development of Boxer is following the same overall 15-year time scale. It appears that, by 1995, Boxer will run on machines schools can afford. In the meantime, members of the Boxer Research Group are

addressing questions similar to those addressed in Logo research: Can students and teachers master this medium? What can we learn about learning from watching people use Boxer?

A Brief Introduction to Boxer

This section introduces some of the basic characteristics of Boxer. In keeping with the articles in this issue, the emphasis here is on how Boxer can be used as a representation language. A more extensive overview of Boxer can be found in diSessa & Abelson (1986); diSessa (1990) reviewed the goals of Boxer from a socially oriented point of view.[3]

Representing Data. Data in Boxer takes the form of *data boxes*. A named data box serves as a variable. Evaluating the name of a data box returns a copy of the box. One of the essential features of Boxer variables is their visibility. The user can always directly see the computational state of the system and, as directly, change it with simple editing commands. This avoids much programming input and output. More importantly, the user can easily see what is going on and feels in control of the system.

The following example is drawn from a project completed by two high-school students during a 6-week Boxer course taught at the University of California, Berkeley, in the summer of 1990. The example has been simplified for the purposes of presentation. The students' project was actually much more complex.

Figure 1a shows a data box called **carbon,** with a value of 3, and a data box called **hydrogen,** that is empty. Figure 1b shows how information about variables can be accessed. When **carbon** is evaluated, its value is returned (as indicated by the copy of the box to the right of the vertical line). When **hydrogen** is evaluated, an empty box is returned.

The value of a variable can be changed simply by editing the contents of the box. Boxer also includes a **change** command that allows variables to be changed under program control. Figure 2 shows how the value of **hydrogen** (which was empty in Figure 1) is changed to a new value, which is twice the value of **carbon** plus 2. (This illustrates the relation between the number of carbon atoms and the number of hydrogen atoms in an alkane, the first class of molecules the students considered in their project.)

Manipulating Structured Data. Any Boxer objects—numbers, multiple subboxes, pictures, programs, and so on—can be placed in a data box. This makes it easy to represent complex data. The **carbon** and **hydrogen** data boxes shown in Figures 1 and 2 represent a natural grouping: the number of atoms in a molecule.

[3]An annotated bibliography of papers about Boxer, as well as reprints of those papers, is available from the Boxer Project, Graduate School of Education, University of California, Berkeley, 94720.

Figure 1.a. Variables in Boxer.

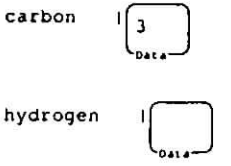


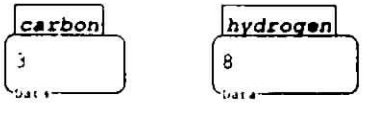
Figure 1.b. Accessing the value of variables.

Figure 3a shows this grouping made explicit by placing the two data boxes into a third data box, called **molecule.** The result is a very simple database.

With the data structured as in Figure 3a, the value of **carbon** and **hydrogen** are local to the **molecule** box. Their values are not accessible outside the molecule box, and one could, without confusion, have another box that included its own local **carbon** and **hydrogen** boxes. One can access the parts of complex data boxes in a number of ways. Figure 3b shows two of the most useful ways: using the **ask** command, and using a command name **molecule.hydrogen.** Figure 4 shows how to change the information in the two sub-boxes under program control. (Of course, the information could also be changed by directly editing the boxes.)

Structure in Boxer can be constructed under program control using a command called **build. Build** takes a template that contains exactly what you want to see, with a special symbol, @, in front of pieces that need to be filled in from computed values. **Build** preserves the spatial layout that is shown in its input template. Figure 5 shows an example of information retrieval that makes use of an existing database. This basic template can easily be modified to any level of complexity. Figure 6 shows one such extension that students used during their third week of Boxer programming.

Databases can be modified to include graphics as well as text. Figure 7 shows the inclusion of graphic information. Boxer uses sprite graphics, an extension of




```
change hydrogen 2 * carbon + 2
```

Figure 2. Changing the value of a variable.

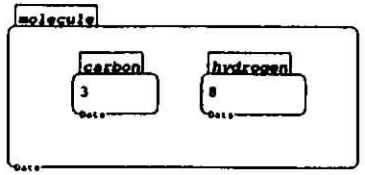


Figure 3.a. A simple database in Boxer.

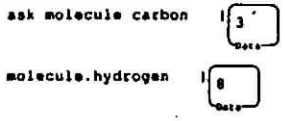
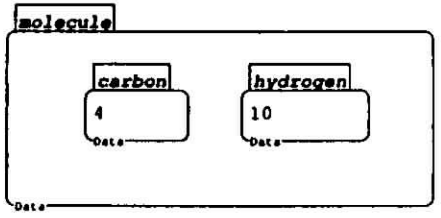


Figure 3.b. Accessing the information in the database.




```
tell molecule change carbon carbon + 1
tell molecule change hydrogen hydrogen + 2
```

Figure 4. Changing information in a database.

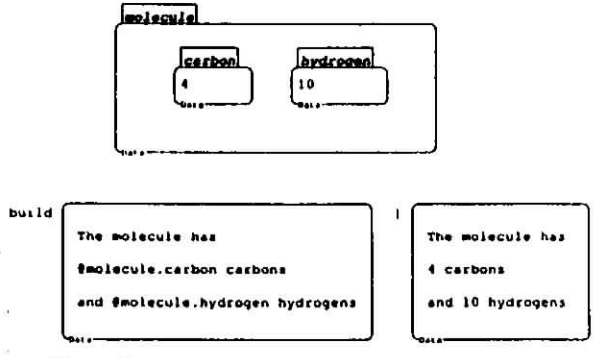


Figure 5. Using **build** to construct a simple description.

Logo's turtle graphics. Sprites occur in *graphics boxes* (a special form of data box) and respond to commands that are familiar from Logo, such as **forward, back, left, right, penup,** and so on. Note that graphics can be returned as the value of a procedure, like any other data.

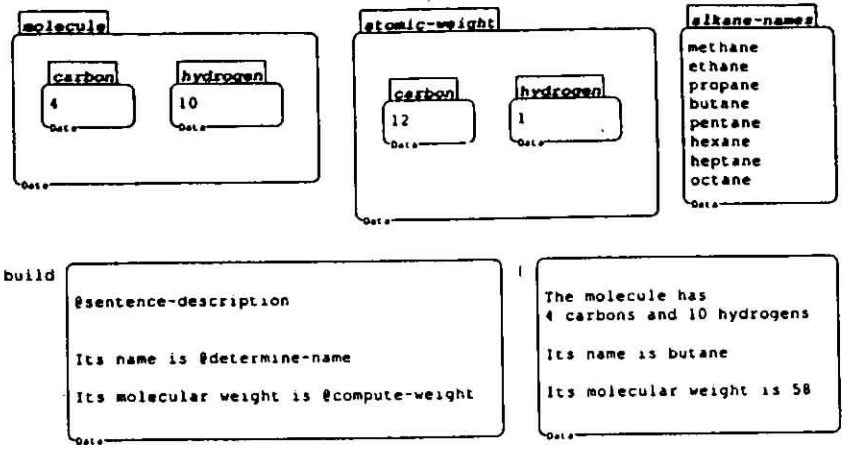


Figure 6. Using **build** to construct a more detailed description.

Observe in Figure 7 that the boxes **atomic-weight** and **alkane-names** have been shrunk, as indicated by grayed boxes. Boxes automatically expand to hold as much as desired, and they can be shrunk or reexpanded with a click of the mouse. Expanding a box that is already open causes it to fill the full screen. Thus, expanding and shrinking allow one to peruse complex data easily, suppressing detail or "zooming" in to it.

Figure 8 displays a more sophisticated approach to manipulating names, with a simple program that automatically generates the names of molecules. For

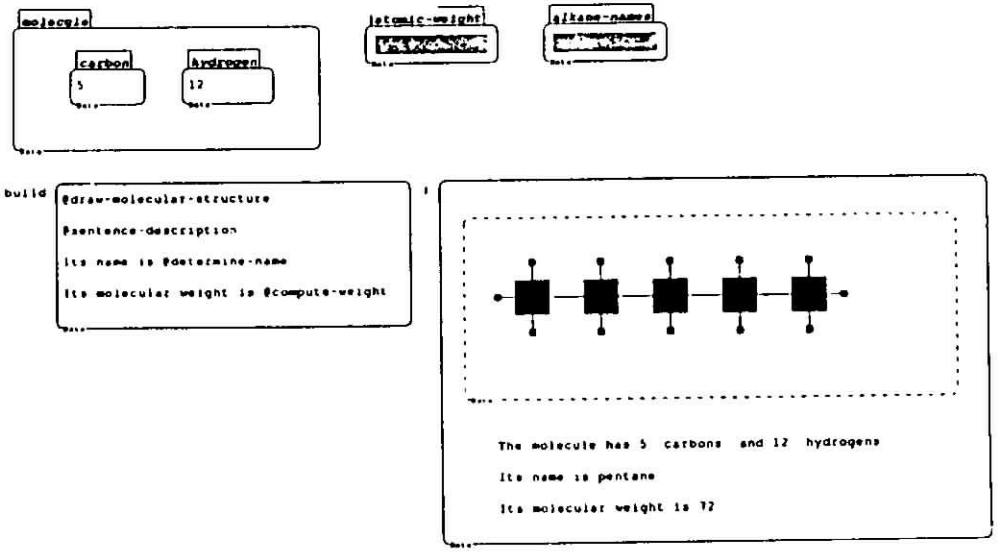


Figure 7. Including graphics in the constructed description.

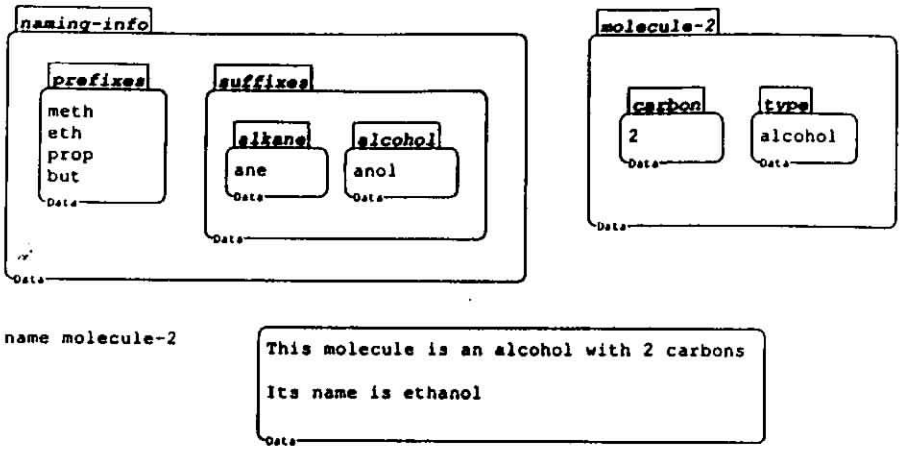


Figure 8. A naming procedure that handles different classes of molecules.

molecules in the classes considered, two pieces of information are used. These are stored in the data boxes **carbon** and **type.** The value of **carbon** determines the prefix, the value of **type** determines the suffix. In **molecule-2,** the value of **carbon** is 2, and this selects the second item in prefixes, "eth." The value of **type** is "alcohol," and this leads to the selection of the suffix, "anol." When the suffix and prefix are combined, the result is the complete name, "ethanol."

Representing Processes. Boxer programs are called *doit boxes.* Figure 9 shows some boxes that implement a Logo-style turtle moving according to a simple model of speed and acceleration. This *tick model* was used in the Boxer-based course on motion described in the articles "Inventing Graphing," to appear in the next issue, and "Learning by Cheating," which appears in this issue. The **tick** procedure specifies how speed and position change with the passage of a single unit of time.

Doit boxes provide the same structuring and interaction capabilities for procedures that data boxes provide for data. Any program or part of a program may be selected and executed. Thus, a student can step through the lines in **tick** by selecting them one at a time, watching the turtle move, and watching the **speed** variable increase. Hierarchical box structure can be used with programs as well

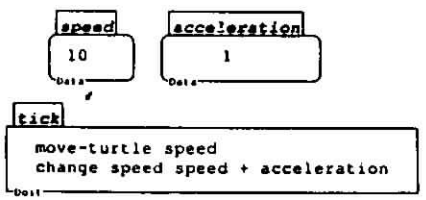


Figure 9. The tick model. The procedure **tick** implements one unit of change in time.

as data. Figure 10 shows a modified **tick** program that contains a subprocedure **new-acceleration** that computes a new acceleration from an external input such as a force sensor. Local data in the form of variables may, of course, be included where appropriate in any procedure or subprocedure.

Object-Oriented Programming. Many kinds of systems are conveniently structured using a programming methodology called *object-oriented programming* in which procedures and data are organized by grouping them into computational structures called *objects.* Combining data and doit boxes, and the locality of data and programs that arise from nested boxes, provides an easy way to implement such objects. Figure 11 shows the implementation of an object named Henrietta that can move around on the display screen, maintaining her own speed and acceleration, when told to execute the procedure **tick.** (In the vocabulary of object-oriented programming, **tick** would be called a *local method* for the object.)

Interactive Data Objects. The appearance and interactive properties of Boxer objects on the screen may be modified to suit new needs. The program in Figure 12 represents vectors as boxes containing arrows. Vectors are implemented using graphics boxes, which can be made "touch sensitive" (they can respond in various ways, and can be altered, using the mouse). In this implementation, vector boxes can be created by students with a keystroke, and like any ordinary box, they can be named and used in programs. In addition, students can modify a vector by dragging the tip around with the mouse. This implementation was used by children to learn about vector descriptions of motion, as described in the articles "Inventing Graphing" and "Learning by Cheating."

Overview of the Articles

The articles in this issue represent current research of the Berkeley Boxer Group. Like Logo, Boxer is more than a piece of technology. It is a developing notion

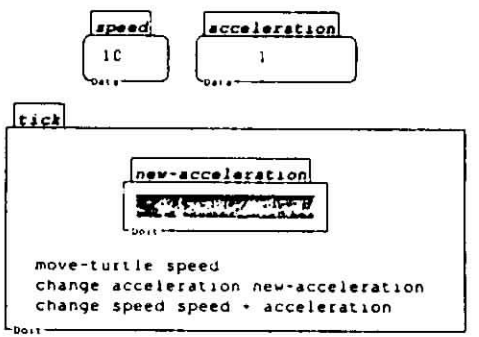
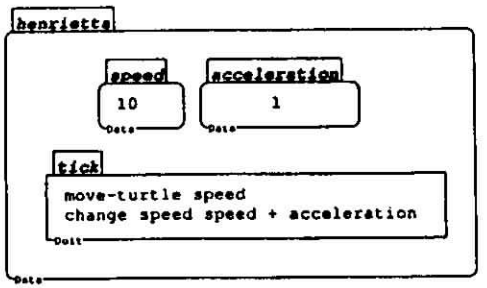


Figure 10. The tick model extended so that a **new-acceleration** is provided by an external sensor.




```
tell henrietta tick
```

Figure 11. An object, Henrietta, responds to tick messages.

for how to foster learning. The Berkeley group's orientation is toward learning and educational experimentation. They regard their most important goal as one of developing models of new styles of learning that will establish the educational value of a computational medium.

"Inventing Graphing," by diSessa, Hammer, Sherin, and Kolpakowski, and "Learning by Cheating," by Adams and diSessa, are both part of a larger project on teaching elementary school children about physics. The Berkeley group calls this project "A Child's Science of Motion." This name emphasizes two fundamentally constructivist orientations that pervade all Boxer efforts. First, there is a commitment to understand the conceptual structure of children's ideas as a basis for developing scientific comprehension. In contrast to an orientation that seeks to identify and correct misconceptions, the Berkeley group believes that it is quite possible to find areas of children's expertise on which one can build from strength, rather than by propping up weaknesses. They have already uncovered a

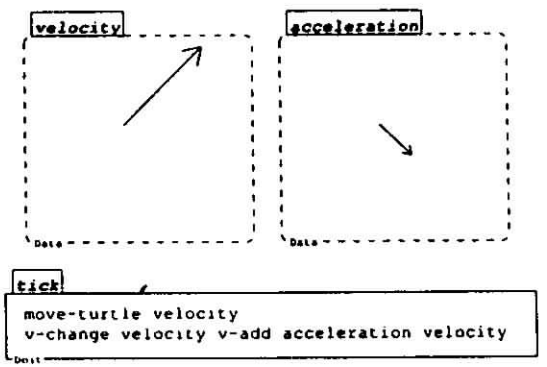


Figure 12. Vectors are implemented in graphics boxes that respond to mouse clicks by repositioning the vector.

few areas in which children are much more expert than they are usually given credit for. "Inventing Graphing" documents one of those areas by chronicling how 8 sixth-grade children engaged in an intense and deeply collaborative joint design of optimal representational forms for expressing motion. This was a remarkable display of children's meta-representational expertise, that is, expertise in the capability to invent, critique, and refine representational forms. [Editor's Note: Because of space limitations, "Inventing Graphing" will appear in the next issue of the *Journal*.]

The second constructivist orientation is in continuity of activities. Just as one seeks continuity in conceptual structure, one can try to understand the forms in which children can effectively engage in self-motivated activities. The point is to find natural paths of development toward scientific activities. "Inventing Graphing" is just as much a story about surprising child interest and competences in action as it is about competences in conception.

"Inventing Graphing" describes a group design of a paper-and-pencil representation. Although the study has everything to do with the Boxer group's educational goals and strategies, it has little to do with Boxer per se. On the other hand, "Learning by Cheating" highlights a Boxer microworld (NUMBER-SPEED) that aims to teach children about motion by building numerical process models of velocity and acceleration. Sixth-grade children don't know algebra, the basic analytic underpinning of most approaches to teaching the concepts in the child's science of motion. But, instead of algebra, one can use programming in various forms to carry the burden of a technically precise and useful representation. In NUMBER-SPEED, functions are represented as lists, and operations such as changing speed or acceleration are represented as list-processing operations. "Learning by Cheating" concerns children subverting the rules of the microworld: because the rules are built in Boxer, the children can easily reprogram them. But the story has a very happy ending in that the children, supported by a learning-oriented classroom culture, use their "cheating" as a stepping stone to deeper understanding.

In "Learning about the Genetic Code via Programming," Don Ploger describes how a centrally important area in biology can be studied as an abstract representational system. It is not surprising that programming can be an apt representation for many ideas in mathematics and physics. But it is not obvious that computational representations can carry much weight in other sciences, such as biology, where static structure plays a much greater role than in physics. Knowledge of the structure of a cell (much less of an organism composed of trillions of cells organized into complex, interrelating subsystems) cannot be reduced to elegantly simple and "mathematizable" abstractions like force or mass. Instead, biology educators need a more diverse representational medium. Boxer is a good candidate for this task because it provides the capacity for easy construction and visualization of complex static structures. It is essential that these Boxer representations are all simultaneously computationally accessible, so

that the processes creating and modifying biological structures can also be easily represented.

Ploger shows how a 16-year-old learned about the genetic code in Boxer by writing programs to manipulate symbols. The processes central to the flow of genetic information involve changes in the structure of important molecules. In Boxer, it is possible to make data objects that represent those biological structures and to write procedures that change those structures. A major theme in "Learning the Genetic Code" is that students can invent their own representational forms, which is also a theme of "Inventing Graphing."

In "Learning about Sampling in Boxer," Henri Picciotto and Don Ploger describe the development of a Boxer-based introduction to probability and statistics, taught to a group of academically talented 12–16-year-olds during the summer of 1988. Picciotto, an experienced teacher and curriculum developer, has a long-time commitment to teaching by providing children with flexible computational tools. Boxer proved to be particularly well-adapted to Picciotto's approach. One of the more important outcomes of the course is that students used and modified the tools provided by instructors as well as fragments of other students' programs. This study demonstrates that a classroom can be a synergistic community of tool builders and sharers. This is a familiar image in science, but it is a new and attractive possibility in classrooms.

Michael Leonard's "Learning the Structure of Recursive Programs in Boxer" focuses on an enduring scientific goal of the Boxer and Logo communities: to use the technological context as a window into basic learning processes. Leonard's work addresses the comprehensibility of programming systems and the ways in which children learn to program. This is part of a project called "Cognitive Benchmarks" (diSessa, in press), whose aim is to develop an elaborated view of what constitutes programming expertise, and thus to provide a better basis for judgement about how well the Boxer group is doing in designing comprehensible programming languages and in teaching programming. Leonard tests some early presumptions about the forms in which programming knowledge appears. His especially novel contribution is an empirical technique and a view of the acquisition of programming skill in terms of impasse resolution, a view that places a child's goals at the center stage in learning.

REFERENCES

Abelson, Hal, & diSessa, Andrea A. (1981). *Turtle geometry: The computer as a medium for exploring mathematics*. Cambridge, MA: MIT Press.

Davis, Robert B., Maher, C.A., & Noddings, N. (1990). *Constructivist views on the teaching and learning of mathematics*. Reston, VA: National Council of Teachers of Mathematics.

diSessa, Andrea A. (1985). A principled design for an integrated computational environment. *Human–Computer Interaction, 1*, 1–47.

diSessa, Andrea A., & Abelson, Hal (1986). Boxer: A reconstructible computational medium. *Communications of the ACM, 29*, 859–868.

diSessa, Andrea A. (1990). Social niches for future software. In M. Gardner, J. Greeno, F. Reif, A. Schoenfeld, A. diSessa, & E. Stage (Eds.), *Toward a scientific practice of science education*. Hillsdale, NJ: Erlbaum.

diSessa, Andrea A. (in press). Local sciences: Viewing the design of human–computer systems as cognitive science. In J.M. Carroll (Ed.), *Designing interaction: Psychology at the human–computer interface*. New York: Cambridge University Press.

Hutchins, E.L., Hollan, J.D., & Norman, D.A. (1986). Direct manipulation interfaces. In D.A. Norman & S.W. Draper (Eds.), *User-centered system design: New perspectives on human–computer interaction*. Hillsdale, NJ: Erlbaum.

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

Watt, M., & Watt, D. (1986). *Teaching with Logo: Building blocks for learning*. Menlo Park, CA: Addison-Wesley.