20

Three Interactions between AI and Education

Kenneth Kahn

MIT AI Laboratory
Cambridge, Massachusetts

The understanding of intelligence that has arisen from AI research can be applied to education to significantly enhance learning. The LOGO project, described herein, is based on this premise. Extensions of the LOGO experience are proposed that are based on AI in interesting ways. One extension is to encourage children to write simple AI programs. Tools to aid the child in this endeavor are discussed. Another supplement to LOGO is the interaction of the child with AI programs specially designed for this purpose. These programs, in addition to being easily used by children, should be modifiable by children in interesting ways. The specifications of such a program are described.

INTRODUCTION

This paper describes some ideas concerning three different ways in which AI can play a role in education, which are illustrated in Figure 1. Children can both be encouraged to write simple AI programs and to interact with specially designed AI programs. Both of these interactions with AI are guided by the understanding of intelligence and learning that has arisen from AI research. These ideas are within the framework of the LOGO project at MIT. LOGO is concerned with the more general problem of enhancing education by applying the concepts that have come from AI research. A description of how LOGO attempts to do this is described in the next two sections. Following that are some of the author's ideas as to how AI can enhance education in ways that supplement LOGO's.

THE LOGO PHILOSOPHY

One of the main occupations of a child is to learn. Yet, typically no effort is made in schools to encourage the child to learn how to learn, or even what this means. The explicit teaching of general principles of learning, at least without many examples, does not accomplish this task. The theory, to the child, is abstract and difficult to understand—even more difficult to use.

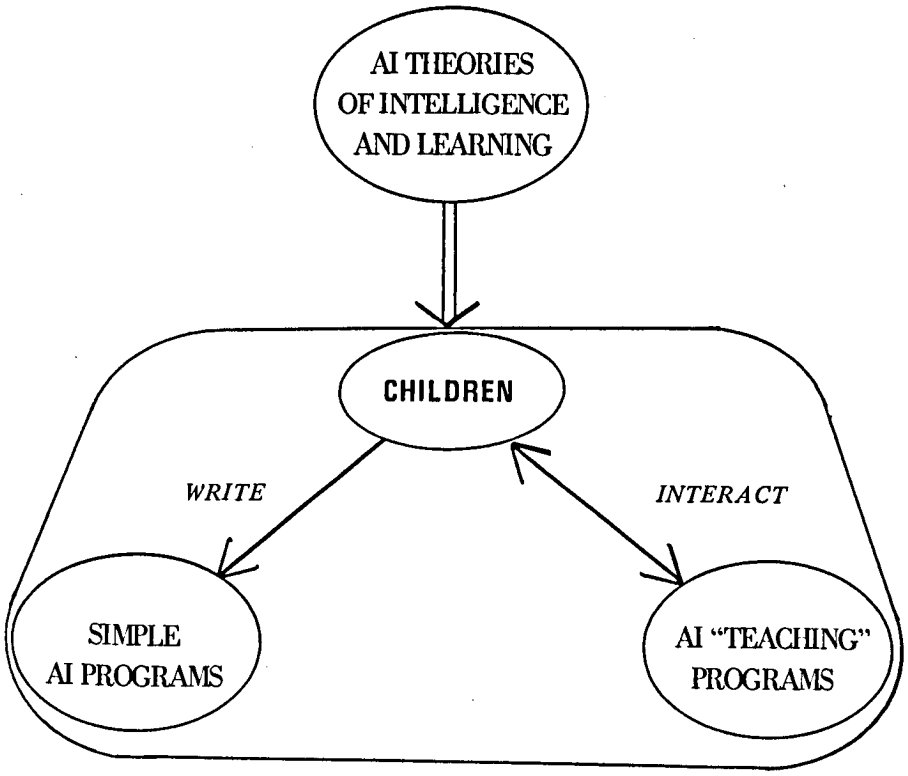

FIG. 1. The three roles of AI in education

The child should, instead, acquire some good "powerful ideas"† for learning and problem solving. The discovery of "powerful ideas" is central to AI research and some have been developed. For instance, the notion of "linearization," or the strategy of attacking the sub-problems of a problem independently, and then dealing with the interactions is a powerful idea. Another one is to view "bugs" in a solution as things to correct and learn from, *not* mistakes that lower a test score. These ideas are typified in Sussman's HACKER (Sussman, 1975). Some other powerful ideas are "naming of concepts", "planning", and "being explicit about one's one thinking". Most important of all the powerful ideas is the notion of a powerful idea itself.

If children could really learn these powerful ideas they would presumably be better students and better problem solvers. But how are they to learn these ideas? What is needed is an environment where these notions surface frequently, where one needs to be explicit about problem solving, an environment where the problems are often found to be intrinsically interesting, an environment where

†As they are called by Seymour Papert

the children constantly use powerful ideas to accomplish their goals. Programming is one such environment. "Linearizing" becomes "implementing subroutines independent of other subroutines". "Naming of concepts" becomes "naming subprocedures and variables". Debugging is an important component of programming.

There are other beneficial effects of choosing programming as an environment which if properly created can help the child become a better student. These are:

1) The experience of working on a long-term project, solving sub-problems and planning as one goes
2) The experience of having explicit control over one's environment
3) The experience of learning about the domain to which the child's programs apply
4) The experience of finding that mathematics can be useful.

Equally important is that children usually find it fun.

The reader should note, however, that there is no claim made here the programmers are any smarter, or any better learners, than other people. It is not the act of programming that encourages one to pick up powerful ideas, rather it is programming well chosen problems, within a language that encourages powerful concepts (such as recursion), and with proper guidance by a teacher.

Computers provide a rich environment for the child to develop concepts of problem solving and learning. Other environments, however, are also useful towards this end. The learning of physical skills, such as circus arts, constitute such an environment. The main thesis is that the *right* verbal description of a physical skill does aid greatly in learning it. Also, interaction of learning in this domain and programming is rich. The child can learn that the powerful ideas learned while programming are useful for more everyday activities.

THE LOGO LANGUAGE

The LOGO language was developed to provide the kind of environment in which children can learn to learn. It is a programming language that is human engineered (or more accurately "child engineered"). The structure of the language, the primitives, and their names were designed to aid in the kind of conceptual thinking described above.

An important part of LOGO is a form of computational geometry called "Turtle Geometry". A turtle is a computational entity with a state consisting of a position and a heading. A turtle accepts commands, telling it move forward or backward, that change its position. It also accepts commands to turn left and right, thereby changing its heading. Turtles are usually realized as physical devices called "floor turtles" or as "cursors" on a display. Turtles also have a pen that can leave a trail as they move.

As an example, let us consider a turtle procedure for drawing a polygon. The procedure expects two inputs, the size of each side and the amount the turtle

should turn after drawing each side. More details are shown in Figure 2.

```
TO POLY :SIZE :ANGLE
10 FORWARD :SIZE
20 RIGHT :ANGLE
30 POLY :SIZE :ANGLE
END
```

FIG. 2a. Turtle procedure for drawing a polygon

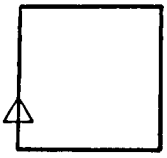

FIG. 2b. Picture produced by the call POLY 100 90

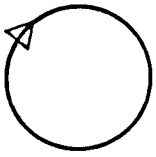

FIG. 2c. Picture produced by the call POLY 1 1

FIG. 2. An example of Turtle Geometry

Children can prove Turtle Geometry theorems that are useful, not only for proving more theorems, but for writing programs. A theorem called the "Total Turtle Trip Theorem" is an example. It states that if a turtle takes a trip and ends up in the same state (position and heading) in which began, then the total amount turned right (minus that turned left) is a multiple of 360 degrees. Children usually understand the intuition of the proof when asked to think about a broken turtle that could not move, only turn. This theorem is very useful in writing state independent subprocedures.

The previous sections describe the LOGO Laboratory as background for what follows. The rest of this paper deals with ideas and research of the author that is consistent with the LOGO framework, but relies more heavily on AI research.

AI FOR CHILDREN

There are several good reasons why children should write and interact with AI programs, a few of which follow:

1) Children are encouraged to think explicitly about how they solve problems. Hopefully the children will thereby improve their ability to

describe and understand their own thoughts.
2) The problem domain to which the AI programs are applied is learned, and in a new and perhaps better way
3) If children are to program, then AI can be an interesting open ended problem domain for that programming
4) The children will learn about AI which is a subject, in the opinion of the author, that is as important as spelling or history.

As an example, consider a child writing a simple natural language understanding system. The child will hopefully learn much about computational linguistics and something about how he or she talks and listens.

To facilitate AI programming of, for example, natural language understanding, or common sense systems, the proper primitives (from the child's point of view) need to be provided. For this purpose the author has developed a set of programs LOGO called "LAIL" for LOGO AI Language. LAIL contains a set of operators useful for writing AI programs. Currently LAIL consists of the following tools:

1) A powerful pattern matcher for natural language understanding
2) A context-free generator for sentence generation to which the child provides rules.
3) A relational data base handler facilitates memory and inference
4) An actor-like animation system which is described later.

Most important of all, these tools should be simple, powerful, natural, and encourage the right kind of conceptual thinking.

AN ACTOR-LIKE ANIMATION SYSTEM

Animation is a programming domain for children in which I am particularly interested. My dissatisfaction with the present animation primitives in LOGO led to the design and implementation of an animation system written in LOGO. Each object in the system is very similar to an "actor" in Carl Hewitt's formalism (Hewitt, 1973). An actor is a computational entity that can receive and transmit messages. Each object in my system can accept turtle-like commands (e.g. forward), remember items told to it, and be taught how to handle new kinds of messages. Each object has a set of patterns which are matched against the incoming message. If a pattern matches then the action associated with that pattern is invoked. Parallel movements of objects on the display screen are handled by an actor called a "scheduler". When the scheduler is sent a message asking it to run, it passes those messages it has associated with the time, redisplays the screen, increments the time, and loops. The scheduler can also produce actors called "movies" that can show a cartoon at any reasonable speed. An example of the use of this system is shown in Figure 3.

The view of programming as collections of actors, or a community of "little people", that send and receive messages from each other is very powerful. It is conducive to a modular, simple, natural representation of the knowledge needed

SQUARE [MAKE GEORGE]

This sends the message "make George" to SQUARE, which creates a square named George

GEORGE [REMEMBER SPEED 25]

This tells George his speed of movement

GEORGE [IF RECEIVE ZIG ZAG ?N THEN ZIG.ZAG "GEORGE :?N]

George is told that if he receives the message "zig zag" followed by some amount, then he is to call the appropriate procedure

CIRCLE [MAKE SALLY]

A circle named Sally is created

SALLY [DO FORWARD 100 AT FRAME 2]

Sally is told to go forward 100 at movie frame 2. This problem is passed on to the scheduler

GEORGE [DO ZIG ZAG 75 AT FRAME 2]

George is told to zig zag at frame 2

SCHEDULER [GO]

The scheduler is told to go. After 2 frames the appropriate messages are sent to Sally and to George, producing the following cartoon:

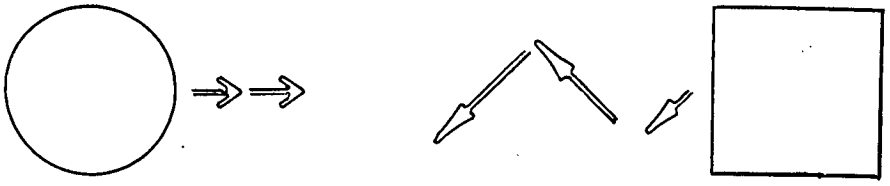

FIG. 3. An ACTOR-like animation system, an example of its use

for the application. The model for intelligence can be a community as easily as it can be an individual with an actor system.

Another AI aspect of this system is the explict "kind-of" hierarchy of actors. Each object is told what class it is a member of when it is created. When any object received a message it cannot handle it passes the problem on to the class of which it is a member. An example of such a tree is shown in Figure 4. The important concepts of instantiation, class membership, placement of knowledge

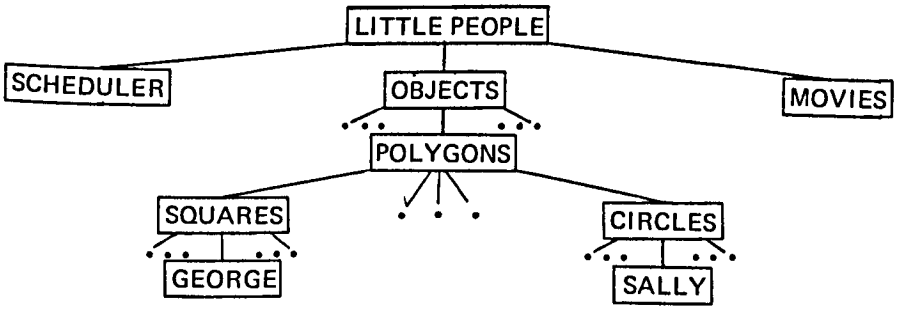

FIG. 4. Actor hierarchy

at the best level of generality, inheritance of properties, and exceptions hopefully flow from the proper use of this aspect of the system.

AN INTELLIGENT ANIMATION SYSTEM

Another environment for learning powerful ideas is one in which the child interacts with specially designed AI programs. I am in the process of designing an intelligent animation system as a test of this hypothesis. In the normal mode of interaction the child will tell a story concerning the "Peanuts" characters, and the system produces an animated cartoon based on the story. The child can then describe the modifications he or she desires. The system produces a new cartoon and the process iterates until the child is satisfied (or quits). The usual concepts of linearization, debugging, and the like prevail here without the need to learn a computer language. Hopefully the child will learn about animation, film making, and story telling by writing stories and scripts to produce the desired cartoons. An hypothetical example of a conversation with this system is shown in Figure 5.

User: Charlie Brown is walking and meets Lucy. He says, "Good morning". She says, "What's so good about it?...". Charlie Brown frowns and says, "Good Grief!".
The system shows the cartoon after having made many simple inferences and default choices. For example, it decides where to put Lucy and Charlie Brown, how fast they walk, how they are oriented, their facial expressions, and so on.
System: How was that?
User: OK, but Lucy should look crabby when she says, "What's so good about...".
The system changes the necessary parts of the cartoon and shows the new one.
User: That's fine

FIG. 5. A hypothetical discussion with the intelligent animation system.

Another more important way in which a child could interact with this planned system is by understanding and modifying the knowledge of the system. The system's knowledge about animation techniques, the real world, common sense facts about people and things, and the "Peanuts" characters will be as modular and simply represented as possible. The child, with the help of the system, will be able to define new characters for the stories, change the system's vocabulary, add a reasonable default course of action and so on.

One hope is that in understanding and modifying the knowledge of an intelligent system the children will actually see the knowledge of something actually accomplish something complex. The idea that knowledge and intelligence can be something formal should help the children to be more explicit about their own thoughts and knowledge. The use of the system will also, hopefully, help provide a vocabulary for talking and thinking about one's thoughts. If the design and implementation of this intelligent animation system is done well, it will provide a rich and exciting supplement to LOGO's traditional environment.

SUMMARY

In this paper three ways in which AI can interact with education have been described. After an introduction to LOGO thinking and language, the benefits of children writing simple AI programs using the proper tools were described. Finally, the ways in which an AI system designed for education can interact with children were discussed. These ideas should be implemented and tested with children. Only then will the effects on education be known.

REFERENCES

Abelson, *et al.*, (1974) Logo Manual, August 1974, LOGO memo No. 7.

Goldstein, I.P. (1974) *Understanding Simple Picture Programs*, MIT AI Laboratory AI-TR-294.

Hewitt, C., *et al.*, (1973) A universal modular ACTOR formalism Proc. *IJCAI-73*, Stanford, California.

Papert, S. (1971) Teaching children thinking, MIT-AI Laboratory Memo No. 247.

Papert, S. (1975) Uses of technology to enhance education, MIT-AI Laboratory Memo No. 298.

Sussman, G.J. (1975) *A Computer Model of Skill Acquisition*, American Elsevier Publishing Company, Amsterdam.