

Using LEGO Robots to Explore Dynamics

by

Mario Octave Bourgoïn

B.Sc., University of Ottawa
(1978)

Submitted to the Media Arts and Sciences Section
School of Architecture and Planning
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

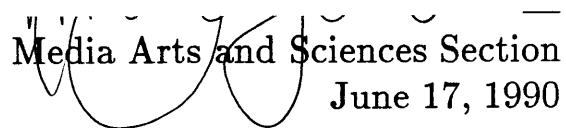
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

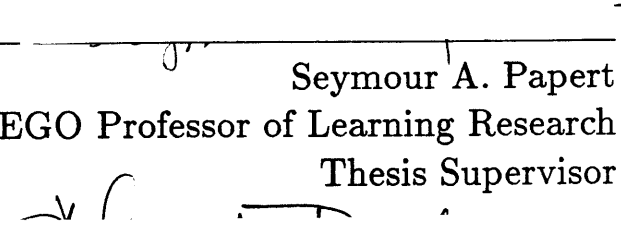
September 1990

© Massachusetts Institute of Technology 1990
All Rights Reserved

Signature of Author _____


Media Arts and Sciences Section
June 17, 1990

Certified by _____


Seymour A. Papert
LEGO Professor of Learning Research
Thesis Supervisor

Accepted by _____


Stephen A. Benton
Chairman, Departmental Committee on Graduate Students

OCT 04 1990

LIBRARIES

Using LEGO Robots to Explore Dynamics

by

Mario Octave Bourgoïn

Submitted to the Media Arts and Sciences Section
School of Architecture and Planning
on June 17, 1990, in partial fulfillment of the
requirements for the Degree of
Master of Science

Abstract

A system for programming LEGO robots was created for children's use in learning how patterns can emerge in the activities of systems of interacting elements. A group of nine 6th and 7th grade children took part in a workshop involving programming the robots to locate, capture, and then release a source of light. The children were observed in their process of learning how to make patterns emerge in the robot's activity. The children's notes and videotapes of them both at work and reflecting on their work were used to analyze how they accomplished it.

Thesis Supervisor: Seymour A. Papert

Title: LEGO Professor of Learning Research

Contents

1	Introduction	9
2	Dynamics and LEGO Robots	12
2.1	Emergence and Dynamics	13
2.2	Using LEGO Robots	20
2.3	Related Work	22
2.4	Preliminary Study	22
3	The Materials	26
3.1	The Robots' Bodies	26
3.1.1	The Workshop	26
3.1.2	The Competition	29
3.2	The Robots' Minds	33
3.2.1	The Editor	33
3.2.2	The ACT Language	33
3.2.3	Debugging Programs	36
3.2.4	The Initial Program	38
3.3	Related Work	39
3.3.1	The Subsumption Architecture	39
3.3.2	Cybernetics and LEGO Robots	40
3.3.3	Robot Odyssey	41

3.3.4	Agar	42
3.3.5	MACNET	44
4	The Workshop	46
4.1	Overview	46
4.2	The Participants	47
4.3	Workshop Time Line	48
4.4	The Sessions	49
5	Observations and Analysis	53
5.1	Research Questions	53
5.2	The Children	54
5.3	Learning About the Workshop	55
5.4	Learning About the Tools	71
5.5	Learning About Emergence	73
6	Conclusions and Future Work	82
6.1	Summary of the Thesis	82
6.2	Children's Thinking About Emergence	85
6.3	Improving the Materials	88
6.4	Future Explorations	92
A	Resources Used in this Study	96
B	Example Programs	97
B.1	Sample Light Seeking Programs	97
B.2	MOVEAROUND	101
B.3	The SMART Robot's Program	102
C	ACT Language Reference	104
C.1	Basic Language Structure	104

C.2	Primitive Action Statements	104
C.3	Primitive Actions	105
C.4	Primitive Operations	105
C.5	Special Words	106
D	The ACT Keyboard Hotkeys	107
D.1	Editor Command Keys	107
D.2	Compiler and Debugger Keys	109
E	The ACT Compiler	110
E.1	The Parser	110
E.2	Intermediate Representation	111
E.3	Run-Time Environment	111
E.4	Code Generation	112

Acknowledgments

I owe everything that is good in this research to people who cared; I will never be able to thank all of them enough. Amongst the hundreds of people who helped me in this work, I must thank above all my advisor, Seymour Papert, for his support, his inspiration, and his confidence in me. Philip Agre gave me central idea for this research and Fred Martin, Allan Toft, Steve Ocko, and Mitch Resnick gave me the materials I needed to carry it out. Andy diSessa and Idit Harel were my flawless readers, they helped me see what was important and kept me focussed. Steve Benton, Linda Peterson, and Ilze Levis were this project's midwives. Susan Cohen of the Brookline Schools helped me find a place and people with whom to work, and Hillel Weintraub helped me find Susan Cohen. Edith Ackermann provided academic support that was essential to the development of my research. The masters students of the Media Laboratory, particularly those of the Thesis Prep class of fall 1989, helped me with their support and company. I owe equally great thanks to each of the Epistemology and Learning Group's members, because they fostered my intellectual growth over the last five years. My office mates, Carol Strohecker and Aaron Falbel, gave me their spirit to get it done but with care. Kevin McGee showed me the way with his uncompromising will to do it right. Fred Martin, P.K. Oberoi, and Randy Sargeant led this year's MIT Robot Design Competition that provided such special excitement to the children in the workshop. I must also thank the Hennigan School teachers and those of the Science and Whole Learning schools for sharing with me their thoughts and ideas and for showing me the teacher's side of life. The Media Lab's researchers and support staff were responsible for the exciting and inspiring life that led to this work. I owe special thanks to Nicholas Negroponte and Jerome Wiesner for making this work possible by creating the Lab. I owe equal thanks to Michael Travers and Alan Ruttenberg for caring enough to champion the cause of academic freedom for us all. The pictures in this thesis would not have happened without the generous help of Brian Anderson, Trevor Darrell, Elaine McCarthy, and Laura Teodosio. Larry Ward and Ray Hinds helped me assemble the LEGO and electronic materials. I must thank my

parents, my sister, and her family for their unfailing and prodigal love, care, and help. I owe very special thanks to Susan Scott who made me love life enough to live it this hard. All of the help I've gotten is enough to make anyone's life fantastic, but what made it glorious were the children with whom I've worked. I want to thank them for sharing their ideas, thoughts, and feelings with me, but above all for the time that brought us all together.

The research reported here was conducted at the MIT Media Laboratory and was supported in part by the National Science Foundation, the McArthur Foundation, and InterLEGO/SA.

To all the children of the world.
The young, the old,
The great and especially the small.
I love you.

Chapter 1

Introduction

To those accustomed to the precise, structured methods of conventional system development, exploratory development techniques may seem messy, inelegant, and unsatisfying. But it's a question of congruence: precision and flexibility may be just as dysfunctional in novel, uncertain situations as sloppiness and vacillation are in familiar, well-defined ones. Those who admire the massive, rigid bone structures of dinosaurs should remember that jellyfish still enjoy their very secure ecological niche.

Beau Sheil, "Power Tools for Programmers"

A system for programming LEGO robots was created for children's use in learning how patterns can emerge in the activities of systems of interacting elements. A group of nine 6th and 7th grade children took part in a workshop involving programming the robots to locate, capture, and then release a source of light. The children were observed in their process of learning how to make patterns emerge in the robot's activity. The children's notes and videotapes of them both at work and reflecting on their work were used to analyze how they accomplished it.

This thesis presents a micro-genetic study of the process that the group went through to use the emergent properties of a robot's interactions with its environment to program it for a simple task. This study was done in the context of an eight week after-school

workshop in the programming of robots made from LEGO materials. The workshop's purpose was to engage a group of children into exploring how regularities can be made to emerge in the activity of the robots. In doing this kind of work, the children looked at phenomena that was emergent from the particulars of the situations they were creating. The inspiration for this work came from Philip Agre's doctoral research [Agre 1988]. Agre and David Chapman had worked on understanding the patterns that arose in the interactions between Pengi, the game player, and Pengo, the game, and had programmed Pengi so that it would take advantage of them. In his doctoral thesis, Agre named those patterns of interaction "dynamics." What results from this kind of work is often called "emergent phenomena."

Chapter 2 will describe some of the ideas that influenced the design of this study. The phenomenon called emergence will be described and the idea of dynamics will be related to the concept of emergent phenomena. Some aspects of the approach to learning developed by the Epistemology and Learning Group MIT's Media Laboratory will be discussed, and different project to develop tools for learning about emergence will be presented. Finally, a preliminary study of adults observing robot activity will be described and some of its results will be discussed.

Chapter 3 will discuss the LEGO and electronic materials that were provided to the children, both for the workshop and for the Robot Design Competition. It will also describe the ACT parallel programming language that was used by the children to program robots and will talk about the facilities that were available for creating programs and debugging them. Five other robot construction systems will be discussed.

Chapter 4 will talk about the structure of the workshop and will give an overview of the events that happened in each of the eight sessions and in the post-interviews.

Chapter 5 will discuss the process through which the children appropriated the tools that were provided and the interactionist ideas for programming robots. Selected transcripts from the sessions will be used to show how the children thought about the materials and ideas as the workshop progressed.

Chapter 6 will reflect on the children's use of dynamics to control the robot. It will also look at how the ACT system and the workshop's goals could be changed to facilitate their appropriation by the children. New directions for exploring emergence that are suggested by this research will be discussed.

Chapter 2

Dynamics and LEGO Robots

Two family of ideas informed the design of this study. The first one concerns the study of patterns that can emerge in the activity of systems of interacting elements, and the second one supports the view that learning can often best be done through building. Emergence is a phenomenon that is receiving a lot of attention these days in such forms as theories of activity, chaotic systems such as those that produce our weather, and self-organizing systems such as colonies of ants. The aim of this attention is to find ways of understanding these phenomena and to develop formal theories to explain them. By looking at how we already work with such phenomena, we can improve our tools for thinking and learning about it. Constructionism claims that learning is best done through a process of self-directed building that involves ideas and facts that we desire to learn. This view of learning motivated the use of a workshop as the context in which to explore thinking about emergence.

The first section of this chapter will look at emergence, the second one at Constructionism, the third one will present a project currently underway that explores emergence in self-organizing systems, and the final section will present an example of adults observing emergent phenomena.

2.1 Emergence and Dynamics

In the workshop, the children worked with emergence that arises from the interactions of an agent in an environment. In this section, the idea of emergence will first be explored in general, and then with respect to the specific kind of emergence that was created by the children.

What is Emergence?

A survey of articles that address the concept of emergence shows that it is difficult to define. Often, people will see some definitions as excluding obviously emergent phenomena while they will see others as being so broad as to include all phenomena. Still, by using a range of examples of systems that exhibit what people call emergence, a view of this concept can begin to be assembled.

- A triangle emerges from the interaction of your brain with a picture of blackened circles with cutout arcs and angles (see Figure 2-1.)
- Zigzag motion patterns emerge from the reactions of a robot to features of its environment (see Figure 2-2.)
- Eyes and other organs emerge from the growth of clumps of undifferentiated cells.
- A colony of ants emerges from the interactions of the ants, both chemical and mechanical; it builds a hill and will act to defend it.
- In cellular automata, “flowers”, “gliders”, and other structures emerge from the interactions of many simple machines.

One striking feature that these examples have in common is that a structure emerges without the intervention of central control, be it a designer, a plan, a circuit, or anything else that directly puts the structure in place. Rather, this structure is the consequence of the interactions of a number of parts. And while the particulars of the parts seem

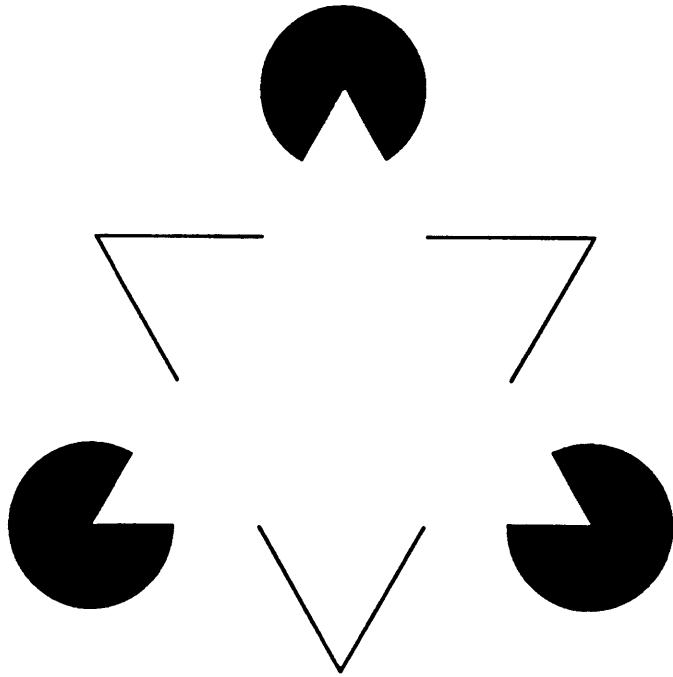


Figure 2-1: Subjective Contour

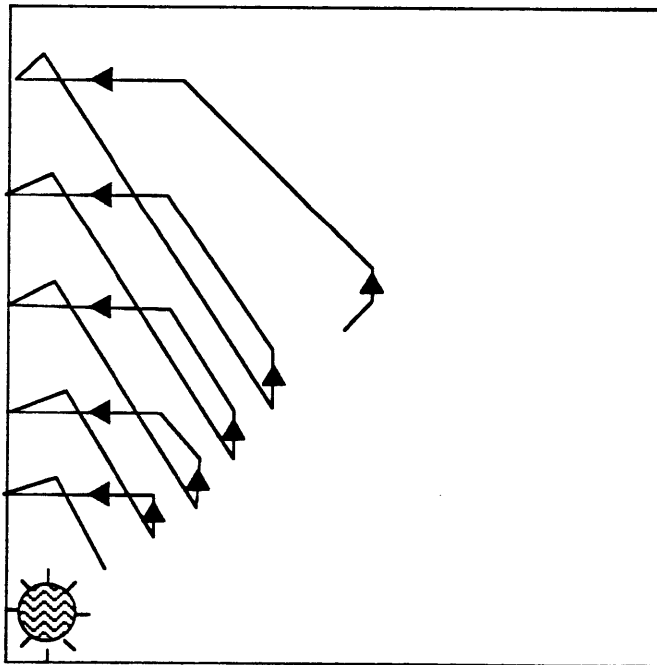


Figure 2-2: Emergent Zig Zag Motion

to have a great influence on the details of the structures that emerge, the character of those structures often seems to remain the same if the parts change. These structures are robust, yet in a way that is hard to define.

Often, people will remark on their surprise upon seeing emergent structures, but not everything that we call emergent is unexpected. For example, Maja Matic designed a robot named Toto and programmed it with only four rules of action so that when it moves it displays, amongst other emergent phenomena, boundary-tracing behavior [Matic, 1990]. This shows that emergent phenomena can be sought out, actually deliberately assembled. Perhaps this surprise we experience comes from the fact that we do not expect anything to happen in such systems because we do not understand how it could happen.

Intelligence as an Emergent Phenomenon

Emergence is an idea that is often cited when talking about intelligence. Part of this comes from our knowledge that intelligence is based on the brain, a huge collection of simple parts none of which alone can do what the brain can do. Sherry Turkle speaks about the idea of the emergence of intelligence in her book *The Second Self* [Turkle 1984]. She points out that while emergence may seem like magic, nature is rich in examples that can serve as a metaphor for it. In talking about emergence, she describes systems where higher order comes from the interactions of parts without there being anything in the systems that directly specifies what the order should be. She applies this idea to programming and contrasts the view that computers can do no more than what they are programmed to do, which she calls the “Lovelace” model, to a view that says that heuristically programmed computers will do what emerges from the interactions of their parts, the “Society” model.

The idea of a computer as a “society” of competing programs is one of several key ideas from the AI community that challenge the image of the computer as following step-by-step instructions in a literal minded way...

[Turkle 1984] pg. 277

According to Turkle, in the Lovelace model results come from systematic rules and ordered instructions, and what these results will be can known in advance. She contrasts the results of the Society model as emerging from the interactions of independent parts, almost through a democratic process. She adds that those results can only be known in advance in a vague way and depend greatly on the program’s environment.

Danny Hillis describes systems made from many simple components but that as a whole are more organized than each of the individual parts [Hillis 1988]. In doing this, he focuses on the different scales at which the components and the emergent phenomenon operate. In talking about emergence he uses the example of ice crystals forming as a consequence of the rules of interaction of water molecules. But he quickly points out that the connection between the rules and the crystals is not obvious. He then considers the possibility that intelligence might be due to an emergent phenomenon.

It would be very convenient if intelligence were an emergent behavior of randomly connected neurons in the same sense that snowflakes and whirlpools are emergent behaviors of water molecules. It might then be possible to build a thinking machine by simply hooking together a sufficiently large network of artificial neurons. The notion of emergence would suggest that such a network, once it reached some critical mass, would spontaneously begin to think.

[Hillis 1988], pg. 175

His view brings out the statistical character of many emergent systems: that if enough elements are brought together, something will emerge from their interaction, although often no one is quite sure what. But Hillis criticises the notion of emergence as offering

“neither guidance on how to construct such a system nor insight into why it would work” ([Hillis 1988], pg. 176). In the end, he returns to the problem of connecting the rules to the patterns that emerge from them. He concludes that:

The emergent behaviors exhibited by these systems are a consequence of the simple underlying rules defined by the program. Although the systems succeed in producing the desired results, their detailed behavior are beyond our ability to analyze and predict.

[Hillis 1988], pg. 188

In “Waking up From the Boolean Dream”, Douglas Hofstadter also looks at the relationships between emergence and intelligence. In talking about ant colonies as a metaphor for emergent symbol systems, he says that “...what keeps [*the team of ants*] coherent is ... the regular patterns that are sure to emerge out of a random substrate when there are enough constituents. Statistics in short” ([Hofstadter 1985] pg. 653). So, like Hillis, Hofstadter hopes that if enough elements are put together, emergent phenomena will result. To get at the issues underlying the emergence of intelligence, Hofstadter contrasts two kinds of models to explain the existence of limits to how many different things people can remember in the short term. One model is the view that ascribes the number of things that can be remembered to people possessing a limited number of “registers” in which to store representations of objects. In this model, short-term memory is explicit and its limits are clear. If some people can remember more than others, it’s because they have more registers; you only need to add registers to a system for it to be able to remember more things. In the other model, the limits of short-term memory are a consequence that emerges out of the design of a thinking system. It is “...a product of many interacting factors, something that was not necessarily known, predictable, or even anticipated to emerge at all” ([Hofstadter 1985] pg. 641). If some people can remember more than others, the system must be somehow different, but it makes much less sense to think that just adding a something to the system will increase

how much it can remember. The lower level is the substrate out of which emerges the activity seen at the higher level; the higher level describes what the system appears to do while the lower level describes what is actually happening. In Hofstadter's example, believing that short-term memory is the product of explicit memory structures in the head is to confuse these two levels of description.

What is a Dynamic?

In his doctoral thesis, Philip Agre looks at emergence from what seems to be a very different perspective from the three just mentioned [Agre 1988]. While Turkle, Hillis, and Hofstadter examined interactions in systems composed of a large number of simple and similar elements, Agre considers the patterns that emerge in the interactions of a few different elements. Yet, as with the other examples, these systems lack a central control structure and the emergent activity described at the higher level is absent from the lower levels.

Agre uses the word "dynamic" to refer to descriptions of the patterns that emerge from the interactions between certain kinds of agents and certain kinds of worlds. In his doctoral thesis, Agre analyzed the dynamics that arose in the interactions of a mostly reactive agent with a video game. A mostly reactive agent is one whose next action is a response to immediate stimuli and not due to some change of internal state such as the advance of a program counter.

Dynamics are a part of talking about the interactions of an agent with the particulars of a world; we say that a robot is situated in a world. In an earlier paper [Bourgoin 1990], an example of a dynamic is shown as giving the means to explain why a person tripped daily at the same place in his morning trip to the subway.

If I want to explain this tripping, I must talk about my interactions with that particular broken part of the sidewalk; after all, I do not trip all the time, and not everyone I have seen pass by the break tripped. In fact, I only tripped at that spot in the morning when there was a lot of traffic in the streets and there was a stream of pedestrians going from the T station to the Government buildings that were behind me. The only way I can explain this tripping is by describing my relationship with my environment at that particular time: my situation.

[Bourgoin 1990] pg. 141

Because dynamics describe something that arises from the combination of an agent and a particular world, they are different from behavioral descriptions that talk about something that mostly arise from the nature of an agent alone.

What is gained in the difference between Agre's systems and the other ones is that their relative simplicity makes the process through which the patterns emerge from the interactions of the parts more accessible to an observer. As a means for studying this kind of emergence, Agre counsels:

It seems like an interest in emergence requires the ability to move back and forth between the robot's perspective and an observer's perspective; emergence, it seems to me, has precisely to do with the difference between those two perspectives, the kinds and degrees of orderliness that the observer can see but the robot itself is not in whatever sense aware of.

Agre, personal communication

As Agre points out, emergent phenomena is understood as a consequence of the interactions of the parts when a comparison is made between different levels of description. An observer can see the zigzag patterns formed by a robot as shown in Figure 2-2, but the robot that makes them only moves in straights and turns in reaction to stimuli.

2.2 Using LEGO Robots

MIT's Epistemology and Learning Group, of which the author is a member, is the center for the *Constructionist* approach to education. We believe that people learn by reconstructing knowledge, and that this process is facilitated when done in the context of building meaningful objects.

Constructionism is a synthesis of the constructivist theory of developmental psychology and the opportunities offered by technology to base education for science and mathematics on activities in which students work towards the construction of an intelligible entity rather than on the acquisition of knowledge and facts without a context in which they can be immediately used and understood.

[Papert 1986], pg. 8

This view of learning is synergistic with the approach to exploring emergence suggested in the previous section by Agre. In keeping with this approach, a set of tools was created so that children could build systems of interacting elements and observe patterns of activity that emerge from them. Some of the elements in this set of tools were actions to be programmed into agents by the children, while others were physical structures in the world in which the agents were situated. The agents were chosen to be pre-built robots made from standard LEGO Technics parts combined with a portable computer called the Programmable Brick. The computer was to be programmed using the ACT programming language. These tools will be described in greater detail in Chapter 3. The world with which the robots interacted included stationary and moving lights, other agents, both robots and children, and a corridor, a pair of rooms, and their contents all located within an elementary school in Brookline.

The Epistemology and Learning Group is also concerned with the children "appropriating" the tools and ideas so that they feel comfortable with them.

Much of our research is based on the working hypothesis that children ... will learn science best if they use it ... right now. ... And we do not mean that they use it only to perform activities imposed on them ... We mean that they use it for purposes they experience as their own.

[Papert 1986] pg. 9

And as part of this appropriation, we want to foster the development of a community of ideas that can serve its members not only as a resource on how to think about the tools and how to use them but as a group with which new ideas for applying the tools can be developed.

For these reasons, an after-school workshop setting was chosen as the most suitable context in which to use the tools to explore emergence. The children who participated in the workshop were not bound to see it through to its end, possibly doing something that was not of their choosing or in line with their interests. In addition, they were presented with the opportunity to do something which had not been done before to the knowledge of the author, a feature which helped foster in them a sense of excitement and of personal accomplishment. In a workshop setting, the children could work at their own pace because they were not under the usual school constraint of having to remain on a particular subject for an entire session or to change to a different one according to a set schedule. Finally, while the children were split into pairs to program the robots, this did not diminish their interactions, in particular since they were encouraged to help one another complete their projects. In fact, this allowed the children to compare a number of different programs for controlling the robot.

Because of limits on available time and equipment, the workshop was not as open-ended as would have been desirable for a Constructionist environment. Some suggestions for a different use of these tools and ideas in future workshops will be given in Chapter 6.

2.3 Related Work

The author knows of only one other research project that combines education and the study of emergence. Mitchel Resnick has proposed to build a programming system that children can use to explore self-organizing systems [Resnick 1989]. Self-organizing systems exhibit the statistical emergence that was described in Section 2.1. Resnick's system will include a programming language called *Logo (pronounced *star-Logo*), a version of Logo that allows the "control the *concurrent* actions of (at least) *hundreds* of "objects" (e.g., ants or molecules or automobiles)" ([Resnick 1989] pg. 6).

Resnick has many goals in developing this tool. He intends *Logo to be an accessible tool with which non-expert programmers will be able to use the Connection Machine, a massively-parallel computer with at least 16,000 processors. He also wants to use it to argue that the study of self-organizing behavior deserves a place in pre-college education. He intends "to trace how students' understanding of self-organization develops and evolves" ([Resnick 1989] pg. 2). Finally, he wants to develop new ways of categorizing, classifying, and thinking about self-organizing behavior.

Chapter 6 will consider the relevance of the results presented in this thesis to the kind of emergence that Resnick intends to study.

2.4 Preliminary Study

While constructing the environment and the agents for use by the children, a preliminary study was done with adults to find out what kind of language they used when describing robot activity they observed. The information presented here was also included in a previous publication of preliminary results [Bourgoin 1990]. This study used a robot that could find a source of light using only a sensor which reports the intensity of the light that reaches it from the direction in which it is pointed. During the experiment, the robot was made to find a light six different times, each time following a procedure chosen at random from a set of four different procedures for finding a source of light. Every

procedure included a method for making the robot back up and turn towards the left if its front bumper detected that it had run into an obstacle. Three of the procedures made use of the light sensor to locate the light, while the fourth moved about at random. Three equal implementations of one of these light-using procedures, each written in a different language, can be seen in Section B.1. The first implementation is in Logo, the second is in ACT, and the third is in the form of a logic circuit. These implementations will allow the reader to compare these different languages. With this procedure, the robot moved in a straight line as long as its sensor reported an intensity which either stayed the same or increased. If the intensity dropped, the robot turned left by an eighth of a circle. However, if the robot ran into an obstacle, it would back up even though it may have been moving towards increasing light beforehand. The Logo version of this procedure was used for the preliminary study.

The robot was set in a large open-top square box, and a light was placed in one corner of the box. The participants were given the following written directions (in this protocol, the robot is referred to as a “turtle”):

The turtle’s task is to reach the light. It will have six tries. You will first observe all six tries. Each time, you will be asked to describe aloud the activity that you see. After the sixth try, you will be given a sheet on which you will be asked to write in what order you rank the tries according to the turtle’s ability at reaching the light at each attempt. I will then ask you some questions related to how you ranked the tries. Before each try, I will check the turtle to see that it’s physically ok. I will place the turtle in the center of the pen, facing away from the light. I will then press a button that tells the turtle to begin that try. Each try will end when you are satisfied that you have seen enough of that attempt. I will then press a button that tells the turtle that it’s the end of that try. During a try, you may ask me to move the turtle back to its starting position, particularly if you think it’s stuck. You can walk around the turtle pen during an attempt to get different views of

the activity in the pen.

After the above procedure was completed, at the end of the six tries, the participants were asked the following questions:

1. In some of the tries, the turtle isn't seeking the light. Do you know which tries?
2. Would you like to know which tries? Now that you know which tries were random, would you like to change your ranking of the tries? How would you change them? Why?
3. In two of the tries, the turtle used a method it had already used in two earlier tries. Do you know which tries?
4. Would you like to know which tries? Now that you know which tries were repeats, would you like to change your ranking of the tries? How would you change them? Why?
5. The turtle is equipped with only two sensors: a switch attached to its front bumper, and a light sensor that reports the intensity of light in roughly a third of a circle in front of the turtle. Can you devise a method that the turtle can use to reach the light? How is your method like those the turtle used during the tries?
6. Would you like to see the methods the turtle used during each of the tries? Now that you've seen the methods, would you like to change the method you described? How would you change it? Why?

It was found that during the observation of the turtle, most subjects first gave local descriptions of the robot's actions such as "it turns away from the light" and "it takes smaller steps when getting closer to the light." One subject also observed how the robot would make large steps when it was far away from the light. After observing the activity for a while, some of the subjects began to give overall descriptions of the actions of the robot such as "it moves in a zigzag pattern towards the light," as was previously shown

in Figure 2-2. The subjects' way of talking about the pattern of motion indicated they believed that it had been built into the robot.

This study was also used as a means of getting the subjects to talk about the kinds of actions they would program into the robots to accomplish the goal. It was found that they extracted simple actions from the overall descriptions and related them to the local information available to the robot. For example, one subject talked about what the robot would "see" as it was moving at right angles to the light and used that information to decide what the robot should do next.

These results showed that some adults have a tendency of moving from local descriptions of action to global descriptions of activity—which are dynamics. However, their development of procedures showed they had some trouble in separating out the elements that came together to make up the activity they described. If they had been able to implement these procedure in a working robot and incrementally improve their design, they might have gotten a better understanding of how to make a robot reach a light.

A version of this study was used to introduce the children to light-seeking robots as detailed in Section 4.4.

Chapter 3

The Materials

The materials used in this workshop combined hardware and software components. The hardware consisted of LEGO robots controlled with pocket-size portable computers. The software consisted of a programming language embedded within a development environment that included an editor, a debugger, and an example program. After these materials are described, alternative materials for constructing and programming robots will be reviewed.

3.1 The Robots' Bodies

The robots were built using materials from LEGO Technics construction kits. The robots used during the workshop were controlled with a 65C02-based computer and their electrical and electronic parts came from LEGO TC logo kits. The robots used during the competition were controlled with a 68H11-based computer and their electrical and electronic parts were provided by the contest organizers.

3.1.1 The Workshop

LEGO TC logo is a set of tools that children can use to explore the design and construction of machines. It combines the LEGO Technics as building materials with the

Logo language for programming the machines. In addition to the beams, axles, gears, and wheels of the standard LEGO Technics kits, the LEGO TC logo kits include motors, switches, threshold light sensors, and an interface box through which a desktop personal computer can control the electrical components.

The Programmable Brick To get robots free of cables that would run to to a fixed location, this workshop substituted the Programmable Brick for the desktop computer. The computer in this LEGO brick is often described as being equivalent in power to an Apple II computer. It is three and a half inches long by two and a quarter inches wide and one and a quarter inches high and it weighs almost three and a half ounces. It must be used with a battery pack which weighs almost nine and a half ounces so that the entire control system comes to under a pound. The Programmable Brick includes a 65C02 microprocessor, 8K of ROM, 32K of RAM, a serial communication port, and an interface board compatible with the LEGO TC logo materials. Using this board, the brick can receive inputs from four sensors numbered from 0 to 3 and can drive four bidirectional effectors lettered from A to D. The sensors used in this workshop were on/off switches and analog light sensors. The effectors used were bidirectional motors.

The Light Sensor The kind of light sensor used for the workshop was based on a light-sensitive resistor, and reported values on a scale of 0 for low or no light to 255 for the maximum light intensity that it could sense. This maximum amount was close to the intensity of light received by the sensor when put against a 60-watt light bulb. The sensor was able to detect ambient light in about a third of a circle (120 deg) in front of it. Readings from this light sensors were unreliable for a number of reasons. First, the Programmable Brick's way of using an analog sensor made readings from them vary wildly when the value reported dropped below 192. A second source of unreliability was variation in the components used to build light sensors. One result of this variability was that different sensors used during the workshop had different light intensity resolution in the top of their scale, some reporting a change from 254 to 255 when moving an inch

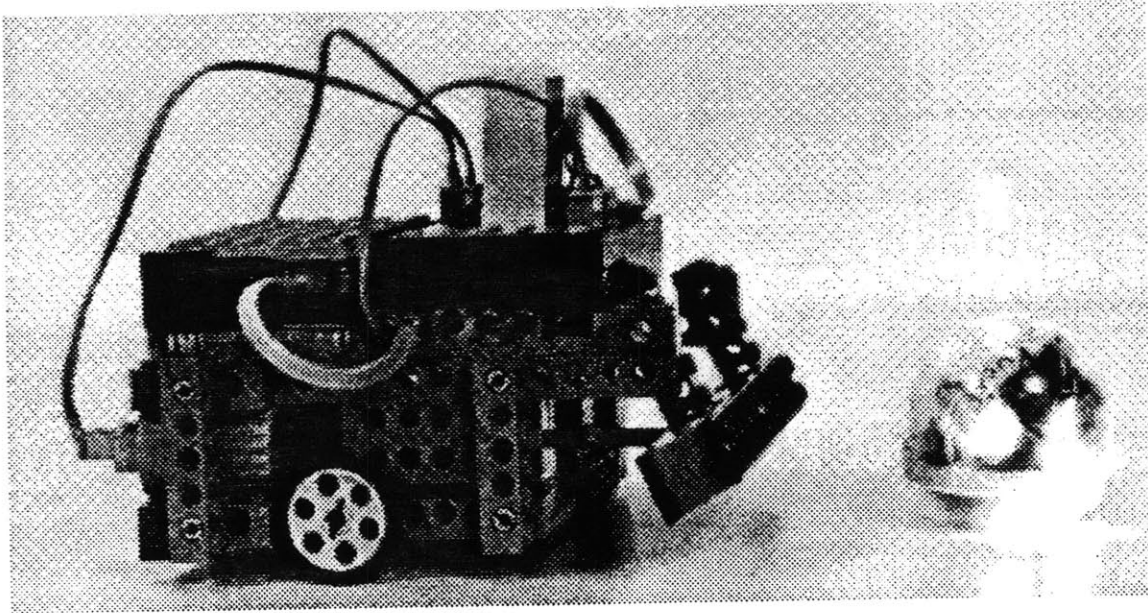


Figure 3-1: A Turtle Capturing a Light

towards a bulb, while others required a move of a foot. One final source of unreliability came from variation in the intensity of the light because of the use of different kinds of sources of light.

The Turtle The first kind of robot available to the children was called a “turtle”. One of these slow-moving robots is shown in Figure 3-1 face to face with the ball containing light bulbs that was used as a moving source of light in the workshop. The turtle is driven by two independent motors, each controlling one of the side wheels. Each of these motors is geared down through a combination of a pulley, a worm gear, and a sixteen tooth gear, so that the turtle’s maximum speed does not exceed a quarter of a foot per second. The turtle has one touch sensor mounted under a wide swing-arm bumper that is located at the front of the turtle.

The Beetle The beetle was the second kind of robot available to the children and was not introduced until the workshop’s fourth session. The children named this kind “beetle” because of its profile and speed. A beetle can be seen in Figure 3-2. Like the

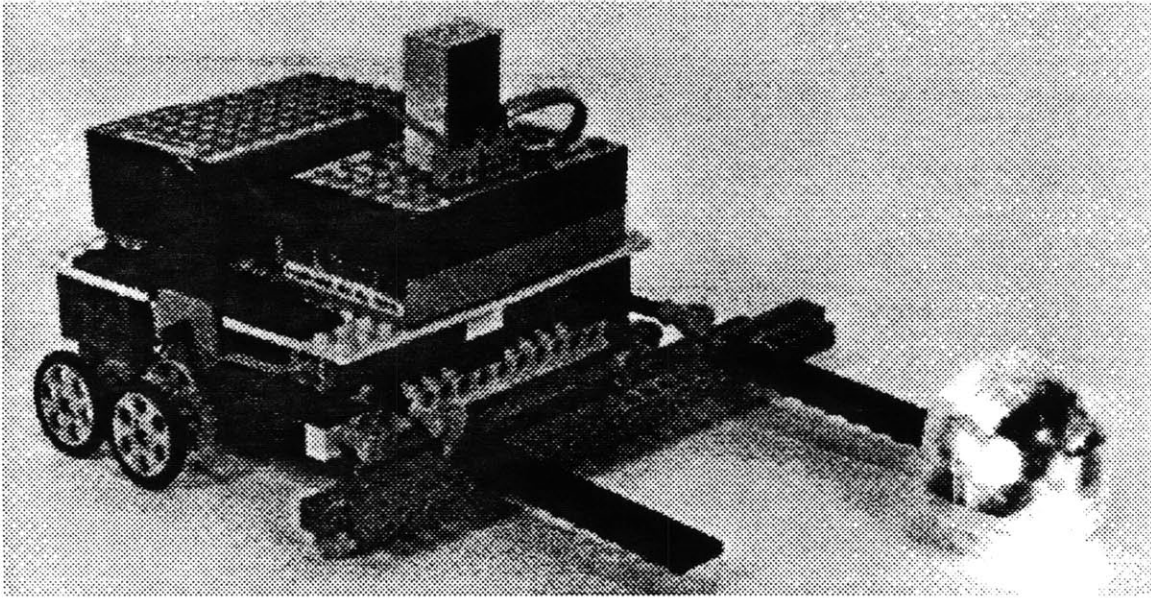


Figure 3-2: A Beetle Capturing a Light

turtle, the beetle is driven by two independent motors and has one touch sensor located under a front bumper. However, the beetle gears the motors down only through a pulley and a sixteen tooth gear so that it's maximum speed exceeds two feet per second. This extra speed made the children confident that the beetle could be efficient at capturing a light.

3.1.2 The Competition

The MIT 6.270 Robot Design Competition is described in Section 4.4. Every contestant in the competition was provided with an identical set of materials. While the competition used the same LEGO materials as the workshop to make the structure of the robot's bodies, its electrical and electronic components were provided by the contest organizers and came from a variety of sources. The motors had to be fitted with LEGOs so they could be mounted onto the robot, and the computer and sensors had to be assembled from discrete parts.

The Contest Computer The contest's computer was a 68H11-based microcontroller with 8K of ROM, 256 bytes of RAM, a serial communication port, and an interface that could have eight digital sensors, four analog sensors, and eight unidirectional driver ports. The driver ports could be paired to control up to four bidirectional effectors. Two types of effectors could be plugged into them: a solenoid and a bi-directional motor. The driver ports were attached to current-sensing circuitry that allowed the controlling program to know how much strain was put on the effectors at any moment. Other sensors included switches, analog light sensors, and four 40KHz infra-red detectors. Only assembly language could be used to program the robots.

The Contest's Puck Sensor The 40KHz infra-red sensor was the equivalent of the workshop's analog light sensor. It was used to detect the contest's "puck", a regular hockey puck with a one foot stem that carried 40KHz infra-red emitters. While this sensor could reliably detect the puck at a distance of more than six feet, it would only report whether or not it was detecting 40KHz infra-red and not the intensity of what it was detecting. In this respect, it provided the same information that the children eventually obtained from the workshop's sensor.

The SMART Machine Basing themselves on their analysis of contest rules and of the layout of the playing field, the children conceived of a robot which they named the Super Massachusetts Artificial Robot Trophy winner, or SMART for short. The robot that entered the competition is shown in Figure 3-3 face to face with the contest puck. The robot's design was similar to that of the Beetle, except that it sported a fall-down arm which was to be used by the robot in the last second of the contest and its hooper had tall sides. In Figure 3-4, the robot is shown with its arm having fallen onto the puck. Because the Robot Design Competition required its participants to make their own sensors from electrical parts, the author assembled the SMART machine according to the children's design. He also hand-translated the children's program from ACT to 68H11 assembler. Some parts of the children's machine could not be constructed as

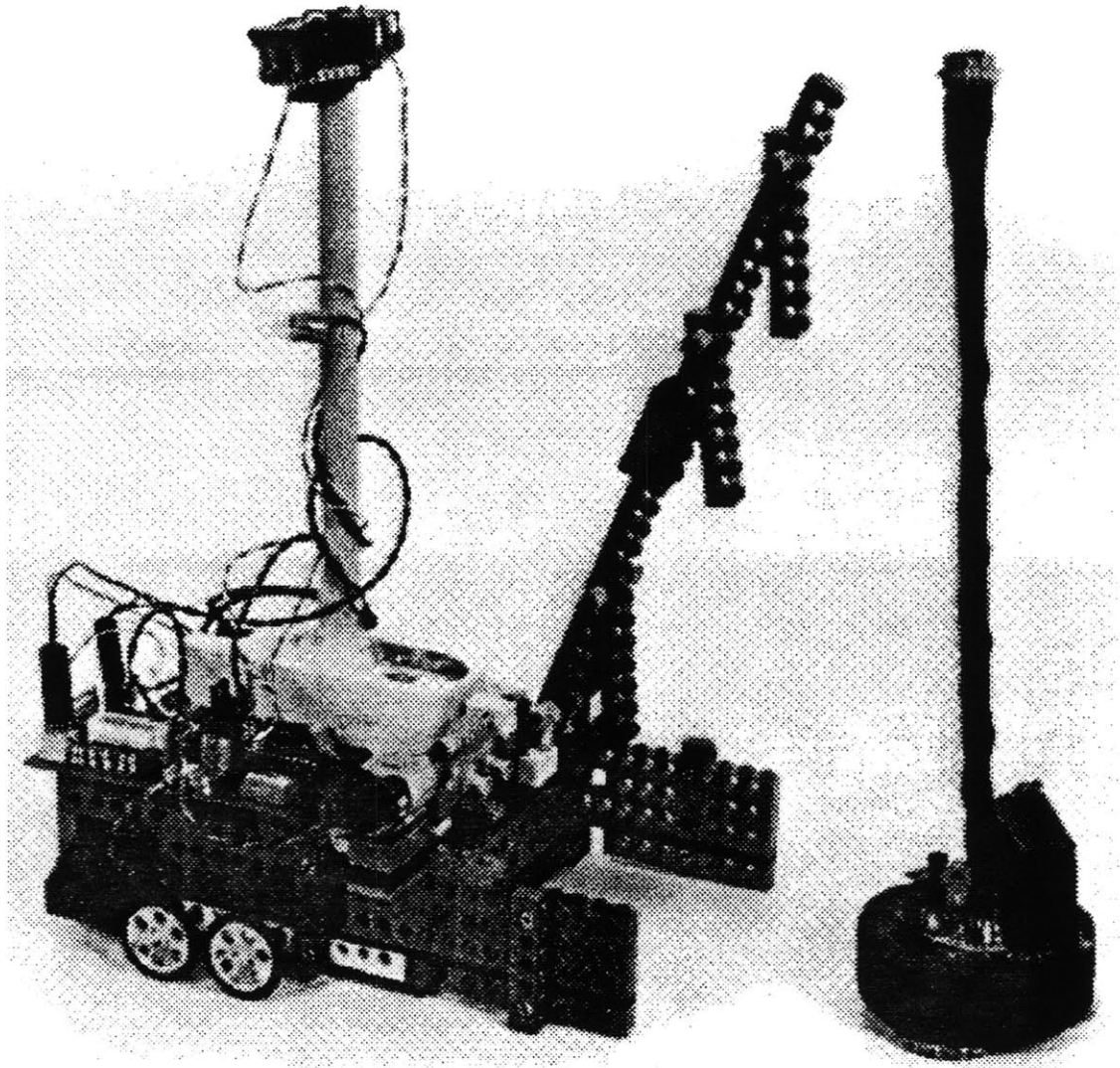


Figure 3-3: The SMART machine and the "puck"

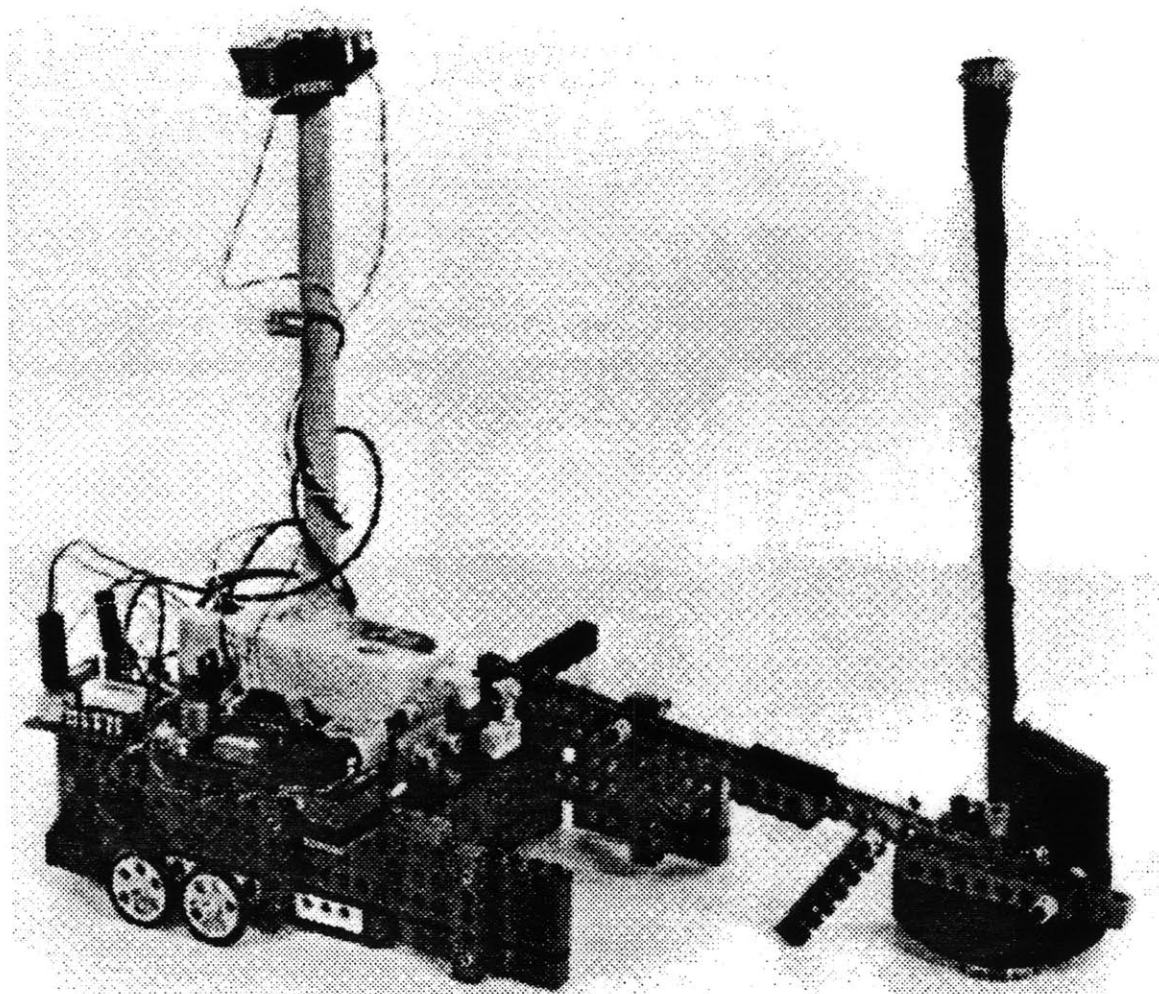


Figure 3-4: SMART touches the puck with its arm

planned in the week between the time the children finished their design and the night of the contest. However, the children were able to make changes on the day of the contest.

3.2 The Robots' Minds

ACT is a system for programming the reactions of robots to their environment. It includes an editor, a programming language, and debugging tools.

3.2.1 The Editor

The ACT editor is closely modeled after the editor of the LogoWriter programming environment available for personal computers. LogoWriter uses the metaphor of pages in a scrapbook for organizing different programs on a same disk. It was chosen as a model on the assumption that many children would have encountered it in school and that they would have some understanding of how to use it. In practice, eight of the children had worked with LogoWriter before the workshop. In Chapter 6, some of the children's difficulties with using the ACT editing environment will be discussed.

3.2.2 The ACT Language

The ACT language makes it easy to write parallel programs; sequential actions are possible but they must be explicitly created. ACT was designed to be an approachable version of the arbitration macrology language developed by Chapman for describing the control circuitry of the videogame players Pengi and Sonja (see [Agre 1988] and [Chapman 1990]). A description of Chapman's system can be found in Section 3.3.5.

The central organizing structure in ACT is the *action*. A typical action is shown in Figure 3-5. Actions can enable or inhibit motors from turning, or sensors from reporting their value, or even the functioning of other actions. Actions can also be used to abstract away from the details of a situation by joining a number of separate actions under a name, or by hiding the details of implementation of a set of actions. While actions seem

```
action back-away-from-obstacle
  (propose move-a direction: odd power: 7
   propose move-b direction: odd power: 3)
touching-obstacle?
```

Figure 3-5: A Typical Action

like the procedures of more conventional languages because both serve as a means of abstraction and both can have *parameters* that modify what they do, it is a mistake to think of actions as procedures. A procedure serves as a pattern of action for different invocations of the procedure, each of which possesses a local version of the procedure's parameters. In the process of executing a program written in a procedural language, the calling of any procedure causes the creation of a new invocation and so of a new set of parameters. By contrast, in ACT each action is a unique object and so there never exist more than one instance of its parameters.

ACT is built on a model of argumentation so the rule for determining which actions should be done can be stated thus: every action which is proposed and has no opposition is done. An action that can be done is called *valid*. ACT has no rule for weighing propositions against oppositions, but it allows opposing an opposition and therefore defeating it.

One way an action can be proposed is if its *condition* is true. This condition can be any logical expression and can involve the reports from sensors or from other actions. Such conditions allow ACT programs to get "off the ground" so they can act.

If an action is valid, every act described in its *body* is done simultaneously. There are five kinds of acts that can be described in the body of an action. An action can:

- propose another action,
- set another action's parameters without proposing it,

```
action move-a direction power
  (propose power-a power: power?
   propose spin-a direction: direction?)
```

Figure 3-6: An Abstract Action

- oppose another action,
- prefer one action to another without proposing either, and
- report a value.

These acts serve to control other actions. For example, by using a propose, an action says that another action should be done. If it had used an oppose, it could have said that that other action should not be done. By using a prefer, an action can say that given a choice between two possibly valid actions, the second one should be opposed.

The most basic actions, the ACT *primitives*, are those that control the functioning of motors; for example, action SPIN-A sets the direction of spin of the motor plugged in bidirectional port A to either ODD or EVEN¹. Other basic actions are those of reporting a sensor's state; for example, SENSOR-0 reports whether a switch plugged in port 0 is on or off.

As was said above, actions can be used to combine other actions under one name. In Figure 3-6, the action MOVE-A combines under one name the actions of setting the power output and the spin direction of a bi-directional port. This abstract action does not have a condition under which it might be proposed. Proposing an abstract operation is to be done by whatever actions use it.

A short reference manual for ACT is given in Appendix C.

¹Because the Programmable Brick cannot determine how a LEGO motor is plugged into its interface board, there is no way to guarantee that a particular direction of spin will happen for a particular setting. For example, that setting a direction of EVEN will make a motor spin clockwise. This is why the relatively neutral terms of ODD and EVEN were chosen to represent the directions.

3.2.3 Debugging Programs

The ACT debugger is a way for a user to make visible the functioning of programs. There are at least two situations in which information useful for debugging can be gathered: either while the robot is actually working in the target environment, or while the robot is on the bench and the user stimulates its sensors directly. The ACT debugger supports the second method which allows the users to know that their program is at least reacting as they had expected.

The debugger is used with the Programmable Brick plugged into the computer running the ACT development system. It can be used to see changes in the parameters of selected actions as the sensors are manually stimulated. In particular, it is possible to see the actions going between the active and inactive states. The user can directly change what the sensors report by, for example, shining a light into a light sensor or pushing a bumper's switch. Using this method, the children were able to find program bugs such as actions contending for the use of the motors.

For example, below is a program a child could have written to control a robot with a single motor attached to port A and a switch attached to port 0. The child wants the robot to normally move forward, but to backup when it hits something. The comments listed on the right give a running description of the meaning of the actions listed on the left.

```
action forward          ; Always propose to make the motor move
  (propose move direction: even) ; forward.
always
```

```
action backward        ; Propose to move the motor backwards when
  (propose move direction: odd) ; bumping into an obstacle.
bumping?
```

```
action move direction  ; Port A has a motor. Name 'move' the action
  (propose spin-a direction: direction?) ; of setting its direction.
```

```
action bumping         ; Port 0 has a switch. Name 'bumping' the
```

```
(report sensor-0)      ; action of reporting its state, and always  
always                 ; propose to make that report.
```

```
action live
```

```
(propose power-a power: 7 ; Always propose that the motor have power  
  propose sensor-0)      ; and that sensor 0 report.  
always
```

When the child downloads the program to the robot and runs it, she finds out that the robot doesn't back up when it runs into an obstacle. To see why, she plugs the robot into the development system, and tells the system she wants to watch the FORWARD and BACKWARD actions. Every action has local variables called parameters, and two of these parameters are automatically there for every action: `active` and `report`. `Active` is TRUE when the action is valid, and FALSE otherwise. `Report` is the value the child said the action reports if `active` is TRUE, and is 0 when `active` is FALSE. When the child tells the robot to take a "step", it computes what it would do next, given the current sensor inputs. After the decision has been made the debugger prints out what FORWARD and BACKWARD's parameters values are:

```
backward> active: false report: 0  
forward> active: true report: 0
```

The child can see that FORWARD is active, and BACKWARD isn't. Then, she pushes the bumper switch and tells the robot to take another step, and sees:

```
backward> active: true report: 0  
forward> active: true report: 0
```

So both BACKWARD and FORWARD are active at the same time, and therefore they're contending for the use of the MOVE action. This is a problem because BACKWARD wants MOVE's DIRECTION to be ODD and FORWARD wants it to be EVEN. Because EVEN wins out over ODD², FORWARD wins out over BACKWARD.

²Because the motor direction settings that FORWARD and BACKWARD propose are internally combined with a logical OR function.

To fix this problem, the child adds the action:

```
action backward-before-forward
  (prefer backward forward)
always
```

which says that if the robot must choose between doing either FORWARD or BACKWARD, BACKWARD wins out. She recompiles the program, downloads it to the Programmable Brick, and steps it once again with the bumper pushed, getting:

```
backward> active: true report: 0
forward> active: false report: 0
```

Now that the contention is eliminated, she can let the program run on its own, and try out her robot.

3.2.4 The Initial Program

The children were given an initial ACT program named MOVEAROUND that allowed a robot to move in a straight line until hitting an obstacle and then back up away from it. A copy of this program can be found in Section B.2. This program was intended to be an example of how the robot could be programmed to move about an area and react to obstacles. The program included two abstract actions, MOVE-A and MOVE-B, the first controlling the robot's left motor and the second controlling its right motor. The actions touching-obstacle and brightness showed how actions could be used to provide meaningful names to sensor ports. The actions back-away-from-obstacle and move-forward showed how these abstract actions could be used, and how actions could be grounded into the input from sensors. Finally, back-away-before-move showed how a prefer act could be used to choose between to possibly conflicting actions.

3.3 Related Work

There are at least two kinds of systems for constructing and controlling robots: the first kind of system uses actual robots, and are called robot construction systems, and the second kind simulates the robots and their worlds, and are called virtual-world systems. Two examples of robot construction systems are Brooks' subsumption architecture robots controlled with state-machines, and the Epistemology and Learning's LEGO/Logo robots controlled by Logo programs. Examples of virtual-world systems are the sensory turtles controlled by Logo programs shown in Abelson and diSessa [Abelson and diSessa 1980], Robot Odyssey which is an adventure game where players build logic circuits to control game playing robots, Mike Travers' Agar which is an animal construction kit where each robot is controlled through a society of interacting agents, and Chapman's video game playing system.

3.3.1 The Subsumption Architecture

The Subsumption Architecture was designed in reaction to difficulties experienced while using Artificial Intelligence planning techniques for constructing robots [Brooks 1986]. Brooks noted that the classical approach resulted a sequential decomposition of tasks into processing modules. As a result, the entire activity of the system depended upon the correct functioning of every processing module. Instead, he proposed to "decompose the desired intelligent behavior of a system into a collection of simpler behaviors" ([Brooks 1986] pg. 6). Furthermore, each behavior should independently produce "observable phenomenon in the total behavior of the system" ([Brooks 1986] pg. 6). These behaviors should together provide the robot with some level of competence. The behaviors can be seen as arranged in layers that can subsume one another's actions. The system's performance in a situation no longer depends on the weakest link in a processing chain, but rather on the strongest relevant behavior for the situation.

In her masters thesis, Maja Matic describes the Behavior Language that is used for

creating Subsumption Architecture robots [Materic, 1990].

The Behavior Language is a real-time rule-based parallel programming language, an extension of the *subsumption architecture*. In contrast to the subsumption architecture language, which was programmed with interconnected augmented finite state machines (AFSMs), the Behavior Language groups AFSMs into behaviors. Each behavior is a coherent collection of related real-time rules producing a related set of responses.

No information was available to the author at the time of writing this thesis about what debugging facilities were available for use with the Behavior Language.

3.3.2 Cybernetics and LEGO Robots

In 1987 and 1988, Fred Martin conducted an exploratory study of the uses of the Programmable Brick as a controller for LEGO robots [Martin 1988]. As described at the start of this chapter, the Programmable Brick combines the controlling computer and LEGO interface box into one, and can, when combined with a battery, allow the creation of free-moving robots. Martin's study was centered on the concept of feedback in the control of processes and involved programming pre-built robots for tasks such as following a light and sequencing the action of light bulbs to control the paths of robots. He worked with two groups of children, one from an Boston inner city school, and the other from a school the outer community of Newton.

Martin's materials for building robots were identical with those used in the study described in this thesis. However, he used the Logo programming language as a means to control the robots. Logo is a conventional sequential-procedural programming language invented in the sixties that is currently widely available in elementary schools. Because of its nature, any parallelism had to be either explicitly managed by the program written by the children, or came from using more than one Programmable Brick at a time. There was no debugger available for use with the Logo.

3.3.3 Robot Odyssey

Robot Odyssey is a logic game marketed by The Learning Company [Wallace and Grimm 1984]. It was created by Mike Wallace and Leslie Grimm and includes a tutorial, a construction and experimentation environment, and a game. The tutorial invites directed exploration by putting the users in an environment somewhat like the one they will be in in the game, providing them with instructions on what to do, and leaving enabled every feature of the robot that will be learned even though a particular feature might not be strictly necessary to following the instructions at a given moment. Using the tutorial, users can learn the basics of the robot's construction, how to use the circuit building tools, and how to create new circuits inside of a couple of fun hours.

The robots that are available to the user are pre-built. They have four bumpers and four thrusters, one per side. Robots have a hand that can be told when to grab and reports when it has grabbed. They have an antenna that can be told to broadcast and reports when it senses a broadcast. Robots are powered by a battery and can be controlled through enabling or disabling power to the thrusters. The robots move around rooms that are probably better called screens because they aren't a closed area. There can be more than one room in a screen, and a room can, and often does, spill over into another screen. The robot can have sensors specialized for particular kinds of objects. There can be three kinds of sensors for each kind of object: one that reports whether the robot is touching that kind of object, another reports whether there is at least one object of that kind in the room where the robot is located, and the last one reports in a North-East-West-South format the direction of an object of that kind located in the room.

Circuits can be assembled during the game out of basic components and microchips can be designed before the game, but new microchips can't be created during the game. Once a particular kind of circuit is designed in the construction lab, it can be compiled into a unit (burned microchip) that can be used as a component in other circuits. This is the only method of abstraction given in the game. Chips can communicate through 8

pins and these can simultaneously be inputs and outputs. During the game, players can select microchips designs to be used from a library of designs that they have previously created, but they are limited to using them by reprogramming whatever microchips they can scrounge up during play. The only way of debugging a circuit is by observing the passage of signals over the wires connecting the circuit's components.

The dependability of the game world and the limits of the construction environment makes Robot Odyssey a game played by a user who makes some use of automatic machines, basically semi-automatic tele-operators. It is not a game construction kit and is not a full robot construction environment.

3.3.4 Agar

Agar is an animal construction kit whose ideas come in part from Marvin Minsky's Society Theory of Mind [Travers 1986].

The *Society of Mind* refers to a broad class of theories that model mental functioning in terms of a collection of communicating agents. An agent is a part of a mind that is responsible for some particular aspect of mental functioning. Agents communicate with and control each other, coordinating their activity to produce a coherent mind.

[Travers 1986] pg. 19

As a virtual world system, Agar provides the tools for the creation of both creatures composed of agents and the worlds they live in. This thesis will only look at the means of creating the creatures.

Agar was originally written on a Symbolics Lisp Machine and made use of its object-oriented programming facilities as a means of making creatures. A creature is a computational object created using a `defcreature` form that specifies amongst other things what kinds of sensors the creature possesses. Motor functions of the creature are defined as methods for the creature that can be called by agents. The agents themselves are

defined using a defagent form that associates the agent with a particular creature and specifies its behavior.

An agent consists of a condition (if part) and action (then part), plus a gating mechanism that decides when the rule is to be run. The condition part of a rule is typically a sensor predicate, and the action is either a motor function or a command to activate or inhibit other agents.

[Travers 1986] pg. 30

Most agent-to-agent communication is done by simple activation although Agar has mechanisms by which an agent can be more specific about what it wants another agent to do.

Debugging of societies of agents in Agar can be done using the Symbolics Lisp Machine facilities. Travers also provided various tools to facilitate novice usage of Agar. From Travers' thesis:

- the ability to select a particular creature with the mouse. Subsequent commands and debugging displays would then pertain to the selected creature.
- the ability to start, stop, and single-step the agents.
- a continuously updated display of the internal state of the selected creature or of one of its agents.
- a facility for displaying the sensitive range of individual sensors (as a circle surrounding the creature, for example).
- a facility for manually disabling particular agents.
- a facility for displaying and editing an agent's definition while the creatures are running.

[Travers 1986] pg. 42

3.3.5 MACNET

MACNET is the system used by David Chapman to build Sonja and is direct descendant of system used by Chapman and Agre to build Pengi ([Agre 1988], citechapman). It is a language for constructing clocked digital circuits and it provides its user with constructors for the basic logical operations and for latches. MACNET is a language embedded in a Lisp environment so that circuits are built by running Lisp programs. The results of running these programs can then be run on a circuit simulator.

MACNET's primitives are the logic gates *org*, *andg*, and *invert*, and the state element *latch*. Each of these primitives returns an object which represents its output wire and requires as inputs either constants or other wires. Higher order constructs can be built using Lisp functions; for example, exclusive-or could be implemented with:

```
(defun (xorg a b)
  (andg (org a b)
        (invert (andg a b))))
```

MACNET was made to create the central circuitry of a game player. This circuitry controls a limited number of *operators* whose parameters it can set and from which it receives information. Some of the operators represent the available motor effectors but many control of the processing of sensor information.

For reasons of modularity and to keep the complexity of circuit descriptions under control, Chapman developed a set of macros for writing a game player in MACNET. This *arbitration macrology* includes statements for creating *proposers* for the values of the parameters of operators, declaring *conditions* under which a proposer is valid and *overrides* of one valid proposer by another, and *default proposers* that are automatically overridden by valid proposers. It also includes mechanisms for lumping a set of proposers under one abstract proposer and for creating abstractions of a set of operators under one name. There can be *supports* and *objections* to proposals, either naming the proposals or the operator states involved. They are mostly used in conjunction with the *instruc-*

tion buffers, a mechanism that allows someone observing a game player to interact with it. Supports and objections are different from overrides in that they do not guarantee that a particular proposal will be valid or invalid but rather bias the choice the arbitration circuitry will make in cases where it has no basis for choosing between conflicting proposals.

As the arbitration macros build the circuits they represent, they also create additional circuitry for detecting arbitration errors: cases when even after arbitration there still is more than one proposal that wants to set a particular operator. The additional circuitry triggers an error condition that causes the circuit simulator to enter a debugging environment. To facilitate debugging, it is possible to name any circuit that is created. This name can be used to find out why that circuit produces the output that it has at a particular moment. Another debugging tool allows the printing of the values of named circuits as the program runs. It is also possible to cause the system to enter a debugging loop when a particular named circuit changes state. An operator's state can be displayed including the state of the proposers, whether valid or invalid, that would like to set its inputs. Finally, the contents of the instruction buffers can be listed.

Chapman's arbitration macrology language inspired the design of ACT, so that language has most of the capabilities of the macrology. ACT sought to make these capabilities more approachable for novice, so it introduced the *action* as construct around which they could be organized. Actions also subsume the macrology's specialized constructs for creating abstract operators and proposals, and the facility for naming language constructs for debugging. However, ACT does not include capabilities beyond those of the macrology so it does not represent a technical advance over it.

Chapter 4

The Workshop

The workshop was done in the context of the Brookline Public Schools' Gifted and Talented Program. Approval for doing this project in a the context of the program was obtained from the Brookline Public Schools' Testing, Research, and Evaluation Committee. An overview of the workshop's activities will be given. Then, the means by which the participants in the workshop were chosen will be described. Finally the workshop time-line and a sketch of its sessions will be given.

4.1 Overview

The schedule of events as described in this chapter might give the impression that the workshop was mostly composed of lectures. Nothing could be further from the truth. While in the first session the author provided an ultimate goal for the workshop, its schedule was not fixed beforehand, but rather emerged from the rate at which new materials became available and from the choice of pace of work made by the children. But this does not allow qualifying the workshop as being child-directed because this would not take into account the participation of the author.

Workshop sessions would typically begin with a half-hour to forty-five minutes of socializing where the students and the author would discuss their experiences during

the previous week and many other subjects. Then the author would suggest that the discussion turn to the programming of robots. In the first few sessions of the workshop, new materials were often introduced and demonstrated at the start of the session. Then, the rest of session would consist mostly of the children integrating the materials into their work. In those sessions, the author's role consisted in responding to the children's requests for help and advice, and in asking the children to describe what they were doing. Towards the end of the workshop, after the children accepted to participate in the MIT Robot Design Competition, the start of a session consisted mainly in deciding how to best use the rest of that session to complete the work at hand.

The children did not work uninterruptedly towards their goal for a session. Rather, work would often be interrupted by discussions of many different subjects, especially towards the end of the workshop. The subjects the children discussed were diverse and included their views on school, education, the nature of intelligence and whether machines could ever be said to be intelligent, computer viruses, shareware, and who exactly would be their competitors in the MIT contest and how good they would be. A full recounting of these discussions is beyond the scope of this thesis.

4.2 The Participants

The children who took part in the workshop were from a variety of schools in the Brookline Public School System and knew each other from previous Gifted and Talented Program workshops or other school-related programs, such as Math League or Spelling Bee competitions. They were selected by Susan Cohen, the director of the program, because of the interest they expressed in the workshop based on the description of its goals as outlined in a handout prepared for their parents. The workshop began with a group of eight children, three boys and five girls. The children were Ben, Debbie, Julie, MatthewP, MatthewS, Meagan, Sarah, and Sargeant. A new boy, Misha, came to the workshop at the second session, but he and Meagan stopped coming by the fourth session from lack

Date	Main Event
Nov 29, 1989	Observing light seeking turtle robot
Dec 6, 1989	Editing and programming in ACT
Dec 13, 1989	Compiling and debugging programs
Jan 3, 1990	Using the Beetle robot
Jan 10, 1990	Using a moving light
Jan 17, 1990	Children's robot competitions
Jan 24, 1990	Programming the 6.270 competition robot
Jan 31, 1990	Designing the 6.270 competition robot
Feb 7, 1990	MIT's 6.270 Robot Design Competition
Mar 7, 1990	Post-interviews

Figure 4-1: Schedule and Main Event by Session

of interest. Susan Cohen attended the first session of the workshop but only some of the sessions that followed. She would not always be present for the entire duration of a session but rather would arrive in its middle and depart before its end. The author was present at every session.

4.3 Workshop Time Line

The workshop was composed of nine sessions each lasting two and a half-hours and happening on a Wednesday immediately after school. Between the eight and the ninth session, five of the children attended the MIT Robot Design Competition to watch a robot they had constructed compete as an exhibitor. The ninth session happened a month after this contest. A listing of each session's date and of its main event is given in Figure 4-1.

In the first session, the children were introduced to the LEGO robots and the ACT programming language. In the following session the children were introduced to the ACT programming environment and were given a model of how the ACT language behaved.

In the third session, the children were given the ACT compiler, were taught how to find and fix syntax errors, and were taught how to use the ACT debugger. The fourth session, the first one after the Christmas and New Year vacation, had a refresher in what had been covered, and the new LEGO robot, the “Beetle”, as introduced. At the fifth session of the workshop, the children were told they could participate as an exhibitor in the MIT Robot Design Competition. This yearly contest involves building a robot that competes against three other robots without human intervention (see Section 4.4). At the sixth session of the workshop, the children ran a contest to determine which of their program was the fastest at reaching a light, and chose it to control their entry in the MIT competition. The seventh session focussed on determining how the competition robot would act in the contest. The eighth session, the last one before the competition, was used to complete the design of the machine to be built for the contest. A final session was held after the competition to get the children’s reflections on the workshop and on their designs.

4.4 The Sessions

Session 1

The children were introduced to the workshop’s goals and materials. It was emphasized that the workshop was not a class, did not have grades, and that the children were free to stop coming at any time. The children were shown the Turtle, Programmable Brick, and light sensor and their operation was demonstrated. The idea of the ACT language was introduced and its parallel nature was discussed. The children were walked through a written example of an ACT program that makes a robot move in a straight line and backup and turn when it hits an obstacle (see Section B.2.) Then, they observed the interaction of a robot with a light in a similar manner as was done for the preliminary experiments outlined in Section 2.4. Although in the preliminary study the adults were not allowed to handle the robot, in the children in the introductory session were allowed

to move robot themselves. They were asked to write their observations about the robot's activities and to describe aloud what they believed the robot was doing. After having observed the robot, the children were given the correspondence between the different procedures and the trials and were given descriptions of what each made the robot do. Then, they were asked to describe what procedure they would make the robot follow in order to reach the light. After each of the procedures had been discussed, the children were asked if they had seen patterns in the robot's activity. At the end of the session, the children were asked to list what previous experiences they had had with Logo, LogoWriter, and LEGO/Logo. They were also asked if they had regular access to computers like those used in the workshop (Apple II GS).

Sessions 2 and 3

In session 2, the children were split into groups of two. They were given the ACT editor and taught how to operate it. They were given a model to think about what an ACT program would make a robot do. In session 3, the completed ACT compiler and debuggers were introduced to the children. They were shown how to compile a program they had written, how to interpret the syntax errors that the compiler reported, and how to correct them. Once they were able to compile programs, they were shown how to send them to the Programmable Brick and each team was encouraged to try out whatever program they had up to then. Then, they were introduced to the debugger and were shown how to see what actions were triggered by a selected stimulus. They were shown how to correct the bugs they encountered.

Sessions 4, 5, and 6

Session 4, was the first session after the Christmas and New Year vacation. The children were given a refresher in how to use ACT system with the Programmable Brick. A new model of a floor robot was introduced, and MatthewP promptly named it "Beetle". The children were encouraged to try out this beetle. Those children who already had working

programs were encouraged to help those who were having difficulties working with the ACT system.

At session 5, a transparent plastic ball that held a battery and three light bulbs was introduced and the children were asked to make their robots find it while it was rolling about the floor. Also, the children were told they had the opportunity to participate in the MIT Robot Design Competition, a challenge which they immediately accepted. A brainstorming session was held about what kinds of methods could be used to win the competition.

In session 6, the children held a contest to determine which of two machines was the most efficient at finding a fixed light. They analyzed the rules of the competition and were told how the competition equipment differed from the equipment that was used for the workshop. Another brainstorming session was held on the method by which the robot would win the competition, with a special emphasis put on how it would know how it had captured the competition light.

Sessions 7 and 8

In the final two sessions, the children worked on designing and programming the robot that would take part in MIT's Robot Design Competition. This robot would have to compete against other robots to be the last one to touch a source of infrared light.

The Robot Design Competition

A week after the end of the workshop, the children participated in the MIT 6.270 Robot Design Competition as exhibitors. This MIT event has happened every February for the past five years. It is the Electrical Engineering Department's version of the famous Mechanical Engineering Department's 2.70 Machine Design Competition. The 2.70 contest requires the participants to design, build, and control their own tele-operator devices starting from the same simple set of parts. That course is designed to emphasize the principles of mechanical design.

In contrast with the 2.70 course, the 6.270 contest stresses the electrical, electronic, and computer science aspects of building a device. For this reason, the robots must be able to compete without any human intervention whatsoever. In early competitions, the robots were only programs that competed in virtual world environments. The February 1989 competition was the first one where the robots were actual machines that were built out of LEGO parts¹. The February 1990 competition was the first one where the robots were controlled through an on-board computer similar in capability to the Programmable Brick used by the children (see Section 3.1.2). Preparation for the competition happened over a three week period in January, and included a robot design course where MIT students could learn about batteries, assembling electronic components, and designing LEGO machines. The children did not attend the MIT class.

On the day of the contest, Sarah and MatthewP were able to see the machine and make final changes. Five of the children, Julie, MatthewP, MatthewS, Sarah, and Sargeant were able to attend the event and observe their robot in action. Their robot won competing against the three other robots that were in their category.

Session 9

A month after the robot design competition, the children reviewed a tape of some of the competition's events. Then, they were asked to reflect on the activity they observed and to re-think the design of their contest robot as if they would be able to compete one more time.

¹LEGO parts were used to minimize the mechanical engineering aspects of the design process.

Chapter 5

Observations and Analysis

5.1 Research Questions

There are two directions that can be taken in the analysis of the workshop. One looks at the children's appropriation of interactionist ideas while programming the robots, and the other looks at the difficulties the children had in using the ACT system that had been created to support this activity. In the beginning, the children were uncertain about the nature of the task they were asked to accomplish and did not know what they could do with the tools. As the workshop progressed, the children came to understand what they could do with the tools and this knowledge allowed them to elaborate and transform the goal of the workshop and create solutions that they found satisfactory. This chapter will reflect this progression by beginning with a focus mostly on the children's mastery of the tools and then moving to analyzing the children's assimilation of the ideas.

The question that was at the fore in this analysis of the data from the workshop was about the nature of the process through which the children were able to use dynamics. When looking at the introductory session of the workshop, this question was worded so as to ask whether the children could recognize and construct emergent phenomena. In analyzing the events in the workshop, the question became one of finding out whether the children learned that something had emerged, whether they could identify what

had created the emergent phenomena, and how they transformed this phenomena to accomplish their goal. Overall questions included whether the ways of looking at emergent phenomena used by the children varied over the course of the workshop, and what were the difficulties the children experienced while using the tools and how could they be removed. These questions will be taken up in Chapter 6.

Transcript Conventions

The transcripts in this chapter will be indented from the normal text and printed in a smaller font: “It’s going to find a brighter”. Each utterance will be preceded by the name of the person who said it; when it was not possible to attribute the utterance, the name “Unknown” will be used. Descriptions outside the transcript will be bracketed and italicized: “[*correcting herself*]”. The symbol “()” will be used to indicate an untimed pause. Stressed words or parts of words where relevant are shown by underlining the affected text: “brighter”. Connected additions to words will be shown with the use of a long dash: “bright-er”. Simultaneous utterances are not indicated. The symbol “...” will be used to indicate an ellipsis. Each participant will be referred to using their given name. The first letter of the family name will be used to disambiguate cases where the given names are identical. In the transcript as well as in the text, the author is referred to as “Mario”.

5.2 The Children

Of the nine children who took part in the workshop, eight had worked with LogoWriter before and those same eight had worked with a different version of Logo. Only Misha had no previous computer experience, but he reportedly excelled in mathematics¹. The other children had encountered Logo as part of their classroom activities. Only one child, MatthewP, had worked with LEGO/Logo before the workshop, and this had happened

¹Susan Cohen, personal communication.

as part of a program offered by the Boston Museum of Science. MatthewP was also an experienced BASIC programmer and had a number of games written in BASIC to his credit. Debbie had previously encountered an activity involving observing LEGO robots as part of a science class she had attended in the Fall of 1989. Of the children at least three had regular access to personal computers. MatthewP had an IBM PS/2, Meagan had an IBM PC, and MatthewS had a Macintosh.

5.3 Learning About the Workshop

In the first session the children were introduced to the LEGO and electronic materials, and were shown a light-seeking robot in action. There were eight children present at this session: Ben, Debbie, Julie, MatthewP, MatthewS, Meagan, Sarah, and Sargeant. Mario led the session, and Susan Cohen, the director of the Gifted and Talented Program, also participated.

Introduction to the Materials

The children were told about the sources of the ideas for this workshop; Martin's work with the Programmable Brick and Agre and Chapman's work with game-playing programs were mentioned. The children were asked to draw on their experiences while programming the robot: "imagine what you would do were you the robot, then tell the robot to do that."

The Programmable Brick was introduced as the equivalent of an Apple II+ computer with less than the usual amount of memory but with a LEGO interface board. This last part was introduced as something in which sensors could be plugged in, and sensors were defined as being switches or a light sensor. One of the light sensors that was to be used in the workshop was brought out and shown to the children. It was introduced as something the robot could use to see in front of itself the brightness in a cone about one

third of a circle wide. The light sensor was plugged into the brick's number 2 port² and the children were told the computer could be asked "what do you see out of this port?" The brick's sensor ports were shown with their numbers, and the driver ports and their letters were shown. The children were told that the command `LEVEL 2` could be used to find out what was the brightness reported by the sensor. The children were told that both motors and lights could be plugged into the brick and that the brightness with which such lights shone could be set. The children were asked if they've ever had experiences with LEGO/Logo, and only MatthewP had had one³. LEGO/Logo was described and the role the Programmable Brick could play in that context was explained.

The Nature of the Light Sensor

At that moment, MatthewP asked for more details about what the light sensor could do.

MatthewP: *[unintelligible]* light sensor. How wide is the tracking?

Mario: The () light sensor?

MatthewP: Yah.

Mario: ... it senses the brightness in about a third of a circle in front of it. *[Mario's hands are in front of his face and splayed about 120 degrees apart.]* So if you hold your hands about a hundred and twenty degrees apart, somewhere in there *[his right hand sweeps the angle between the hands]*, is something as bright as the brightness *[the sensor]* reports.

That explanation is not quite correct as the brightness reported will be proportional to the light that reaches the light sensor and not really that of a bulb emitting that light. However, it is simpler to understand than the true situation, and, as is shown later in this section, it will later be corrected.

²Ports are places on the Programmable Brick where electrical and electronic equipment is plugged.

³As explained in Section 5.2.

MatthewP shows he already has something in mind about how to use the sensors when he asks how to control them.

MatthewP: Can you set how narrow that be?

Mario: No, it's not possible to ()

MatthewP: Cause then, it can turn around until it sees a light.

Mario: No. Ah. No, it's () What you could do is () you could put baffles/*blind*ers/ on it [*his hands show blinders on the sides of his head*] like like you do for a horse, if you only want a horse to look in one direction. Does somebody ride horses?

Julie, Sargeant, MatthewS, Unknown: No, no.

Ben: I know what those are.

Meagan: Blinders, yah.

A problem with the Programmable Brick allows the discussion of the experimental nature of the materials. In particular, that their unreliability is bound to show up in the children's future experiences with them.

The object of the workshop isn't completely solid for the children. Sargeant asks Mario to clarify the link between finding lights and the light sensor, and he answers:

Mario: This robot can be set free to move around, and with the [*light sensor*] it can see brightness and the eye senses about a third of a circle in front of it. [*He holds his hands out in an angle slightly more than 120 degrees.*] So that means that if there's a bright light right besides it, it won't see it [*His left hand is still out, and he plants his right fist behind the line that used to be shown with this right hand.*] But if there's a bright light here [*his right hand moves out into the 120 degree area*] [*the robot will*] be able to tell that there's a bright light somewhere in front of it. [*His right hand sweeps out an area 120 degrees wide starting at his left hand.*]

Sarah is not satisfied with this explanation and wants to know the resolution capabilities of the sensor.

Sarah: But can it tell can it tell say this is the area it can see out of, [*she holds out her hands in front of herself, 120 degrees apart*] and there's a bright light here. [*Her right index finger stabs into the 120 degrees, in the top right corner.*] Will it be able to tell there's a bright light, you know, the right corner of it [*her right hand points straight from her body towards the point where the right index finger used to be*] or will it just be able to tell that somewhere where it can see [*her right hand waves vaguely over the 120 degree area*] there's a bright light.

Mario: Somewhere where it can see there's a bright light.

Meagan asks about the sensitivity of the sensor.

Meagan: But does it have to be a bright light? Can it be () like if the room is dark, can it be even the dimmest light?

Mario: Yes, yah.

Meagan: So any amount of light?

Mario: Weellll, almost any amount. I mean it () it it () it it it can't it gives you a number from zero to two hundred and fifty-five, ah zero begin the brightest number and two fifty-five being the darkest number⁴, so ah ()

Julie is not sure whether this means that the report is absolute.

Julie: You mean how how how bright the light is?

Mario: Yah, it tells you how bright it can see.

Julie: Oh!

⁴This changed at the third session to bring it in line with the motors: the greater the amount of light sensed, the higher the number

Mario's answer is ambiguous, and this might have caused Julie's comment in the final third of the session that the robot should be told how bright the light it was seeking would be.

Once the machine is working, the programmable brick is set to continuously print out the intensity reported by the light sensor. This makes the children return to the question of how the light sensor works. In particular, they want to know if it can locate a bright light in a field of darkness.

Meagan: If it sees really dark and really bright, which one does it print out?

Mario stresses that the light sensor is not like their eyes in that it can't locate an object. He uses the example of a frosted glass to explain that the light intensity reported will be in between dark and bright. This helps straighten out some of the incorrect aspects of the model that Mario had previously given to the children.

The Programmable Brick and the Turtle

The Programmable Brick is then shown to run on batteries, and is then installed on the back of a turtle. Mario downloads a program that makes the turtle move forward until it runs into an obstacle, and then back up while turning away from the obstacle. The turtle is unplugged from the Apple IIGS workstation and made to run around on a table which makes Sarah remark that she now understands that the programmable brick remembers programs even when unplugged. Susan tells Mario that Debbie has told her that she's had a recent experience in school observing LEGO robots and taking notes about them. When Debbie was later asked where she had had this experience, her answer showed that she has worked with Nira Farber's *Weird Creatures* [Farber 1990].

Mario puts the programmed turtle into the children's hands and shows them how to turn it on and off. While they experiment with it and pass it around, he explains that the turtle can be made to remember information and report it through the workstation. Mario also adds that it can be made to remember more than one program, so long as the

programs have different names. This introduces the idea that multiple programs may be included in one computer and prepares the way for the observation part of the session.

Mario uses the fact that the subject is programs to introduce the children to the idea of the parallel ACT language. He contrast this parallelism to Logo's one thing at a time process, and gives the children the central evaluation rule of ACT. He leads them through a printout of the MOVEAROUND program pointing out the potential conflict between the MOVE-FORWARD and MOVE-BACK actions and telling them how the PREFER action is used to choose between them. Susan asks Mario if the text between the parenthesis is a comment on the program, but he replies that that text is the program. Julie says that the ACTION line is how actions get named, and Sarah likens it to Logo's TO line.

Mario hands out to the children notebooks and asks them to use it to keep a record of their thoughts and work in this project. Susan is also offered a notebook but she refuses it. The children will use the notebooks at least today to record their observations of light-seeking turtle actions. There is no record of them using it at any of the later sessions.

Initial Program

Before the children observed a light-seeking robot in action but after they had been told of the task to be done in the workshop and heard descriptions of the light sensors, one child, MatthewP, showed the author a pseudo-BASIC program that was intended to make a robot find a light. This program and the two-sensor robot it was to control are pictured in Figure 5-1. Figure 5-2 has a paraphrase of this program. The robot was to have one sensor mounted so that it faced the direction of motion, and the other mounted so that it was at right angles, pointing to the right of the direction of motion. The front sensor was to have blinders that would prevent it from registering light from the side of the robot. MatthewP said he assumed that a sensor reading of less than 7 meant that the sensor was detecting a light, although he said he wasn't certain that the test was correct.

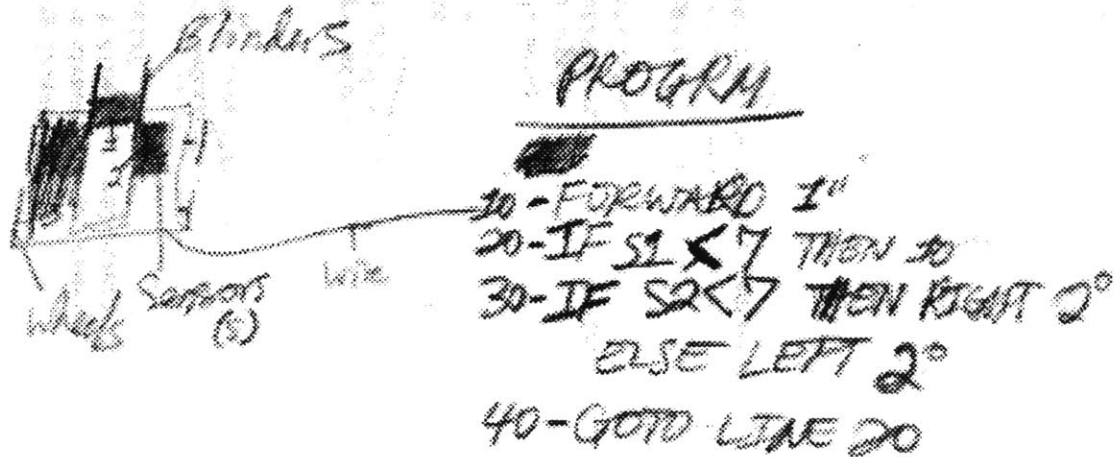


Figure 5-1: MatthewP's Light-Seeking Robot

```

10 FORWARD 1 INCH
20 IF SENSOR-1 < 7 THEN 10
30 IF SENSOR-2 < 7 THEN RIGHT 2 DEGREES ELSE LEFT 2 DEGREES
40 GOTO 20

```

Figure 5-2: Paraphrase of MatthewP's Light-Seeking Program

In describing the statements in his program, MatthewP said that it was important that “you don’t go forward until you make sure your light is forward of you.” This program serves as a base measure of how some of the children were thinking about the task before they started observing the light-seeking robot.

MatthewP’s program was different from the ones he was about to observe in several important respects. It did not detect possible obstacles in the path of the robot. Also, the program showed that at that time MatthewP’s model of the light sensor was that it was able to sense light in a half-circle (180 deg), although he had been told in the introduction session that the sensor was limited to a third of a circle (120 deg). This limit in combination with MatthewP’s sensor arrangement could make his robot have a heading in which it would be paralyzed, oscillating between two actions. Because MatthewP’s forward sensor has blinders, his robot would have had a heading in which both SENSOR-1 and SENSOR-2 would not report sensing a light in that direction, although his program assumes that it does make such a report. If the light were directly to the right of the turtle, SENSOR-2, the right-pointing sensor, would report light, and this would cause the program to make the robot turn right until both sensors would not detect light. Then the program would make the turtle turn left, but this would make SENSOR-2 report it saw light once more. As a result, the turtle would alternately turn right and left while never stepping forward.

Learning About ACT

After MatthewP showed Mario his program, he turned to the question of how ACT programs work. He wondered whether proposing an action would cause it to be done “until you say stop”. Mario specified that an action is done as long as it is proposed but stops being done the moment it stops being proposed. MatthewP changed his question to how the action might stop being proposed, so Mario gave as an example that if MatthewP had proposed an action, he might stop proposing because he in turn would not be proposed. This recursive explanation did not appear to satisfy MatthewP so he

became more concrete about what he wanted to do. He asked how to make the motor stop, and Mario told him he could set the motor's power to zero. MatthewP then asked how an IF statement, one of two control structures in BASIC program, could be done in ACT. Mario explained that while there was no IF statement per se, the same result could be gotten by saying that a given action could be done under a specified condition. Then Mario explained how actions could be made to report values.

Children's Observations of the Robot

About one hour in the session, Susan suggested that the group move on to watching the robot in action, and Mario agreed. Susan specified that she expected that the group would come back afterwards and see "where in the program [*the robot*] was doing what [*the group*] recorded seeing." Mario downloaded to the turtle robot the set of programs used in the preliminary experiments described in Section 2.4 and the group headed out to a corridor of the school.

There is no record of the first twenty minutes of observations because of a camera problem. Notes taken after the session indicate that in this time, the children worked with the turtle on the landing of a stairwell of the school because that area was not carpeted and this would give the turtle good traction. Through prompting from Mario, they began making descriptions of the turtle's activity. They soon asked to move to a carpeted corridor and, after trying the turtle on this floor, Mario accepted to move. Susan found a desk lamp to serve as a source of light for the turtle to work with and placed it on the floor near a wall outlet. By the time the camera is recording again, the children have observed the robot following two different programs for locating the light. Often in the course of the observation period, Susan would ask the children if they were ready to "make an observation," and the children sometimes took this as an indicator that she thought a particular turn had run its course. Sometimes, after such a request, the children would ask to move on to the next program.

Before the observation of the turtle using the third procedure begins, Sarah asks if the

robot could be using over again one of procedures it had used previously. Mario replies that that's possible although he wants to leave it up to the children to decide whether that is the case. MatthewS asks if every turtle uses the light sensor, and Mario replies once again that that's up to the children to decide.

As the children observe the turtle moving about, MatthewP casually waves his hand near its light sensor while keeping his eyes on the turtle as if he wants to see if it reacts in any way. While the children moved the robot about before, the notes do not show them intervening with its activity before this moment in this way. As the session progresses, the children will move from relatively passive observation to direct intervention in the activity. The floor light is off and the children decide to turn it on, but Ben asks them to wait until the turtle is pointed away from the light. When asked why he wants it done this way, he begins to answer but his words are drowned out by Sarah as she offers the explanation:

Sarah: *[unintelligible]* if it steps towards the light then it won't be a coincidence.

A minute later Mario asks the children to venture a guess as to what the turtle is doing. Cutting off Meagan, Sarah says "maybe it's looking for something dark," and Meagan chimes in with a "yeah". Thus, the children are already forming hypothesis about what in the environment causes the robot to move, and they are beginning testing them by directly interfering with the ongoing activity.

Sargeant moves up to the turtle and puts her hand directly in front of it, moving it as the turtle changes heading. As Meagan puts her hand over the light sensor, she mumbles something to Sargeant. MatthewS asks Sargeant why she is putting her hand in front of the turtle:

MatthewS: What are you looking for?

Sargeant: *[Continuing to block the turtle's path,]* I'm seeing if it's not have if it doesn't have anything to do with the light () if it has something to do with the () objects in front of it.

But Susan brings to the children's attention that the turtle turns even when there's no object nearby.

Susan: But it's turning even when there isn't anything in front of it, isn't it?

As the children continue to interact with the robot, Susan asks:

Susan: Is it turning is it going forward a set number of steps before it turns?

She will later come back with this comment and prompt the children to take notes about the turns and steps of the turtle. In this stage of their observations of the robot, the children have established that they can make it react. They are now testing their hypothesis about what directs its activity. However, they are only beginning to focus their efforts on particular parts of the robot and instead are mostly acting as controllable features of the robot's environment.

Later, while observing the turtle being controlled by a different procedure, MatthewP takes the turtle's light sensor in his hand and puts his thumb over it and then off again.

MatthewP: Look at this. It's seeing light and then I darken it and it turns () and then it goes forward. I make it see light again and () it turns?

Finally, they are stimulating the robot directly to control the direction it moves in and the stimulus it receives. They have drawn some conclusions about to which modalities in the environment the robot is reacting, and about what parts of the robot mediate this sensation. Later, they focus on the robot's program that invisibly connects its sensations to its actions.

Sargeant: ... if it sees the same amount of light if it's taking in the same thing () then it'll go towards it and it will keep on going forward and it won't move but if there's some change at all, it'll turn and then when it finds something that's set, it'll keep going forward.

This description is from the viewpoint that the robot is following the goal of finding a set light, and Sarah rewords it to be from the viewpoint of an observer:

Sarah: ... when there's a change of light occurs () it turns [*Brings her left hand down.*] then goes forward [*Turns her left hand to the right and move forward.*] until another change of light.

Mario asks them if by using such a procedure, the turtle could find a light, and the children interpret this as asking what is the goal of the robot.

Mario: Do you think it could find a bright light using using a rule like that?

Sarah: No.

Meagan: I don't think that's what it was

Sargeant: It wasn't going towards a light or a darkness.

Meagan: And it wouldn't go.

Sargeant: It was going towards some type of lightness that was () that wasn't changing.

Sarah agrees with their evaluation of the robot's apparent goal and states what she believes if the rule that controls the robot's actions.

Sarah: I don't think it was actually going towards a light. I think if it sees a change, [*Brings her left hand down.*] it turns [*Turns her left hand to the right.*] and just goes straight [*Her left hand moves across the front of her body.*] () and then another change occurs [*Her right hand moves up and down in front of the left hand.*] () it turns again [*Her left hand turns to the right.*] and just goes keeps going straight.

But MatthewS has noticed something about the robot's actions during the manipulations that this rule doesn't cover.

MatthewS: No () cause when [*MatthewP*] put his finger on it [*the light sensor*] () and took it off [*He holds his left hand up close and brings his thumb down and then up*], it didn't change direction. But when he put his finger back on [*He brings his thumb down once more*], then it changed direction again.

This causes MatthewP to hypothesize that it is steps into darker areas that make the robot turn, and MatthewS agrees.

MatthewP: Maybe it it gets darker

MatthewS: Yeah.

MatthewP: That's true. Maybe if it gets darker, it turns.

After Sarah confirms that she and MatthewS have in mind the same description of the rule, the children seems all in agreement about the rules that govern the robot in this try. They have effectively gone from outside the machine to discover what is inside of it. Yet, as the next section will show, there is still more for them to discover about the robot's interactions with its environment.

Relating Observations to Programs

After the observation session, the children were able to look at the various programs that had controlled the robot during each of the trials. The programs were shown to the children, and the functioning of each one was described in turn. When the children saw the programs that had controlled the robots, they showed that they could *identify* emergent activity.

In one particular instance, they attributed previous observations of the robot circling under a light to the interactions of the light sensor, the light, and the program the robot was using at that time. First, the robot's program was described thus:

Mario: () it remembers what the brightness was the last time it looked () at the sensor. So it compares what the brightness was with what the brightness is now.

And if it used to be in brighter light, then it tries to find a brighter light now. Now in other words, if it went from bright to dark-er ()

And Sargeant jumps to the conclusion that the robot's action is going to be to directly move towards brighter areas.

Sargeant: So it's going to try to go back to the bright.

Mario: It's going to try to do something.

Sargeant: [*Correcting herself*] It's going to to find a brighter.

Meagan immediately connects this description of the program to what they observed earlier.

Meagan: That's what causes it to go in circles.

At first, Sarah thinks that Meagan is confusing the try being discussed with a different one where the robot turned repeatedly in place, but Sargeant clears up the misunderstanding by adding to Meagan's realization the element of the floor light that was used with the robot.

Sarah: No, no, no. No wait, wait, wait. No wait. Ugh. This is the second one that we saw.

Sargeant: I know. It would () Remember it would go under the light and go around and around, right?

Sarah: Right.

And the three of them complete the explanation that Meagan started.

Meagan: That's because it was looking for the light. [*unintelligible*]

Sargeant: But the light was right above it.

Sarah: And it can't see above. It can only see () ahead.

But Sarah is not sure that the robot can only see ahead. Mario clarifies the understanding of the sensor by specifying that it sees light in a cone, so that it can see somewhat above itself, but not directly above. This explanation could challenge the children's certainty about the model they just presented, but they use the extra information to elaborate their view.

Sarah: Especially since the eye was right here. *[She holds her left hand in front of herself to show the sensor at the front of the robot and puts her right hand above her left hand to show the light.]*

Sargeant: So it would move towards it, *[She moves her right hand in a straight line directly towards Sarah's right hand]* and then ()

Sarah: And then it would turn around looking for the brighter light so it couldn't find it () So then it would start like trying you know to go forward a little, remember?

However, this description is not correct because the program does not say that the robot should step forward a little. The robot steps forward because in the process of turning, its sensor goes from a darker area to one that is brighter or the same. The program only says that the robot should turn when light gets darker. Mario gives the children this description of what the robot does, and Matthews notices that there's no mention of stepping forward so he prompts for that element.

Mario: For seeker number 0, finding a brighter light was simply turning left. All it would do was turn left a little bit () turn left one eighth of a circle.

Matthews: And then keep going forward?

Mario: And then it would look at the light again. Was it brighter before than it is now?

But while they are aware of the discrepancy, they do not move to correct it, but express a desire to have more data.

Sargeant: Oh () We should have counted the times it was under the light and it turned in a circle. One, two, three, four, five, six

Sarah: Hmmm.

They are able to explain the dynamics they observed in terms of the elements of the robot's situation, but they are not able to use the finer details to complete their explanations.

Creating Emergence

Later, when the children were asked to create a program that could find a light, they designed a sequential program, a plan that would make the robot capture the light. In particular, they wanted to use the light sensor as a reliable source of information.

During the description of a program whose actions the children had found particularly frustrating to observe, MatthewP comments that that procedure is bad for finding lights because it wastes a lot of time looking around.

MatthewP: The only problem with that is if it was trying to find a rolling ball of light ()

Mario: Yes?

MatthewP: It would have () it would take it forever. It would never catch up with it.

Mario: Because?

MatthewP: Because it would () it takes the time to actually go around and check each time before going around again.

This prompts Mario to ask him what the computer should do instead. This makes MatthewP repeat the rule they had previously concluded the robot used to move about.

Mario: So you think () what does it have to do () in order to find the rolling ball of light?

MatthewP: Either go really fast or go keep turning until it finds a very () well a brighter [*unintelligible*].

Julie shows that she would like to have the turtle know directly what it is seeking in the environment by asking:

Julie: Tell it wha what how bright the what it's supposed to find is.

but this may be due to her having a model that the light sensor can report an absolute brightness. And Sarah would like the robot to take greater chances in finding the light by taking larger steps.

Sarah: And set off like you know how it says it turns around until it picks up the brightest spot? When it picks up the brightest spot, don't just ask go forward 2 steps. Go forward until it chan gets a different reading in light. *[Her right hand moves in a straight line away from her body.]*

Mario: Ok.

Sarah: And then it would be able to tell like you know like whether the ball had gone out of view and then it could you know *[Her right hand twists around in a circle]* pick it up.

Many of these descriptions are identical to those they just heard about the programs that controlled the robot they observed. In particular, while Sarah focussed in the preliminary session on how the light sensor worked, her comments imply that she is thinking of them as being like human eyes that can locate a light inside the field of view.

5.4 Learning About the Tools

Over the four following sessions, the children worked on learning how to use the tools at the same time as getting the robot to find the light. The children had to become comfortable using the ACT language before they could use it to create emergence in the robot. There were some features of the language and of how it was presented that caused problems in its appropriation.

Action Priority

One problem came from creating actions that would contend for the use of the motors. If two or more valid actions propose settings for a motor's POWER or DIRECTION, either

a choice is made between which of these action will take place, or the robot's actions will be unpredictable. The children were able to use the debugger to detect these situations and learned to use PREFER to choose between the conflicting actions.

PREFER as IF-THEN-ELSE

Another common source of problems in the children's ACT programs involved the use of the PREFER command which sets the priority of execution between two actions. MatthewP expected PREFER to act as a conventional IF-THEN-ELSE. A PREFER statement is always located inside an action and so has a condition for when it should be applied. In the case of PREFER as IF-THEN-ELSE, he expected that saying that one action should be preferred to another one would make the first action happen if the condition of the PREFER statement was TRUE, and not just give preference to doing it if it was already proposed. This showed up in particular with the use of actions without conditions where the children expected the PREFER to propose the alternative actions. When he saw this bug, MatthewP complained about the lack of an IF-THEN-ELSE in ACT, which supports the assertion that the some of the children's problem with PREFER was due to assimilating it with IF-THEN-ELSE. This interpretation is plausible in the light of the fact that most of the children had had previous experiences with programming languages such as Logo. Eventually, the children developed a style where the most basic actions were always proposed, and the actions of preferring were exhaustive and conditionalized.

Action Persistence

The children also had problems with correctly using the PERSIST primitive which sets the number of tenths of seconds for which a program will persist in doing the last selected set of actions. PERSIST was intended to provide an easy out to the problem of making the robot make a large turn after hitting an obstacle even though the sensor for detecting collisions would no longer be triggered almost as soon as the robot started

to backup. PERSIST was originally called REACT and was explained in terms of the robot's reactions to external conditions. While this is an accurate description of what PERSIST will do, it caused the children to always set the PERSIST time to 0 to get the machine to react quickly to changing conditions. This meant that their robot used a large number of small turns instead of a few large ones to get away from an obstacle. While Mario urged the children to put a PERSIST within their backup actions and even demonstrated this possibility once, this idea never caught on with them. However, while observing the backup actions of their contest robot in the rink, MatthewP remarked that it would have been important to have some persistence at those times.

5.5 Learning About Emergence

From the fifth session on, the children were often asked to explain what were the sources of the robot's knowledge about its environment. But while questions such "How does the robot know it has bumped into a wall?" are valid, their answer appears direct and trivial: the robot knows it has run into a wall because its touch sensor is triggered. However, the reliability of the answer depends upon the reliability of the touch sensor at reporting a contact, and the digital nature of the interaction: either the robot touches wall or it does not.

A more difficult question to answer was "How does the robot know it has captured the light?" This question came partly from the observations of the children's early desire to make the robot stop when it reached the light. The children first used a direct approach to answer to this question. The robot would know because they would encode the right level of brightness for this situation in its program. But this was not a viable answer because the light sensor was not a reliable source of information and could not be calibrated⁵. As a result, the children had to rely on emergent properties of the interaction of the robot with its environment to formulate a reliable and robust answer to the question. The

⁵It can be shown that if a calibrated light sensor had been used, robot behavior could have been used to calculate such things as the intensity of a sensed light source, its distance, and location.

children's final answer to this question was created in the context of solving the problem of having the robot act differently when it had caught the light. The example presented here was selected as a case of reasoning in emergent terms. We will look closely at how the children worked with this particular case.

The Parts for an Answer

As the children became comfortable with working with the development system, some interest in the work was lost. The introduction of the Robot Design Competition at the fifth session made them enthusiastic once again. The children realized that for this contest, it was important to have the robot act differently when it had caught the light. At this time, Mario asked the children if the robot could somehow know it had caught the light, something that the children themselves had asked in the beginning sessions of the workshop. The children attempted to encode in the robot a sensor reading corresponding to the light being in the robot's scoop. But they discovered that the sensor's operation made it impossible to find a reliable value that would correspond with that situation.

At the sixth session, the children are testing a robot that can capture a light in a hallway. They focus on the property that the light was caught against the wall.

MatthewP: Let it go Mario.

Mario: Ohhh I see. Cool.

MatthewS: Its goal is to trap it.

Mario: The moment it catches it, it goes off the wall.

Later in the same session, Sarah uses this property as part of the strategy being designed to catch the contest puck.

Sarah: Look, we get the beacon and we move to the wall and no one else can be it because we are there. [*Because the robot is in front of the contest puck or "beacon".*]

But a few minutes later, when the children are asked about whether the robot would know it's even touched the puck, they think that that's not possible.

Mario: How do we know whether we've touched the puck?

MatthewS: We don't.

Yet, at the end of the session, in talking about what to do in case their robot doesn't get to the wall first, Sarah continues with:

Sarah: Assuming we get the puck, we will have hit the wall. So let's say by 70 seconds we have yet to hit the wall, and sensor 2 [*the light sensor*] is not active, we go in a wild mad frenzy and attack everyone.

which holds the seeds for a way of making the robot know it's caught the puck.

The Emergence of an Answer

In the next session, session 7, the children experiment with encoding in the program a value that would allow the robot to know that it was roughly pointed towards the workshop's light while at a distance of at least six feet, the width of the rink in which the contest was to happen. They are no longer trying to find a setting that could be used to know the robot is in contact with the light, but rather are seeking an indicator for guiding the robot in the right direction. They found this without difficulty.

Once they are satisfied they have a program with which the robot can find a light, they begin a session to determine the strategy to be used in the contest. But Sarah points out a problem with how the robot will work in the contest.

Sarah: [*She is holding up one hand with two fingers in a "V".*] Here is the robot. [*Her other hand's index finger is acting as the light.*] Robot comes, robot pushes it [*the light*], it hits it to the wall, right? But you know what happens when our () when our robot does () when it hits the wall.

Julie: It backs up.

Sarah: It backs up. So it's going to back up and its going to go on, and we don't want it to do that.

Her hands are showing the robot turning away from the wall and taking the light with it. She probably means that she doesn't want the robot exposing the light to the other contest robots. A couple of minutes later Julie has a solution to this.

Julie: Mario, can we say if its touching an obstacle then it stays if it has here () if it has the light? Cause then it would hold the light in itself and would () nobody else would be able to get it.

Sarah has been trying to speak and she follows up on Julie's idea.

Sarah: We search, we find it, we go for it, we go pushing it forward. Then, look we encase this in the wall.

She picks up a robot and uses it to illustrate her point as she continues:

Sarah: So they can't get to it because we're blocking them. So this way only we can get it and we've touched it last. But the problem is when it hits the wall its going to go whirr whirr whirr [*She makes the robot turn with the puck.*] turn around. We don't want it to do that. But what we want it to do is if it doesn't have the sensor [*She means the puck.*] and it bams into the wall, we want it to turn around.

Mario: So how do we make a difference?

Sarah: We don't know.

An instant later she's struck by an insight. She wants to change the program to remove the constraint on the robot that backing away from an obstacle takes precedence over tracking the light.

Sarah: I've got it. I've got it. I've got it. No, no no. I've got it. We can put it in the program. If it sees the light, instead of having prefer ah () um back away from object over seeing the light, we have it prefer seeing the light over back away from

object. So that way, if it sees the light and we assume it's hit a wall then we can assume it's hit a wall and it's where we want it to be.

This takes the chance that the robot might get stuck running into a wall if it's seeing the puck but hasn't captured it. Mario will ask her about this possibility at the next session. At this point, the children turn to discussing back up plans in the case their does not catch the light first.

Twenty minutes later, Mario asks the children whether the robot can know it's gotten the puck. In doing this, Mario is speaking as an observer, discussing the knowledge as something the robot can be shown to possess. But the children return to the details of the robot's interactions to explain how the robot can appear to have this knowledge.

Mario: ... do we need to know we've gotten to the brick? [*By which Mario means the puck.*]

Sarah: Yes because ()

Mario: How do we know that?

Sarah: Because when we read the light, it goes forward. No, we don't have to. Look, what happens is we read the light. It goes forward, ok? Now that it's gone () ah () one it () it will go forward to a point then it will reach a wall. When it can see the light and it's reached the wall, then we know that () then it just keeps trying to go forward but it can't so it'll stay in one place and we'll have our barrier you know like I showed you.

Mario then paraphrases her sentence, completing the rule that she elaborated, and asks her to say what the robot would do once it has that knowledge.

Mario: Ok, so then the rule that you want is that if it's reached the wall and it still sees the light, that means it's got the light?

Sarah: Yes.

Mario: And using this rule, the robot knows it's caught the light?

Sarah: Yes.

Mario: So what does it do then?

Sarah: It keeps trying to go forward.

She has established a connection between the stimulus and how the robot reacts to it.

Implementing this Answer

In session 8, Sarah restates this connection but Mario challenges it by asking what happens if the robot is running into a wall and sees the light but hasn't captured it.. Sarah states that she is assuming that the condition of both seeing the light and running against a wall means the robot has caught the light. To support this assumption, she describes the worst case of what could happen when the robot is running into a wall and sees the light. She helps her explanation by drawing on a whiteboard.

Sarah: Here is our view [*Draws a quarter circle.*]

Mario: Yes.

Sarah: Ok? () Now if we're rammed into a wall () and our eye is up here [*She draws a dot near the wall.*] we can see this [*She draws a much smaller half circle within the large one.*] Ok?

Mario: Ok.

Sarah: Now we're going to assume we have our things out there [*She draws the robot's scoop arms inside the smaller half circle.*]

Mario: Ok.

Sarah: () Unless the puck () is like () there [*She draws a dot inside the smaller half circle but outside the scoop.*]() which () the chances of it being there are so very very slim.

Mario: Ok.

Sarah: We will not be able to see it unless it's it's we have it.

Sarah is showing her awareness that the effectiveness of the capture rule depends on the relationship between the active radius of the sensor and the position of the scoop's arms. As will be shown below, her awareness of this constraint will play an important role in the session after the contest when she will be asked to explain the robot's activity.

Later, Sarah changes the program the group has created to embody the capture rule. She changes the action:

```
action move-back-before-move-forward
  (prefer move-back move-forward)
  sensor-3?
```

by adding to the condition that makes `move-back` be preferred to `move-forward`:

```
action move-back-before-move-forward
  (prefer move-back move-forward)
  and sensor-3? not more level-2? 230
```

which makes `move-back` be preferred only when the robot is not following the light.

Accounting for the Activity

In the final session after the Robot Design Competition, the children were asked to reflect upon the dynamics that emerged in their robot's activity in the contest. Sarah focussed on the robot's difficulty at catching the puck in its scoop. During the contest, the robot

would move up until its scoop was to one side of the puck. Then, it would turn towards the puck, pushing it around much as a hockey player's stick pushes a real puck. Sarah came up with an explanation that drew on the details of robot's construction to account for the activity.

Sarah: They would see it and they'd go for it but the range [*of the sensor*] wasn't small enough to pick out where it was [*As Sarah speaks of the puck, her hands are on the sides of her head acting like blinders.*]

Sarah is ascribing the difficulty in part to the properties of the robot's sensor, and then she weaves that property into the activity.

Sargeant: [*Sarah asks Sargeant to act as the puck for a demonstration, and she places herself a couple of feet away with her arms held out in front of herself as if they were the scoop's sides.*] I can see Sargeant but if I go straight forward [*Sarah moves towards Sargeant*] I don't get it [*She comes up to one side of Sargeant*].

She has shown that because of its sensor arrangement in combination with the length of the scoop's sides, the robot can move up besides the puck while still acting as if the puck is directly in front of itself. Then, she explains how the robot pushes the puck along.

Sarah: So then I keep on trying to see her and I see her [*She turns to the right pushing Sargeant along.*] And I'm going for her but you see I'm not [*She stops talking and looks towards Mario.*]

Sarah has shown that she understands the robot's activity as the result of the *combination* of the angle of view of its sensor and the length of its scoop. The angle of view of the sensor causes the robot to act as if it is moving straight for the puck until it is very close to it. So close, in fact, that the robot's scoop is directly besides the puck when it falls out of the sensor's range. Then, the robot acts to bring the puck into view by turning towards it but this action only pushed the puck along. In other words, Sarah's explanation shows that she understands that the dynamic came not only from what was directly programmed in, but emerged from the program's combination with the properties of its sensor and scoop.

Summary

The children's answer to how the robot would know it had captured the light ranged over time going from a simple "I don't know" until the answer they finally used for the 6.270 robot. This answer uses emergent property of interaction between the robot, its program, the light, and the surrounding world. The children formulated it as a rule much like "the robot knows it has captured the light when it can still sense the light but it has run into a wall." If the robot sees the light while it's bumping into a wall, it must mean that the light is caught between it and the wall: that the robot has captured the light. In formulating such rules, the children viewed the robot's knowledge as emerging from its interactions with its environment.

The children used this answer to change the program they had developed for finding the light. They conditionalized the robot's preference for backing up from obstacles to make it inactive in its usual triggering situation if the robot can still see the light. They did not add an element of control state to the robot's program that would encode that it had captured the light, and this kind of change could allow the robot to react flexibly in case some other robot would take the captured light away.

Chapter 6

Conclusions and Future Work

The thesis up to now will be summarized. The micro-genetic pattern followed by the children as they evolved an emergent solution will be reviewed and recommendations will be made for further studies. The children's difficulties in using the materials will be examined and appropriate solutions will be proposed. Finally, different contexts and materials in which the results of this study could be reproduced will be suggested.

6.1 Summary of the Thesis

This thesis has shown the micro-genesis of an emergent solution by a group of children. To arrive at this solution, the children had to understand the dynamics of a system made of a robot, its program, and its environment. They understood the functioning of electronic and software parts that were provided to them, built a new program, observed how it interacted with its environment, and modified their construction to take advantage of the interaction.

There were two ideas that informed this study. The first idea is the emergence from simple parts of greater patterns and the second idea is learning through building. While the emergence is being invoked to explain many important phenomena, we are only beginning to develop formal ways of understanding it. It was argued that in order to

develop a better means of understanding emergence, it is important to see how people already think about such phenomena. Five examples of emergence were given, and some of the features they have in common were described. It was argued that these systems lack a central control structure to account for the phenomenon being exhibited, and that this emergent activity exists only at a high level of description of the system and is absent from the lower levels describing the parts of which the system was seen to be made. The views of Turkle, Hillis, and Hofstadter on the possibility that intelligence is emergent from a large number of simple parts were discussed, and the difficulty in understanding this kind of emergence, emergence in self-organizing systems, was emphasized. In contrast with this kind of emergence, that found in systems of agents situated in worlds as those studied by Agre were shown to be simpler and, it was argued, easier to understand. However, it was shown that they had many of the essential characteristics that identify self-organizing systems as emergent. It was argued that in order to understand how emergent phenomena happens, it is necessary to make a comparison between levels of description, and that this is best done with the kind of emergence studied by Agre.

The basic tenet of the Constructionist approach to education was presented, and it was argued that it is synergistic with the approach to understanding emergence suggested by Agre. The concept of a tool kit for exploring emergent phenomena in agent-world interaction was introduced. The idea of “appropriation” and of the development of a community of ideas was presented, and it was shown how these argued for using the tool kit in a workshop environment. The importance to the workshop participants of doing something truly new was stressed.

A tool kit being developed by Resnick for exploring self-organizing systems was sketched and some of the goals that Resnick intends to achieve with this tool kit were presented.

A mini-study of adults observing agents situated in worlds was presented, and it was argued that these adults had a tendency to progress from local to global descriptions of the agent’s activity. An example was given of an emergent phenomenon identified by

some of the observers as being built in.

The materials of the workshop were described in detail. The LEGO robots and their sensors were described, as was the ACT environment for programming it. Five alternative systems for building robots were described and contrasted. The way the participants of the workshop were chosen and the events of the workshop were sketched. Then, a sketch of each of the workshop's nine sessions was given.

The direction for analysis of transcripts from the workshop were given, and the research questions were presented. The transcripts showed how the author presented a group of children with the idea of making a robot capture and release a light. The children were given a robot equipped with a scoop and sensors, an initial program, and a tool for writing new programs. While the children had some ideas of how to go about making a robot capture a light, they did not fully understand the constraints of the problem at hand. The children's process of determining why a robot acted in a particular way was shown. While they could partly understand how the behavior they had observed emerged from the interactions of the underlying parts, their initial reaction to making a robot find a light showed they thought its activity should be directly programmed in. Some of the children's milestones in learning how to use the tools of the workshop were shown.

The children were shown thinking about the question of how the robot could know it had captured the light in its scoop. Their first answer was to encode into the robot's program a light level corresponding to the distance of the robot from the light, but the light sensor's unreliability made this not possible. Later, in working with the robot, they noticed that it could capture the light against a wall but that it would then move away from it. When the goal of the workshop changed to making a robot win the MIT Robot Design Competition, the goal in designing the robot became how to protect the light from their opponents. The children then realized that what they needed to do is to change the robot's reactions so that it would not back up if the light is between it and the wall thereby isolating it from the other contestants. The children knew what kinds

of problems could arise from this approach but they believed it to be their best chance to win the contest. It was shown that if the robot was built this way, it would be as if it could know it had captured the light. The children's reflections on how their robot performed in the contest were used to argue that they could now fully relate the dynamics of the robot's activity to the relationship between its parts and its environment.

6.2 Children's Thinking About Emergence

The children's thinking about emergent phenomena falls into at least two categories: recognition and construction. In the first category, the children are presented with a phenomenon and they must relate it to the underlying elements that are described. In the second category, the children are presented with a set of parts and are asked to make them exhibit a particular effect.

Recognizing Emergent Phenomena

The transcripts showed in three cases where the children analyzed a phenomenon emerging from a substrate. This is not to say that they called this phenomenon emergent, but rather that they were able to relate two levels of description of the same events.

During the first session, the children observed a pattern of motion in robot's light seeking behavior. When the robot was interacting with a floor lamp that rested above its light sensor, its action was described by the children as circling under the light. Later, the children were given a low-level description of what controlled the robot's actions: that the robot would turn when a motion causes it to go from a bright area to a less bright one. They used this information in combination with their knowledge of how the robot's sensor operated and of the robot's situation to explain how the pattern in the robot's motion would arise.

While they were seeking to make the robot know it had captured the light in its scoop, they observed that the robot could push the light into a wall and then move back.

They were able to modify the robot's program so that it would not move back in but only in that particular situation. This showed that they understood what elements from the situation combined to give rise to the activity.

Finally, during the contest, they observed the robot sliding up to the puck. They were able to describe the robot's actions in terms of its program and ascribed the robot's activity to the interactions of its program, its sensor's angle of view, and the length of its scoop.

Creating Emergent Phenomena

In the workshop, the children's first reaction when faced with the problem of achieving an effect was to attempt to produce it through direct means. Only after they had decided that such an approach would not work would they turn to emergent phenomena as a source of solutions.

The direct approach stands in sharp contrast with the light-seeking feedback system that they encountered in the first session. That program used the difference between the currently sensed light intensity and the intensity measured at the robot's previous position as a means of deciding whether to seek a new heading; getting to the light was an emergent property of this process. Yet, even though they had seen this process in action and they knew how it had been implemented, the children's design of a light-seeking robot was to make it turn towards a light of a certain brightness and then move in its direction.

This tendency to use a direct approach was evidenced in the children's answer to the question of whether the robot could know it has captured a light. They wanted to tell the robot the intensity of the light it would register when it was close to the light. But when the children attempted to use this direct method, they saw that the sensor's way of working made the method unusable. Only then did the children resort to using the emergent properties of the system to solve the problem.

The way they did this is best described as opportunistic: the children took advantage

of a dynamic they encountered accidentally in the process of observing their program in action. The children first observed the dynamic of the robot pushing the light against the wall. Later, they reformulated the problem from capturing the light to keeping it against the wall. Only after having done this did they understand that the dynamic could be modified into a solution to their problem. Then, they were able to use their understanding of the system to make the needed modification.

Summary

In the beginning of the workshop, the children showed that they were able to relate levels of description of an activity to identify emergent phenomena. This ability remained unchanged throughout the workshop. But in the beginning, the children were not able to create emergent phenomena, instead seeking direct methods through which they could achieve their goals. Only after having worked with the robot for many sessions and having failed at using direct methods to achieve an effect did they turn to using emergent phenomena to provide a solution.

There is no indication in the transcripts that the children were aware of engaging in this process. And, because of its opportunistic nature, it seems unlikely that having this awareness would improve their ability to take advantage of the dynamics of the situation because which dynamics are seen will depend upon elements unique to the each situation. There are no constants in this process that the children can learn to seek out; there can only be a general knowledge that something might be found. However, Agre provides some indication that an improvement in the children's rapidity in seeking out and modifying the dynamics of the interaction might result from an awareness that this process is happening. In his study of the evolution of routines in the daily activity of persons, he found that people's awareness that they were engaging in an evolving routine would speed up the improvement of the routine [Agre 1985]. This suggests that in a second phase of a project such as the one discussed in this thesis, the children should become aware of the process through which they solved the problem before proceeding

to solving another such problem.

6.3 Improving the Materials

The Hardware

The most frustrating aspect of the project were the difficulties with working with real hardware instead of more reliable software components. In particular, the LEGO materials made robots that broke often, the sensors were noisy, and the Programmable Bricks were easy to burn out. Of course all of these troubles were expected, and some of them, such as noisy sensors and fragile robots, were the welcome sign of a real world experiment.

Nonetheless, while the children were understanding of the experimental nature of the materials available during the workshop, they have a right to expect that future Programmable Bricks and their sensors will be more robust. Fortunately the Brick's creators have gone through a second cycle of design and have assembled a much improved portable computer with more sensor capabilities. This new machine was designed for the Robot Design Competition and has eight digital ports and four analog ports. This last capability should allow more reliable reports from analog light sensors because measuring their state will not be subject to the software interactions that caused variation on the Programmable Brick.

Unfortunately, little can be done to improve the reliability of the LEGO materials. Their weaknesses comes hand in hand with their approachability by children. While it is certain that with more work, more reliable machine designs can be created, quite a bit of time can be spent on making reliable a machine that will only be used for a short while.

In order to understand the capabilities of the sensors, the children had to rely on the metaphors provided by the author or to create their own and relate them to their experience using the robots. For example, when the children first encountered the turtle, it took a while for the limits of its light sensor to be understood. The children readily understood the picture of the sensor's field of vision as being a "cone a third of a circle

wide,” but, as was shown in Section 5.3, they acted for a while as if the robot could localize the brightest light within this cone. If the sensor could have been hooked up to a screen whose overall brightness would vary according to the intensity of the light reported by the sensor, the capabilities of the sensor might have been understood more rapidly. It seems natural to think that being able to sense the world as the robot senses it would make it easier to develop an understanding of what triggers the robot’s actions. More will be said about this while discussing improvements to the program debugging tools.

Many researchers have maintained that the interestingness of the tasks that can be undertaken in a robotics enterprise is tied to the richness of the information that can be gleaned from the sensors ([Brooks and Flynn 1989], [Chapman 1990], [Materic, 1990]). For example, in commenting upon an early description of this project, Chapman stated:

I think that your major stumbling block will be in finding adequate sensors. That’s been the principal issue in all the mobile robot projects I know of. Unless you have pretty high quality sensing, there’s not much interesting you can make an embedded device DO.

Chapman, personal communication

While this cannot be denied, the work the children did with the LEGO robots showed that an understanding the robots’ situation can go a long way towards making their activity interesting. Furthermore, arriving at this understanding is at the center of the kind of activity that this thesis explored and there is no substitute for it. Once again, more will be said about this while discussing improvements to the program debugging tools.

The Software

The basic editor functions such as inserting and deleting text, moving around a program with the arrow keys, and naming, saving and deleting pages were readily mastered by the

children. However, they were never comfortable with using the more advanced functions such as text block movement with cut, copy, and paste operations, and interactive text search and substitution operations. Certainly, having the most of the advanced functions on hot-keys did not help their accessibility or help make the children aware that they were available. Although in the short time of the workshop the basic functions were sufficient for the children's purposes, having a direct-manipulation environment like that of the Macintosh might have made a difference in this respect.

PERSIST was a difficult primitive for the children to use, and in the end they chose to ignore it. It was created because it provided a simple solution to two problems: backing up from an obstacle and achieving quiescence in the digital circuit. Clearly, the latter use was not a good idea, and better code generation would have allowed determining quiescence without it. But also, PERSIST was a bad tool for controlling actions because of its lack of granularity: the current time setting of PERSIST applied to all valid actions. A better solution would be to have PERSIST be a statement that applied only to actions that contained it. However, even if PERSIST had not existed, its effects could have been achieved by creating a sequence of actions, so it was not really required.

Many times during the workshop, the children created programs with actions that contended for the control of the same effector. This was amongst the most difficult bugs for them to detect. But in all of those situations it was possible to find the source of the contention by using the debugger to trace selected actions. While it might seem possible to detect possible contentions for actions while compiling the program, only those contentions between actions with commensurable valid conditions can be detected correctly. For example in Section 3.2.3, the contention between BACKWARD and FORWARD could have been detected because the latter is always valid while the former will be valid only when the bumper sensor is triggered. The case of two actions that contend because their stated activation conditions effectively overlap without involving a logical combination of the same predicates cannot be detected as faulty. For example, contention between an action that is valid when the bumper is triggered and an action that depends

on the light sensor reporting a value above a particular intensity cannot be guaranteed to happen. At best, the user can be advised of actions that use the same effectors, but this might result in a large number of messages warning about conditions that will never occur. Chapman has reported that his arbitration macrology detects contention problems as the program is acting and allows the user to trace their source [Chapman 1990]. It is possible to implement this feature in the ACT system but it would complicate its use. The difficulties of dealing with this complexity was weighed against the difficulty of finding arbitration bugs in programs¹, and it was chosen to leave this kind of error for the user to discover by other means.

One important improvement to the software might be to allow debugging the robot's activities using data obtained from a run. Gathering the information for this kind of debugging requires that the robot's sensor inputs be stored for later use, at which time the data is to be interpreted in terms of its effects on the robot's activity. It can be useful to feed back these inputs to the robot in a controlled way and observe its program's operation. But because the robot's sensor data comes from its interactions with its environment, this method will only be useful to explain the robot's activity if the data can be related to features of the environment. Videotapes of the robot in action can help the designer make these relations, but videotape analysis is a difficult task. Another possibility is to feed the sensor information into transducers that would allow the children to view the world as the robot saw it. This might make it easier for the children to use the videotapes to reflect upon the connections between the robot's activities and its interactions with its environment.

¹Chapman's program is more than a hundred times the size of the children's programs

6.4 Future Explorations

A More Open-Ended Activity

Because of the eight week limit on the amount of time available for the workshop, the children had been provided with a goal to be achieved. This goal served as a challenge and a milestone that informed much of the structure that emerged from the workshop. The author's belief that this goal had yet to be accomplished, and its apparent importance fostered the children's appropriation of it. Yet, more open-ended activity than what was used for this workshop would be better adapted to the Constructionist view of education.

The goal of the workshop was not appropriated equally by all the children. For example, one of the children wanted the robot to use a sound sensor instead of a light sensor, but while such a sensor was available at that time, the child felt that he did not have enough time to learn to use it and to develop his own activities around it. Had the workshop had much more time allocated to it, or better yet had no time limits imposed on it, that child would have been able to explore the possible uses of the sound sensor and decide for himself whether that was a fruitful area for discovery. As it was, time made that choice for him.

The workshop began with an introductory session where the author spoke a lot at the children, and the introduction of new materials usually involved a twenty-minute session where their capabilities were sketched. It is important to stress that this activity did not make the workshop any less Constructionist. The purpose of the introduction session was to communicate to the children the range of ideas, activities, and materials that they would encounter during the workshop. It was not expected that through this session, the children would gain an understanding of the ideas or activities that had been imagined they might encounter or in which they might engage during the workshop. It was considered more important to get the materials into the children's hands, to provide them with some experiences through which they could begin to build their own picture of the goals of the workshop, and to establish the beginnings of a relationship with the

author. The goals of the sessions introducing the new materials were similar to those of the first session.

Different Kinds of Emergence

It is important to relate the results obtained in this study to the possible study of other kinds of emergence. In Section 2.3, Resnick's proposal for the study of emergence in self-organizing systems was introduced. It is not clear how the results presented in this thesis can be used to help Resnick's future work with self-organizing emergence. One possible approach would be to reduce the size of the systems with which he will be working to simplify them, but not all kinds of simplification will be of help. For example, reducing the size of the matrix in the Life Game (a cellular automata that gives rise to many different structures,) until it had only tens of elements instead of hundreds would certainly make it simpler but only at the cost of removing the emergent phenomena that makes it interesting. However, slowly varying the size of the matrix until a particular phenomenon begins to happen might yield important clues to how the structures emerge.

The best help this study can provide to Resnick is in its pointing out the opportunistic character of the activity in which the children engaged to develop the emergent phenomenon. The opportunities arose only because the children were able to work freely and extensively with the materials. By choosing particular goals and seeking out related phenomena, the children should be able to eventually negotiate with the system to obtain the results they seek.

Bibliography

- [Abelson and diSessa 1980] Abelson, H. & diSessa, A. (1980). *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* Cambridge, MA: MIT Press.
- [Agre 1988] Agre, P. E. (1988). *The Dynamic Structure of Everyday Life* Cambridge, MA: MIT AI Lab TR #1085.
- [Agre 1985] Agre, P. E. (1985). *Routines* Cambridge, MA: MIT AI Lab Memo #828.
- [Bourgoin 1990] Bourgoin, M. (1990). Children Using LEGO Robots to Explore Dynamics in *Constructionist Learning* Harel, I. ed. Cambridge, MA: The MIT Media Laboratory.
- [Brooks and Flynn 1989] Brooks, R. A. & Flynn, A. M. *Fast, Cheap and Out of Control* Cambridge, MA: MIT AI Lab Memo #1182
- [Brooks 1986] Brooks, R. A. (1986). *Achieving Artificial Intelligence Through Building Robots* Cambridge, MA: MIT AI Lab Memo #899
- [Chapman 1990] Chapman, David (1990). *Vision, Instruction, and Action* Cambridge, MA: MIT AI Lab TR #1204.
- [Farber 1990] Farber, N. G. (1990). Puzzled Minds and Weird Creatures: Spontaneous Inquiry and Phases in Knowledge Construction in *Constructionist Learning* Harel, I. ed. Cambridge, MA: The MIT Media Laboratory.
- [Hillis 1988] Hillis, D. (1988) *Intelligence as an Emergent Behavior; or, The Songs of Eden*. in the Winter 1988 issue of *Daedalus*.
- [Hofstadter 1985] Hofstadter, D. R. (1985) Waking Up From the Boolean Dream, or, Subcognition as Computation in *Metamagical Themas*. New York: Basic Books.

- [Martin 1988] Martin, F. G. (1988). *Children, Cybernetics, and Programmable Turtles* Unpublished Masters Thesis, Cambridge, MA: The MIT Media Laboratory. A condensed version of this thesis can be found in *Constructionist Learning* Harel, I. ed. Cambridge, MA: The MIT Media Laboratory.
- [Materic, 1990] Materic, M. J. (1990). *A Distributed Model for Mobile Robot Environment Learning and Navigation* Cambridge, MA: MIT AI Lab TR #1228.
- [Papert 1986] Papert, S. A. (1986). *Constructionism: A New Opportunity for Elementary Science Education* Cambridge, MA: Proposal to the National Science Foundation.
- [Resnick 1988] Resnick, M. (1988). *MultiLogo: A Study of Children and Concurrent Programming* Unpublished Masters Thesis, Cambridge, MA: Department of Electrical Engineering. A condensed version of this thesis can be found in *Constructionist Learning* Harel, I. ed. Cambridge, MA: The MIT Media Laboratory.
- [Resnick 1989] Resnick, M. (1989). *A Computational Environment for Exploring Self-Organizing Behavior* Unpublished Thesis Proposal, Cambridge, MA: Department of Electrical Engineering.
- [Suchman 1987] Suchman, L. A. (1987). *Plans and Situated Actions* New York, NY: Cambridge University Press.
- [Travers 1986] Travers, M. D. (1986). *Agar: An Animal Construction Kit* Unpublished Masters Thesis, Cambridge, MA: The MIT Media Laboratory.
- [Turkle 1984] Turkle, S. (1984). *The Second Self: Computers and the Human Spirit*. New York: Simon and Schuster.
- [Wallace and Grimm 1984] Wallace, M. and Grimm, L. (1984). *Robot Odyssey I* Menlo Park, CA: The Learning Company.

Appendix A

Resources Used in this Study

Quantity	Equipment
4	Apple II GS with at least 1 Megabyte of RAM.
1	High-speed shutter videotape recorder.
1	Freeze-frame videotape player.
10	Sets of wheels, axles, motors, switches.
3	Sets of P-Brick, light and sound sensors, binary sensor multiplexor, light and sound sources.
1	Ball of light.
	Miscellaneous LEGO materials (bricks, beams).
	GSLisp and related software.

Appendix B

Example Programs

This section has the text of all the programs that are referenced in this document. The ACT programs have been altered to reflect the state of the ACT language as presented here.

B.1 Sample Light Seeking Programs

Below are three implementation of the same program, first in Logo, then in ACT, and finally directly as a logic circuit. This program causes a robot to act as a light-seeking machine that is much like those observed by the children in the workshop's first session.

Logo Version of Procedure

```
TO SEEKER
SEEKERLOOP 255 BRIGHTEST
END
```

```
TO SEEKERLOOP :OLD.BRIGHTNESS :BRIGHTNESS
IFELSE TOUCHING?
  [BACK 10 LEFT 8]
  [IFELSE BRIGHTER? :OLD.BRIGHTNESS :BRIGHTNESS
    [LEFT 8]
    [FORWARD 5]]
SEEKERLOOP :BRIGHTNESS BRIGHTNESS
END
```

```
TO FORWARD :TIME
TALKTO [A B]
SETEVEN
```

```
SETPower 7
ONFOR :TIME
END
```

```
TO BACK :TIME
TALKTO [A B]
SETODD
SETPower 7
ONFOR :TIME
END
```

```
TO LEFT :TIME
TALKTO "A
SETODD
TALKTO "B
SETEVEN
TALKTO [A B]
SETPower 7
ONFOR :TIME
END
```

```
TO TOUCHING?
OUTPUT SENSOR 3
END
```

```
TO BRIGHTNESS
OUTPUT LEVEL 2
END
```

```
TO BRIGHTER? :BRIGHTNESS1 :BRIGHTNESS2
OUTPUT (:BRIGHTNESS1 - :BRIGHTNESS2) > 2
END
```

ACT Version of Procedure

```
action back-away-before-seek ; When having a choice between
  (prefer back-away-from-the-obstacle ; backing up from an obstacle and
    look-for-a-bright-light ; any other action, back away.
  prefer back-away-from-the-obstacle
    go-towards-the-bright-light)
always
```

```

action back-away-from-the-obstacle      ; Back away from an obstacle by
  (propose move-a direction: odd power: 7 ; spinning both motors backwards
    propose move-b direction: odd power: 3); only making one go slower
touching-obstacle?                      ; than the other.

action look-for-a-bright-light           ; Look for a new source of
  (propose move-a direction: odd power: 7 ; light by spinning both
    propose move-b direction: even power: 7); motors in opposite
lost-the-bright-light?                   ; directions.

action go-towards-the-bright-light       ; Move towards a source of
  (propose move-a direction: even power: 7 ; light by spinning both
    propose move-b direction: even power: 7); motors forward with equal
not lost-the-bright-light?               ; power.

action lost-the-bright-light             ; Report that the source of
  (report more 2                          ; light was lost when the old
    subtract old-brightness?              ; sensor reading was more than
      brightness?)                       ; the current on by 2 units.
always

action old-brightness                    ; Report the previous value of
  (report remember brightness?)           ; the light sensor.
always

action brightness                        ; Report the current value of
  (report level-2?)                       ; the light sensor.
always

action touching-obstacle                 ; Report the current state of
  (report sensor-3?)                      ; the front bumper.
always

action move-a direction power            ; Move the left motor by
  (propose power-a power: power?          ; setting port A's power and
    propose spin-a direction: direction?) ; spin direction.

action move-b direction power            ; Move the right motor by
  (propose power-b power: power?          ; setting port B's power and
    propose spin-b direction: direction?) ; spin direction.

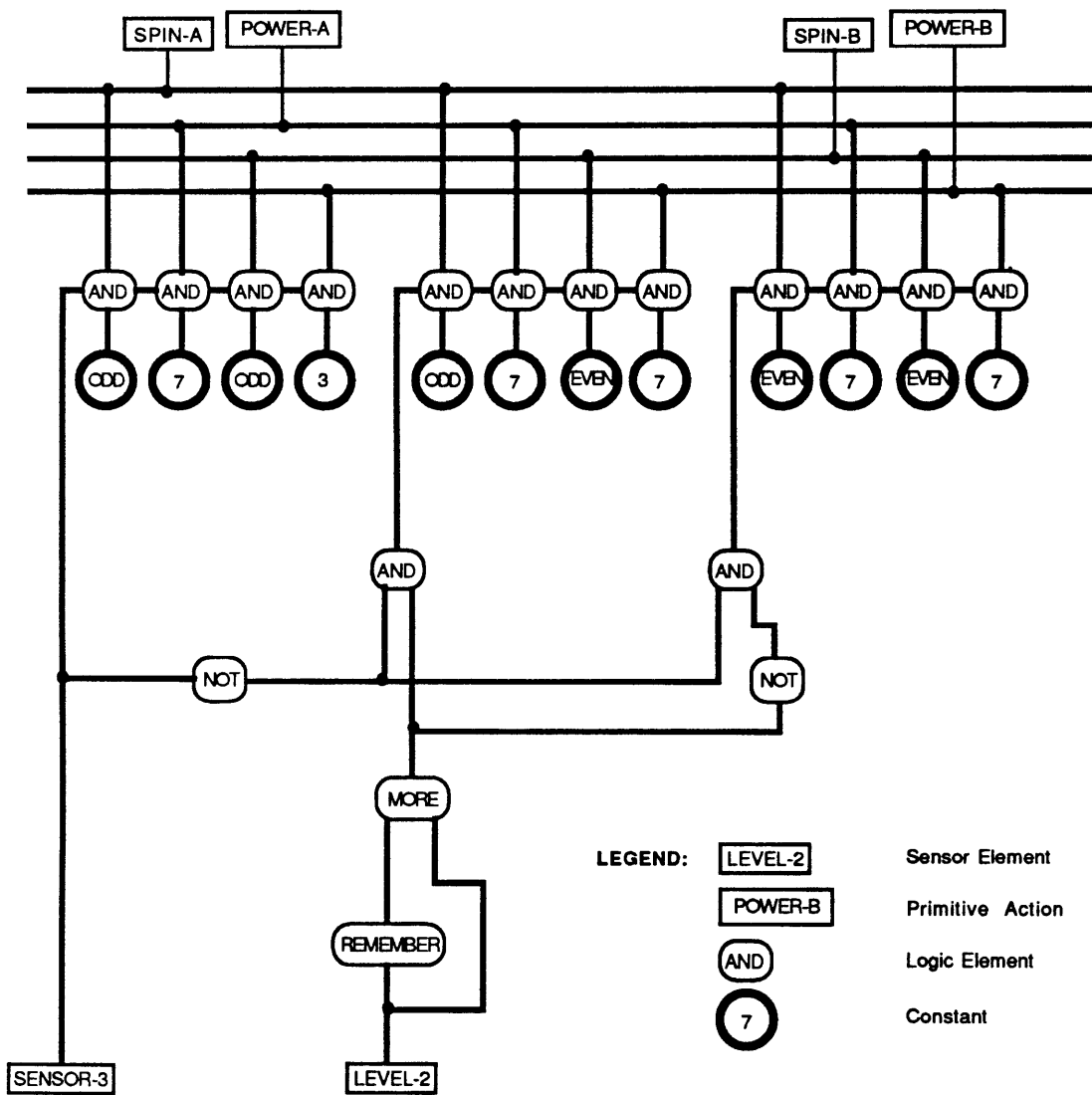
```

```

action live
  (propose persist time: 1           ; Set a tenth-second persistence.
   propose sensor-3                  ; Enable the sensor reports.
   propose level-2)
always

```

Logic Circuit Version of Procedure



B.2 MOVEAROUND

MOVEAROUND is the ACT program that was given to all the children at the beginning of the workshop. It is described in Section 3.2.4.

```
action back-away-before-move
  (prefer back-away-from-obstacle
   move-forward)
  always
```

```
action back-away-from-obstacle
  (propose move-a direction: odd power: 7
   propose move-b direction: odd power: 3)
  touching-obstacle?
```

```
action move-forward
  (propose move-a direction: even power: 7
   propose move-b direction: even power: 7)
  always
```

```
action touching-obstacle
  (report sensor-3?)
  always
```

```
action brightness
  (report level-2?)
  always
```

```
action move-a direction power
  (propose power-a power: power?
   propose spin-a direction: direction?)
```

```
action move-b direction power
  (propose power-b power: power?
   propose spin-b direction: direction?)
```

```
action live
  (propose react time: 1
   propose sensor-3
   propose level-2)
  always
```

B.3 The SMART Robot's Program

This ACT program was written by the children to control their entry in the Robot Design Competition. It was hand-translated by the author into 68H11 assembly language.

```
action move-forward
(propose move-a direction: even power: 7
 propose move-b direction: even power: 7)
always
```

```
action turn
(propose move-a direction: even power: 0
 propose move-b direction: even power: 7)
always
```

```
action move-back
(propose move-a direction: odd power: 7
 propose move-b direction: odd power: 1)
always
```

```
action move-forward-before-turn
(prefer move-forward turn)
more level-2? 230
```

```
action turn-before-move-forward
(prefer turn move-forward)
less level-2? 231
```

```
action move-back-before-move-forward
(prefer move-back move-forward)
and sensor-3? not more level-2? 230
```

```
action move-back-before-turn
(prefer move-back turn)
sensor-3?
```

```
action move-a direction power
(propose power-a power: power?
 propose spin-a direction: direction?)
```

```
action move-b direction power
(propose power-b power: power?)
```

```
propose spin-b direction: direction?)
```

```
action live
```

```
(propose react time: 0
```

```
  propose sensor-3
```

```
  propose level-2)
```

```
always
```

Appendix C

ACT Language Reference

C.1 Basic Language Structure

An ACT program is composed exclusively of ACTION definitions. This statement is the basic abstraction mechanism of the ACT language.

ACTION *name* <*parameters*>
(*actions*)
predicate

Action *name* is proposed when *predicate* is satisfied and no other action opposes it. In turn, *name* proposes the actions that compose it. If *predicate* is absent, a NEVER is assumed to be intended.

C.2 Primitive Action Statements

Each action that is part of an ACTION's actions, must be in the form of one of these statements. For convenience, if an action name isn't preceded by one of these words, it's assumed that it is meant to be preceded by a PROPOSE.

PROPOSE *action* <*settings*>

Propose that *action* be done with its parameters set to *settings*.

SET *action* <*settings*>

If *action* is to be done, it should be with its parameters set to *settings*.

OPPOSE *action*

Action should not be done.

PREFER *preferred overridden*

If both *preferred* and *overridden* are proposed, oppose *overridden*.

REPORT *value*

Set the value reported by the action containing the **REPORT** statement to *value*, when the action is done. If the action isn't being done, or if it doesn't have a **REPORT** statement, it reports zero.

C.3 Primitive Actions

These are the simplest actions found in the version of ACT language used by the children. Versions of the ACT language can contain different sets of primitive actions depending on the details of the machine they are meant to control.

PERSIST *time*

Set for how long the currently active actions are to remain active.

SENSOR-*N*

Reports as a predicate the state of a signal at input port *N*.

LEVEL-*N*

Reports as a numerical value the duty cycle at input port *N*.

POWER-*N* *power*

Sets the duty cycle of the signal output at port *N*.

SPIN-*N* *direction*

Sets the direction of the signal output at port *N*.

C.4 Primitive Operations

These operations can be used to construct complex conditions for proposing actions, or to compute the value that an action is to report.

NOT *predicate*

Reports the logical negation of the predicate.

BOTH *predicate1 predicate2*

Reports the logical conjunction of the predicates.

EITHER *predicate1 predicate2*

Reports the logical disjunction of the predicates.

MORE *value1 value2*

Reports whether *value1* is numerically greater than *value2*.

LESS *value1 value2*

Reports whether *value1* is numerically less than *value2*.

SAME *value1 value2*

Reports whether *value1* is numerically equal to *value2*.

ADD *value1 value2*

Reports the sum of the values.

SUBTRACT *value1 value2*

Reports the difference of the values.

REMEMBER *value*

Report next time the value being remembered now.

C.5 Special Words

These constants are defined to facilitate the reading of ACT programs: **ALWAYS**, **NEVER**, **TRUE**, **FALSE**, **ODD**, **EVEN**.

Appendix D

The ACT Keyboard Hotkeys

The ACT editor is modeled after the Apple II version of LogoWriter 2.0. However, since ACT is a compiled language, some keys were added to control the compilation and debugging of programs.

D.1 Editor Command Keys

Note: Having **Caps-Lock** down is the same as holding the **Shift** key down when the letter and arrow keys are being used.

LogoWriter-like Keys

Escape	Save the current buffer to a file, and show a menu of the files on the disk.
Apple-1	Select
Apple-2	Cut
Apple-3	Copy
Apple-4	Paste
Apple-6	Cut to End of Line
Apple-u	Move Up to the Edit Buffer.
Apple-d	Move Down to the Command Center.
→	Move Right

←	Move Left
↑	Move Up
↓	Move Down
Apple-→	Next Screen
Apple-←	Previous Screen
Apple-↑	Top of Buffer
Apple-↓	Bottom of Buffer
Apple-b	Beginning of Line
Apple-e	End of Line
Apple-o	Open a New Line

New Keys

Apple-Shift-u	Widen the Edit Buffer window.
Apple-Shift-d	Widen the Command Center window.
Apple-f	Find a string.
Apple-Shift-f	Find the same string again.
Apple-r	Replace one string with another, asking for a yes. Both Y and Space mean yes, and Escape means stop.
Apple-Shift-r	Replace one string with another without asking.
Apple-Shift-l	Load a file into the Edit Buffer.
Apple-Shift-n	Name the Edit Buffer.
Apple-Shift-s	Save the Edit Buffer to a file.
Apple-Shift-w	Wipe a file from the disk.
Apple-c	Clear the Command Center

D.2 Compiler and Debugger Keys

Apple-Control-a	Launch the program currently residing in the Programmable Brick.
Apple-Control-c	Compile the current buffer.
Apple-Control-d	Send the compiled program to the Programmable Brick.
Apple-Control-s	Single step the program currently residing in the Programmable Brick.
Apple-Control-w	Ask for the name of an action in the current program and set a watch point on it.
Apple-Control-q	Ask for the name of an action in the current program and remove the watch point that is on it.

Appendix E

The ACT Compiler

The ACT compiler was written in GSLisp, a dynamically scoped Lisp interpreter. Nonetheless, the code was adapted so that it could work in a lexically scoped environment.

E.1 The Parser

For simplicity's sake, the lexical analyzer is that of GSLisp, so ACT's symbols and numbers are equivalent to GSLisp symbols and numbers. The grammar for the ACT language is:

```
program    → action program |  $\epsilon$ 
action    → action aname parameters (body) expression
aname     → symbol
parameters → pname parameters |  $\epsilon$ 
pname     → symbol
body      → statement body |  $\epsilon$ 
statement → aname settings
              | propose aname settings
              | set aname settings
              | oppose aname
              | prefer aname aname
              | report expression
settings  → pname: expression settings |  $\epsilon$ 
expression → constant
              | pname?
```

```

| aname?
| active aname
| operation
operation → not expression
| and expression expression
| or expression expression
| more expression expression
| less expression expression
| same expression expression
| both expression expression
| either expression expression
| add expression expression
| subtract expression expression
| remember expression

constant → number | always | never | true | false | odd | even

```

E.2 Intermediate Representation

In their intermediate representations, actions are sets of named expressions with latches. Each name corresponds to one of the parameters of the action, with each action possessing at least two parameters: **active** and **report**. **Active** names the expression that computes whether the action is valid for this cycle. **Report** names the expression that computes the value reported by the action on this cycle. This expression is anded with the value of the **active** expression so that an action reports 0 when it is not valid.

E.3 Run-Time Environment

The ACT language is made to produce a digital circuit with timing elements and latches from a program specification. Unfortunately, this project did not have access to the current programmable gate array technology. Therefore, the functioning of an ACT circuit had to be simulated on a serial processor, a 65C02 with 32K of RAM to be exact.

The system's software is separated into three parts. The first part, the updating mechanism, transforms the sensor signals into input values and output values into control signals, the second part represents the combinatorial logic elements, and the last part represents the latches. The updating mechanism is driven by a regular system interrupt to accurately maintain the relationship between the computer's state and external conditions. The logic circuit code is executed repeatedly until the reaction-time clock indicates that the next action should be done. At that moment, the latch code is executed causing the latches to be updated, which results in new control and sensor information being available. Then, the cycle begins anew.

E.4 Code Generation

The action language specifies an ordinary logic circuit which is simulated on a 65C02 processor, so speed was a central consideration in the choice of representation. Wires are represented as bytes and circuit elements and their connections are represented as machine code. For example, an ADD gate adds the values stored in the bytes representing its input wires and stores the result in the byte representing its output wire:

```
LDA    WIRE1        ; Get the value on wire 1.
ADD    WIRE2        ; Add the value on wire 2.
STA    WIRE3        ; Store the result on wire 3.
```

With this representation scheme, a circuit simulated on a Programmable Brick has a maximum size of three thousand elements and can in theory run once every fifty milliseconds. While the representation of wires could have been designed so as to allow for a larger number of elements in circuits, the resulting simulations would have been slower. The straightforward representation provides the additional advantage of simplifying the compiler's code generation phase.

The implementation of the ACT compiler used in the workshop did not allow the simulations to run at the above theoretical maximum rate. The compiler did little to optimize the generated code and did not try to order the execution of the gates associated with actions to reduce the time to quiescence of activity in the circuit. This quiescence was established by letting the simulation run for a minimum of a tenth of a second. Also, the run-time overhead for maintaining the relationship between the state of the sensors and the state of the circuit's inputs was higher than had been expected. The compiler was written this way to keep its development simple and to facilitate its integration with a debugger. In the end, these considerations did little to affect the execution programs written by the children as the circuits that were generated by them had at a maximum three hundred gates.