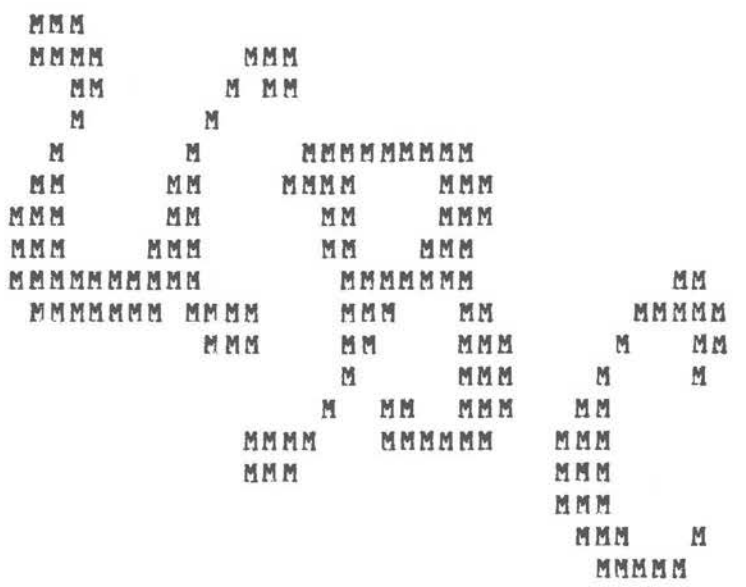


ANTICS - A System for Animating LISP Programs

by

Mark S. Dionne and Alan K. Mackworth

Technical Report 76-7

October 1976

Department of Computer Science
University of British Columbia
Vancouver, B. C.

ABSTRACT

A system, named ANTICS, has been developed for producing animated films, film strips, or slides depicting the execution of LISP programs. The design, implementation, and use of ANTICS are discussed and it is compared to existing systems. ANTICS may be used by entering very simple commands which produce real-time animation. The system may be "backed up" and manipulated interactively. Advanced commands and a set of graphics primitives are available which permit an instructor or film-maker to control details and to add features not provided. ANTICS may therefore be used as an interactive educational tool or as an animation system. It is inexpensive to use: a three minute film showing the operation of the recursive function MEMBER was produced for a total cost of $12.00. The implementation is dependent on specific hardware, but the design, which is based on the organization of the LISP EVAL function, could be used on other systems.

1. PROGRAM ANIMATION

1.1 INTRODUCTION

A system named ANTICS has been produced which animates programs written in LISP [1]. The system is designed for use by an instructor in a programming course, allowing her to make films which will demonstrate various features of the LISP language and of algorithms programmed in LISP. The instructor may control the parts of LISP which are animated and the amount of detail which is shown. Also, the system is simple and inexpensive enough to be used as an on-line interactive instructional aid.

1.2 MOTIVATION

Teaching beginning computer science almost always has involved graphic tools. Flowcharts, data structures, system organization diagrams, parse trees, hardware schematics and graphic representations of certain algorithms each have an important place in the language of computer science. Many computer science concepts, such as the stack, the linked list, and the array have an implicit graphic language of their own which is invariably taught in beginning courses. Other more dynamic concepts do not have well established graphic representations, since their actions are not easy to convey with fixed images. Examples of these are recursion, iteration, binding, and algorithms such as searching, sorting, and parsing.

The motive for animating computer programs is to provide new graphic expressions for complex dynamic processes in the field of computer science. Corresponding ideas in physics have been illustrated in several computer produced films [2, p. 313].

An excellent appreciation of a system can be gained by hands-on experimentation. This is the motivation behind LOGO, a simple programming language which teaches mathematical and programming skills by being "played with" by children [3]. In a similar way, the student could explore the LISP world visually and at her own pace if the animation system were made interactive.

1.3 RELEVANT LITERATURE

Although computer animation has been fairly popular for a number of years, few attempts have been made to produce animated computer science films. Until recently, only one significant computer animated film related to computer science [4] had been produced. This film was produced by a system which cost approximately $600 per minute of film [5].

Baecker and his students have done work in this area [6] which Baecker has aptly named "program illustration" [7]. Two systems have been produced by them. One system animates any program written in a subset of LOGO [7, p. 160]. A fixed set of conventions determines how the execution of statements and evaluation of variables will be displayed visually. Optional parameter settings may be added to the program to tailor the timing and spatial positioning of the animation. The system has been used to animate a simple LOGO program which reverses a

character string, demonstrating features of the LOGO language as well as the concept of recursion. This system allows any LOGO program to be animated without adding special commands. The system is too expensive for production work, however.

The second system animates programs written in a subset of PL/I [7, p. 161]. Pseudo-comments, interspersed with the PL/I source code, call special functions which produce the animation. Producing a film with this system may take several hours of programmer effort since the pseudo-comments must be written for each program to be animated. However, more detailed control of the animation is possible than in the LOGO system. This system has been used to produce a film illustrating a sorting algorithm; the film followed the execution of the PL/I program without showing PL/I language features explicitly. The PL/I system can produce film clips for $100 to $200 per minute.

Both of these systems produce key frames which are used by a computer animation language to produce the final film. (See Section 1.5.) Neither system can be used interactively.

Hopgood has produced computer animated films illustrating hash table algorithms [8]. His system applies various algorithms to examples which are too large to be worked out easily by hand, thus giving students an appreciation for the methods when applied to non-trivial problems. This is claimed to show the advantage of certain algorithms more clearly than mathematical analysis or simple examples.

1.4 TYPES OF ANIMATION

Four major areas in computer science are likely to benefit

from animation. The methods developed are likely to differ, and when "program animation" is mentioned, the exact type should also be specified. The areas are:

1. Animating algorithms, such as sorting, parsing, or searching.
2. Animating programming language features, such as "DO" in FORTRAN or "COND" in LISP.
3. Animating hardware operation. Animating movement of data between registers, arithmetic units, memory and channels.
4. Animating concepts in computer science, such as control structures and data structures.

1.5 ANIMATION METHODS

Both hand animation and computer animation have made use of the key frame animation technique. An image is first produced for every "key" frame of the finished movie, and then less skilled artists (or a computer) produce intermediate, interpolated frames, resulting in the smooth movement of the figures from one key frame to the next. In the case of computer key frame animation systems, the key frames could be produced by an artist using a data tablet, or by another program. This method has been used primarily to produce artistic films.

Computer animation also may be produced by taking advantage of the nature of a particular graphics system. This is tentatively named "direct animation". The tools developed for interactive graphics are extended to produce a sequence of

animated displays. Many graphics systems provide means to displace, scale, rotate and change the intensity of sets of vectors. These facilities can be used to produce movement and other animation effects. The details of producing animation in this way are more involved than in key frame animation: timing considerations and display file organization may be difficult, and complex motions of artistic figures may be impossible. On the other hand, since most of the detailed animating is done at high speeds by the display processor, it is generally more feasible to produce real-time "live" animation by this method. A frame can be set up by the main computer and movement initiated. While the display processor is producing the movement the main computer can be working on the next major display change.

2. SYSTEM DESIGN

2.1 DISPLAY DESIGN

The overall purpose of the ANTICS system is to impart some knowledge of the basic operation of LISP to a beginning programmer. There are two fundamental concepts which form the foundation of LISP: the structure of S-expressions, and the operation of EVAL.

Although S-expressions are changed during the execution of a typical LISP program, they generally specify "structure" in some sense, and the first thing a student must do is understand

S-expressions as static data structures. ANTICS provides the capability of displaying arbitrary S-expressions either in "prettyprinted" form or as CONS-cells and pointers. Figure 1 shows a CONS-cell display of a function definition. Figure 2 illustrates the actions of the destructive functions, RPLACA and RPLACD. This figure was produced by a 25 line script which generated the structures in a timed sequence.

The operation of EVAL is a complex, dynamic process. It consists of a precisely ordered sequence of recursive operations on S-expressions, coordinated with the binding and unbinding of variables on a stack or an association list of variables and values (known as an a-list [1]). ANTICS bases its animation of LISP functions on the operation of EVAL, though it does not presume that the student has any knowledge of EVAL itself. ANTICS' animation simply reflects the various facets of EVAL as they are brought into operation, by displaying them in a suitable fashion. The details of the design have centered on selecting the parts of EVAL to display and devising suitable graphic representations for them.

The basic animation proceeds as follows (refer to Figure 3):

1. The form which is being evaluated is printed on the display screen and, after a short interval, a box is displayed containing the word "EVAL". To suppress detail and save screen space, printing only descends a set number of levels into the structure; any non-atomic structure at this level (level 3 in this example) is represented by an " ".

```
(DEFUN  MEMBER  (THING  LIST )
  (COND  ((NULL  LIST ) NIL )
         ((EQUAL  THING  (CAR  LIST )) LIST )
         (T (MEMBER  THING  (CDR  LIST )))))
```

MEMBER:

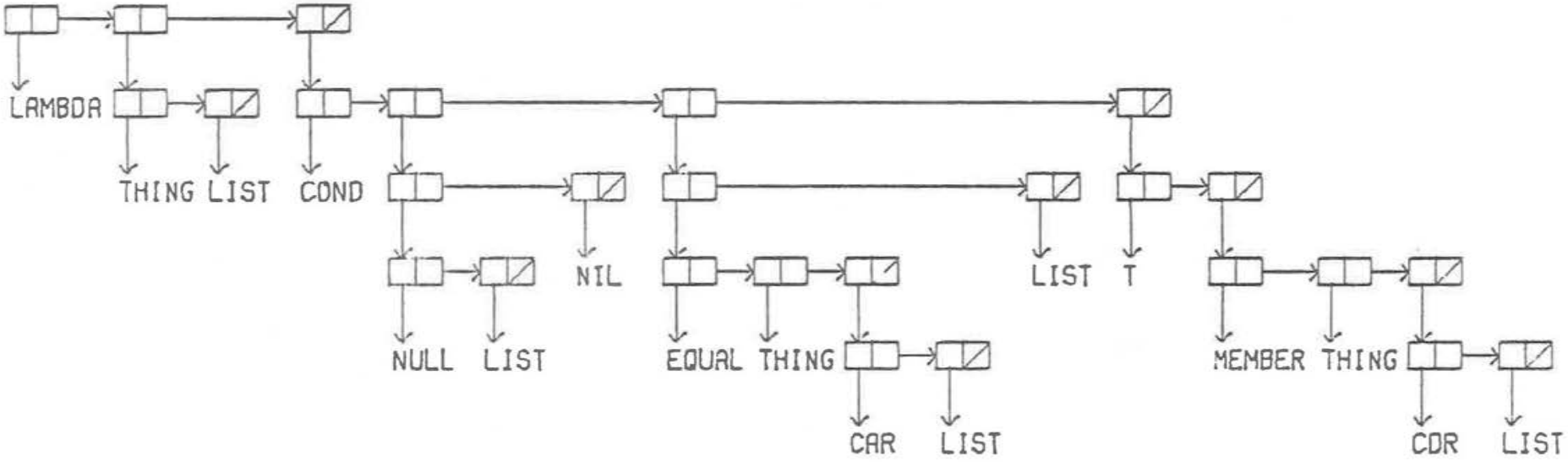
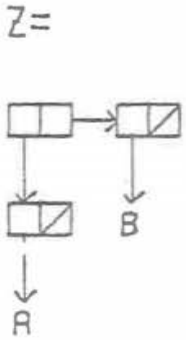
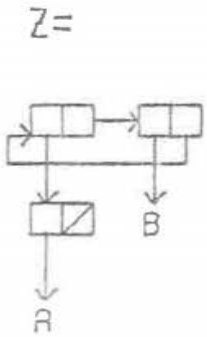


Figure 1. Display of CONS-cells

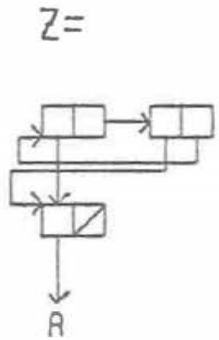
1 - (SETQ Z '((A) B))



2 - (RPLACD (CDR Z) Z)



3 - (RPLACA (CDR Z) (CAR Z))



4 - (RPLACD Z Z)

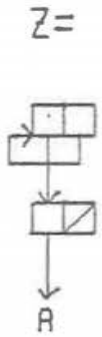


Figure 2. The effects of RPLACA and RPLACD

3 (a) 0:00

Figure 3. Snapshots of animation of LISP form evaluation
(Typical timing is indicated)

```
MEMBER:
(LAMBDA  (THING  LIST)
         (COND  ((NULL  LIST)  NIL)
                ((EQUAL  THING  (CAR  LIST))  LIST)
                (T  (MEMBER  THING  (CDR  LIST))))))
```

Figure 3 (b). 0:16

```
MEMBER:
(LAMBDA  (THING  LIST)
         (COND  ((NULL  LIST)  NIL)
                ((EQUAL  THING  (CAR  LIST))  LIST)
                (T  (MEMBER  THING  (CDR  LIST))))))
```

(MEMBER 'A '(C A T))

EVAL

(COND ((NULL LIST) NIL)
       ((EQUAL THING &) LIST)
       (T (MEMBER THING &)))

EVAL

| THING = A
| LIST = (C A T)

LIST = *UNDEF*
THING = *UNDEF*

Figure 3 (c). 0:24

```
MEMBER:
(LAMBDA  (THING  LIST)
         (COND  ((NULL  LIST)  NIL)
                ((EQUAL  THING  (CAR  LIST))  LIST)
                (T  (MEMBER  THING  (CDR  LIST))))))
```

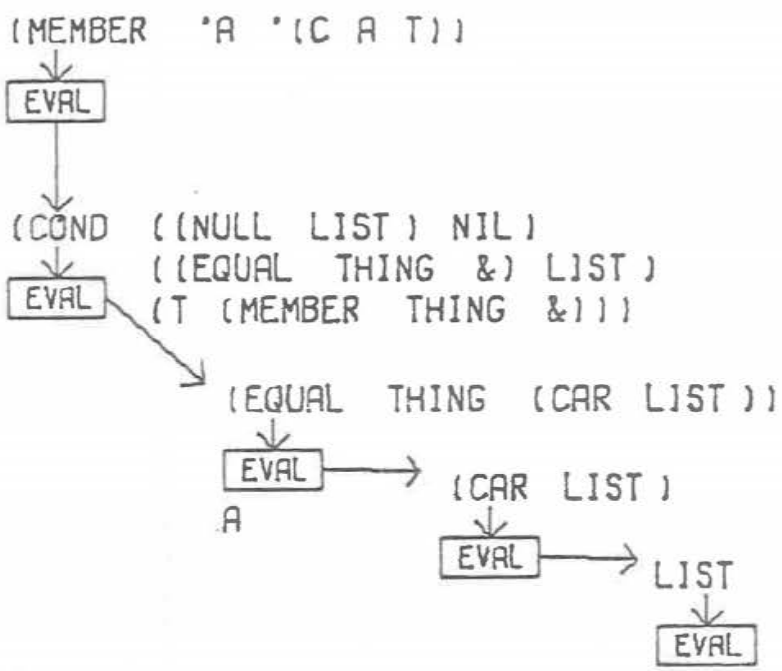


Figure 3 (d). 1:00

```
MEMBER:
(LAMBDA  (THING  LIST)
         (COND  ((NULL  LIST)  NIL)
                ((EQUAL  THING  (CAR  LIST))  LIST)
                (T  (MEMBER  THING  (CDR  LIST))))))
```

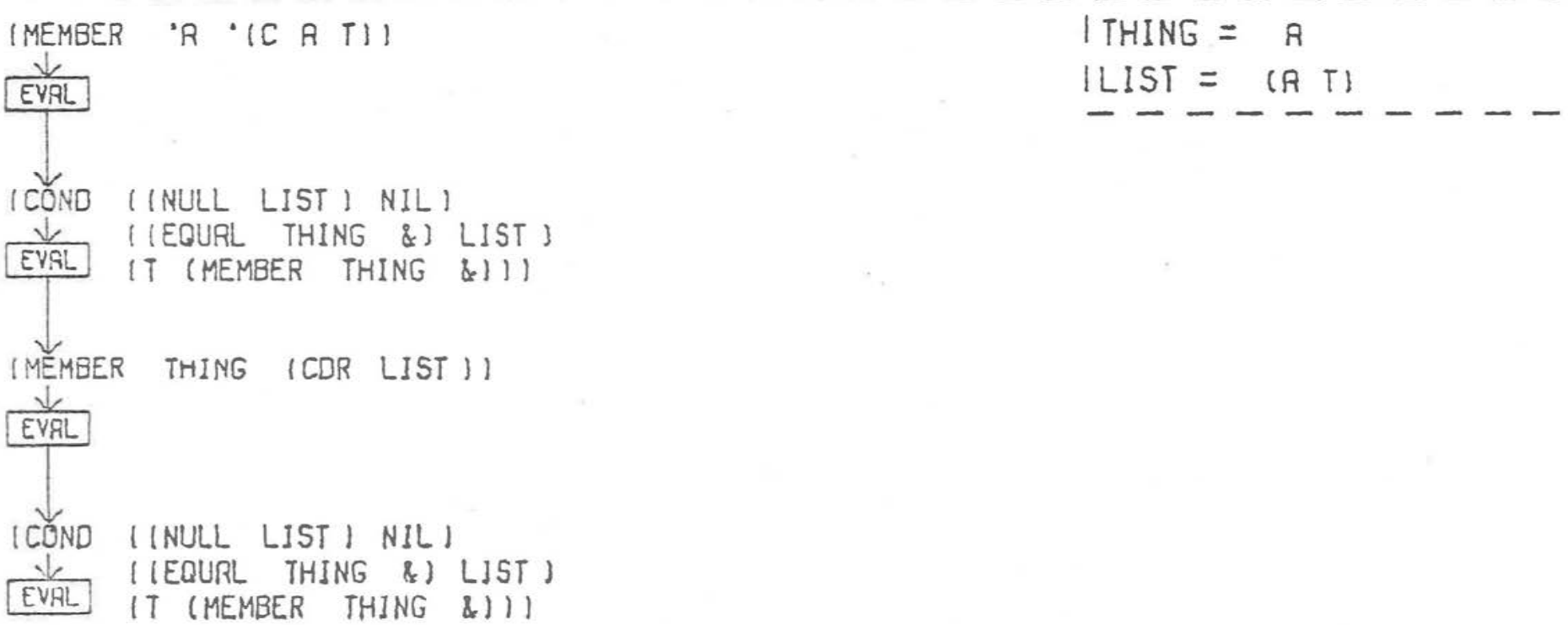


Figure 3 (e). 1:52

```
MEMBER:
(LAMBDA (THING LIST)
        (COND ((NULL LIST) NIL)
              ((EQUAL THING (CAR LIST)) LIST)
              (T (MEMBER THING (CDR LIST))))))
```

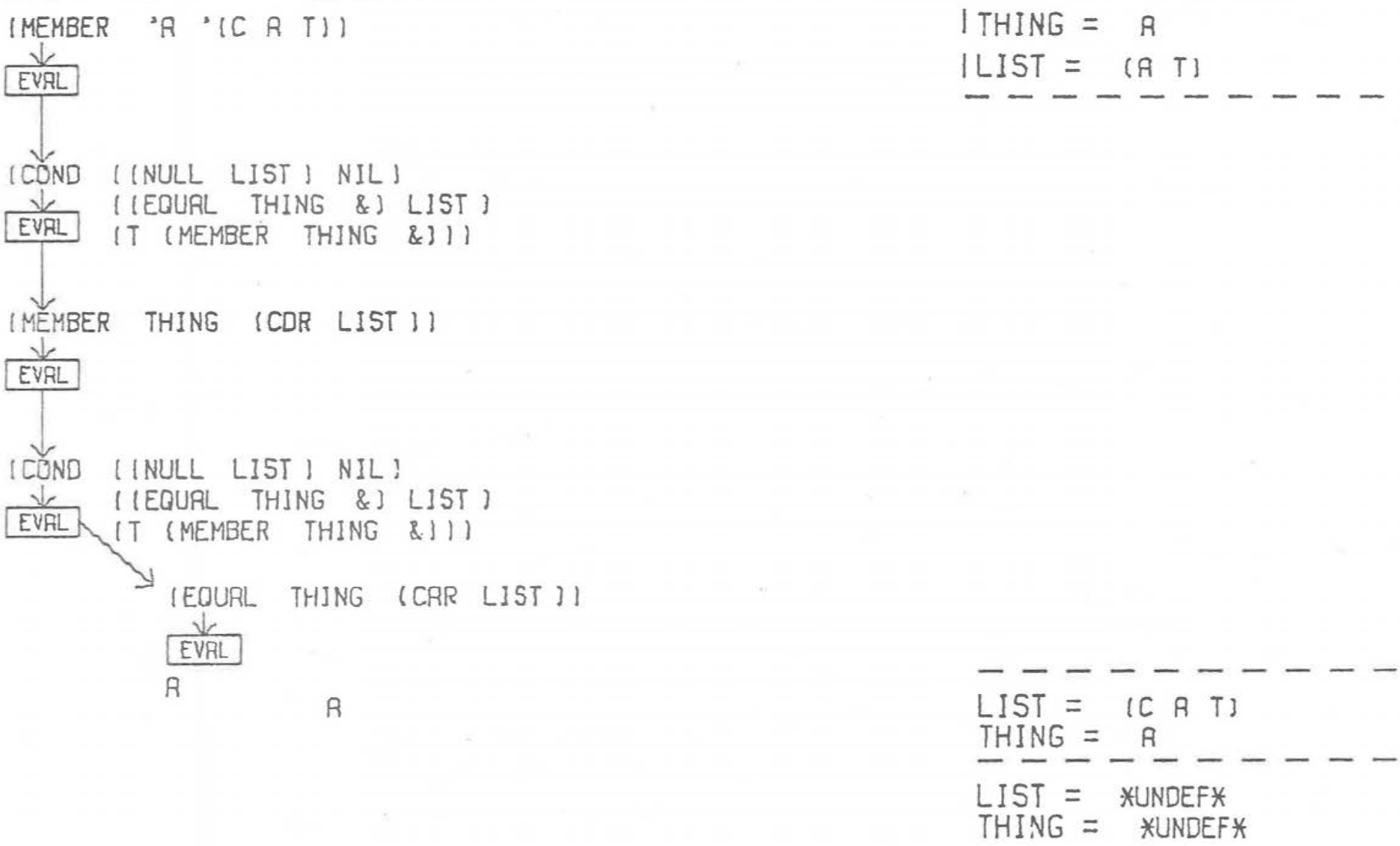


Figure 3 (f). 2:36

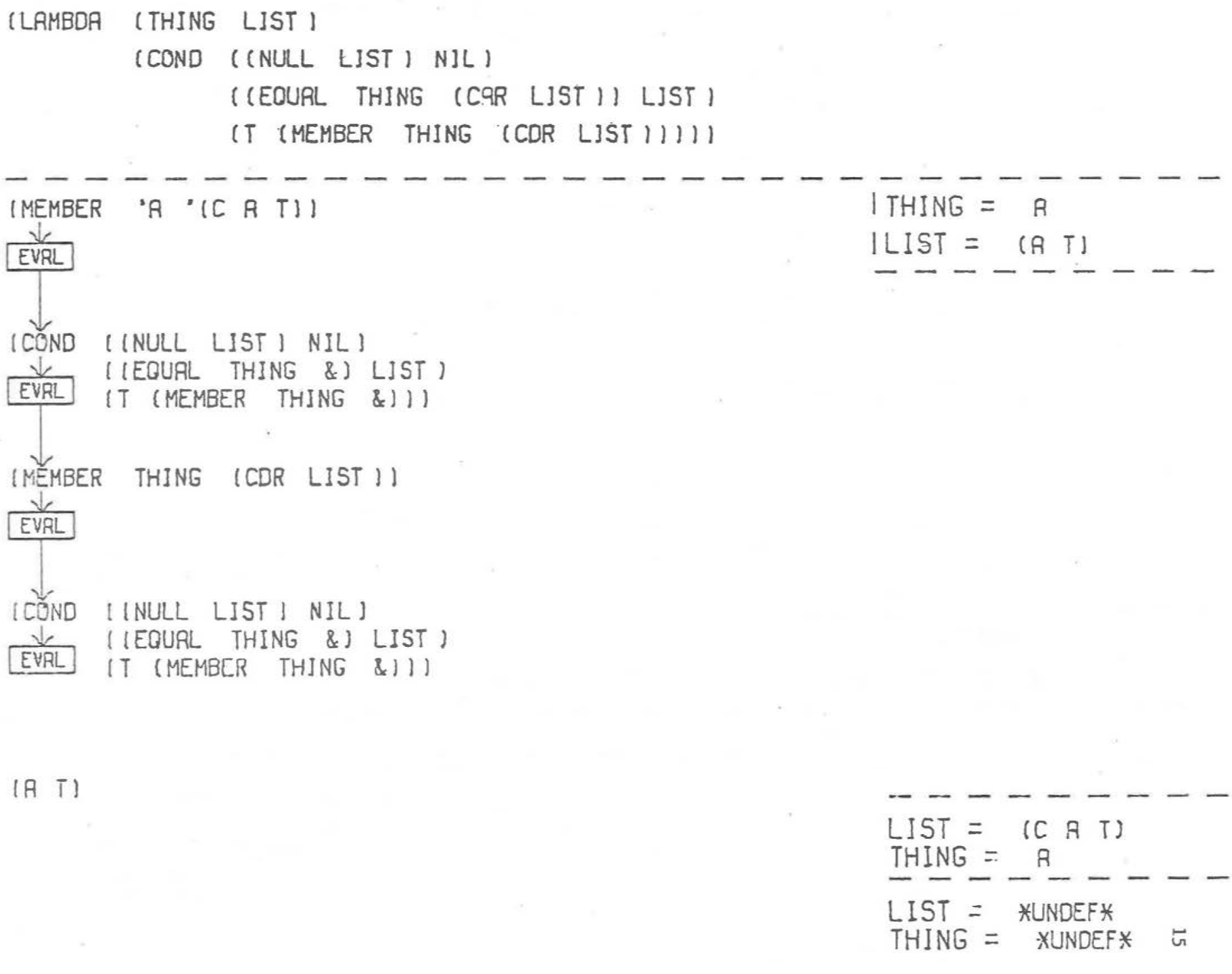


Figure 3 (g). 2:48

```
MEMBER:
(LAMBDA  (THING  LIST)
         (COND  ((NULL  LIST)  NIL)
                ((EQUAL  THING  (CAR  LIST))  LIST)
                (T  (MEMBER  THING  (CDR  LIST))))))
```

```
(A T)
```

```
|THING =  *UNDEF*
|LIST =  *UNDEF*
```

Figure 3 (h). 3:04

2. If the form contains arguments which must be evaluated, then their evaluation is recursively animated to the right of the present display and the returned values are moved into a column below the present EVAL box. After these EVLIS values have all been returned, their bindings to the dummy variables of the LAMBDA-expressions are optionally displayed and the old values of the dummy variables are placed on the stack.
3. If the function being evaluated is part of the LISP system (a SUBR, a function written in assembly language) the returned value is displayed after removing the original form from the screen. If a user-defined function is being evaluated, the expressions which constitute the body of its definition are evaluated in turn. Expressions which are conditions in a COND or SELECT function are animated to the right of the current EVAL box, while all other expressions are animated so as to appear directly below it.
4. Throughout the animation, the definition of a function may be displayed at the top of the screen, and the portion of it that is currently being animated will be intensified.
5. The LISP stack or association list optionally may be displayed on the screen, as well as current variable bindings. Changes in these are shown as they occur.
6. Whenever the EVAL animation or the stack display encounters the border of the display area, part of the

display is "rolled off" the screen and disappears, thus freeing space on the display screen. When the evaluation is exiting the rolled portions reappear on the screen.

2.2 SINGLE-FRAME VS. REAL-TIME ANIMATION

A computer system to produce general-purpose, high quality animation probably would have to produce one frame at a time, given the present state of the development of computer graphics systems. A high priority in the design of ANTICS was that the system be able to produce animation in real-time. There were several reasons for this:

1. The implementation of ANTICS would be very difficult if all testing had to be done by filming one frame at a time.
2. The production of properly timed films for classroom use also would be very difficult.
3. A real-time system also could be made interactive, and this would provide an extremely versatile, though perhaps expensive, teaching tool.
4. Many animation systems have produced one frame at a time. It would be valuable to help to demonstrate that this is not always necessary.
5. A real-time system easily can be made to display one frame at a time if this is needed to produce high quality film.

The goal of producing a real-time system caused some implementation difficulties, but these were considered minor in

light of the first two reasons given above. In addition, single-frame animation is much more complicated or impossible if the scan time of the display screen is longer than the longest camera shutter time, and this was the case with ANTICS and the available equipment.

2.3 SCRIPT DESIGN

It was assumed that a person making a film with ANTICS would be a relatively sophisticated LISP user (a course instructor or assistant), and that the instructions necessary to produce a film (called the script) would not need to be especially simplified. On the other hand, since the system would also be interactive, a relatively naive LISP programmer should be able to produce instructive animation with only a line or two of input. The result was the following design:

1. A single command, #EVALQ, will animate the evaluation of any expression.
2. Several simple commands control the other basic displays.
3. Additional commands provide more elaborate control over details of the animation.
4. A sophisticated user may insert "breakpoints" into the definitions of the functions to be evaluated, thereby changing minor details in the animation.
5. A general-purpose graphics language is available to build special displays.

The types of commands that are allowed in the script can be broken down into several groups: one group controls the major

displays, a second group controls details of the animation, and a third group is concerned with details of producing films, such as timing and camera control.

2.4 GRAPHICS PRIMITIVES

In order to provide flexibility to the user of ANTICS, a general purpose graphics language was provided. The user would most likely want to produce titles, diagrams, and explanatory text. She also may want to highlight portions of the animation by adding special effects such as pointers or outlining boxes. It should be possible tc include figures sketched with the light pen or data tablet.

An obvious choice was to base the graphics language on a subset of GLISP, a LISP based graphics language [9]. GLISP has a useful set of primitive functions for drawing lines and text. It also allows the user to sketch figures with the light pen or data tablet and to adjust their size and position on the display screen, and also to save figures in a library. Furthermore, GLISP is easy-to-learn, supported and well-documented.

2.5 HUMAN FACTORS AFFECTING DESIGN

Since ANTICS is an educational tool, human factors considerations were important in its design. At one level, the graphic representation of concepts falls under this category. At another level, the timing and sequencing of the animation were considered. Because ANTICS is also an interactive graphics system, response time and interaction methods were taken into

account.

The graphic representation of S-expressions is a well established standard form, and there was not much latitude in its design. However, the method of animating EVAL is not standard; it is based on a blackboard method used in a course on LISP, C.Sc. 509, taught by Raymond Reiter at the University of British Columbia. The well-defined nature of EVAL tends to limit the possibility of radically different graphic representations. The real choices lie in the complexity of the functions animated and the amount of detail shown, and ANTICS provides the instructor/animator the ability to match these choices to the level of her students.

The timing and sequencing of animation were important considerations in the design of ANTICS. Early versions of the program produced animation that was difficult to follow because the action moved from one side of the display screen to the other without warning. The idea of a "follower" was developed: a moving figure on the screen naturally catches the eye and directs the viewer's attention to a new area of the screen. Movement was found to be very compelling visually -- in fact, any movement on the screen seems to lock the viewer's attention. Because of this, the rule for lengthening ANTICS' animation gives priority to making the static portions of the film longer, since these sections allow the student to absorb the meaning of the animation sequence. Movement is a dominating element in entertainment animation, but its use in instructional animation must be considered carefully [2, p. 66].

The overall speed of the animation must be slow enough for

the viewers to follow, and this speed is not easy to define. No exact speeds can be recommended, but the following points were considered:

1. Viewers gain skill at following the animation as they become more familiar with the form of representation used in the animation.
2. For naive LISP programmers, "the slower the better" is perhaps a very realistic rule for the speed of animation. (On the other hand, an authoritative source claims, "It has been proved by experience that the shorter the film the more effective the instruction is likely to be, because of the intense concentration which the student must give. . ." [2, p. 136])
3. A short film can be shown several times consecutively and different facets of LISP can be emphasized each time by the instructor.
4. A suitable projector can stop the animation while the instructor makes detailed explanations.

The speed of animation can be adjusted with the #RATE command or, interactively, using a dial.

3. USE OF THE SYSTEM

3.1 THE ANTICS SCRIPT

The input to ANTICS is a series of LISP forms called a

script. When a movie is being produced this script must be entirely in a file so that it can be read without interruptions, which would confuse the timing of the movie; however, the script may be entered one line at a time when the system is being used interactively or while experimenting with ideas. A script may consist of several lines to produce an animation of a single LISP evaluation, or it may contain a hundred or more lines to produce a complete movie with titles and explanatory text. The following script produces a rudimentary animation:

```
(#START)
(DEFUN MEMBER (THING LIST)
  (COND ((NULL LIST NIL)
         ((EQUAL THING (CAR LIST)) LIST)
         (T (MEMBER THING (CDR LIST))))))
(#DISPLAY THING LIST)
(#WAIT 5)
(#EVALQ (MEMBER 'A '(C A T)))
```

#START initializes some parameters and blanks the display screen. DEFUN simply defines the function of interest in the usual way. #DISPLAY causes the names and values of the atoms THING and LIST to be displayed on the screen. They will remain until the screen is blanked, and the display will be changed whenever their values change. #WAIT causes a five second pause before the next line of the script is executed. #EVALQ causes a complete animation sequence, lasting perhaps several minutes, of the EVALuation of the given form. During this animation the values of the atoms specified by the #DISPLAY command are updated constantly.

There are two additional major commands which could be added to the above script anywhere before the #EVALQ command:

(#STACK THING LIST)

(#STAR MEMBER)

#STACK causes a stack to be displayed during animation showing the bindings of the atoms THING and LIST. No display is produced when the #STACK command is given -- the display is shown during the animation produced by #EVALQ. #STAR causes the definition of MEMBER to be shown immediately at the top of the display screen. During the animation produced by #EVALQ the portion of this "star" function currently being evaluated will be intensified.

These commands are the basic high-level animation features of ANTICS. The user can obtain a great deal of variety, however, by using other more specialized commands, selecting options, changing parameters, and producing additional graphical displays with the graphics primitives included with ANTICS. It is possible to abbreviate certain features of the animation after they have been displayed a set number of times.

The general purpose graphics language available to the user of ANTICS contains both graphics primitives and special functions for displaying LISP structures. It can be used within special breakpoints which may be inserted into a function whose evaluation is being animated. By using these breakpoints, which are invisible to the viewer of the movie, the user of ANTICS may tailor the evaluation animation by omitting unnecessary detail and displaying explanatory messages and figures at key points in the animation.

Figures 3(a) through 3(h) are snapshots of the animation produced by the above script. The "starred" function is at the top of the display, the current variable bindings are in the upper right, and the stack is in the lower right corner. Figure 3(h) shows the end of the animation: the stack is empty and the value returned by the function call is all that is left.

3.2 USING THE SYSTEM INTERACTIVELY

Although ANTICS is designed primarily for making movies, it is fast enough to support interactive use. Interaction is by means of the light pen and function buttons. An overlay card labels the functions of the various buttons. Whenever ANTICS is waiting, both of these devices are active. During animation, #WAIT is called before and after evaluating each form.

There are two modes of interactive operation STEPMODE and AUTOmatic. STEPMODE is selected by pressing the STEPMODE button or by using the light pen in any way. While in STEPMODE, the function #WAIT always waits until either the STEP button is pushed or the STEP light button is selected with the light pen. In AUTO mode, the #WAIT function waits the specified time, unless the STEP button is pushed first. AUTO mode is selected by pressing the AUTO button or by pointing the light pen at the AUTO light button.

The light pen has two other functions as well. At any time it may be pointed at any variable in the list created by #DISPLAY, and a new value for the atom may be entered through the keyboard. The light pen also may be pointed to any part of

the evaluation display, and the animation will back up to that point and restart.

There are several other buttons which may be used at any time. These can terminate or "backup" the animation, activate the camera, change the rate of animation, or plot the contents of the display screen.

4. IMPLEMENTATION

4.1 SYSTEM ORGANIZATION

The implementation of ANTICS is dependent on a unique environment of hardware and software. This is unavoidable due to the interactive nature of ANTICS: interactive graphics systems tend to be hardware dependent. The general organization of the implementation environment is fairly typical of graphics systems however, and it may be possible to modify ANTICS to work on other systems without undue effort.

The ANTICS system is implemented on a Model 10 Adage Graphics Terminal which communicates with an IBM 370/168 computer operating under the Michigan Terminal System, MTS. ANTICS is written in LISP/MTS, an interpreter similar to LISP 1.5 [1]. LISP/MTS uses an internal stack rather than an a-list as do LISP 1.5 and several other LISP systems. The animation which is produced reflects this aspect of LISP/MTS and several other minor details, but since ANTICS contains its own EVAL function these details could be changed easily. LISP/MTS

communicates with the graphics terminal through a simple interface.

4.2 DISPLAY FILE ORGANIZATION

The organization of the contents of the Adage Graphics Terminal buffer, refered to hereafter as the display file, was altered several times in the course of the development of ANTICS, and was a major part of the implementation effort. The actual display file stayed fairly far from the ideal due to hardware and software limitations and design considerations. All of the display file organization was implemented at the LISP level of the system. For more implementation details see [10].

4.3 METHODS USED IN THE PROGRAM

The heart of ANTICS is a set of LISP functions constituting a version of the LISP EVAL function. This version of EVAL is interspersed with calls to animation routines, and this is how all animation is produced. The stack, variable and "star" displays are also driven by ANTICS EVAL, and user breakpoints are detected and processed by it. As a result, the animation naturally follows the execution of EVAL and ANTICS has a simple underlying structure.

The routine which displays CONS-cells also uses some novel methods. When given a list it draws a CONS box, and then calls itself recursively with the CAR of the list. The return value is a pair of dimensions indicating the physical size of the display which was generated. These dimensions are used to

locate the display of the CDR which is generated by a second recursive call. A preliminary pass detects circular lists, such as those in Figure 2.

4.4 COST AND EXECUTION TIME

It was originally assumed that ANTICS would be expensive to use since it combined two relatively expensive items in terms of computer charges: interpreted LISP and graphics. Development and debugging costs were not particularly low, but the cost of producing animated films has turned out to be surprisingly low. A three minute twenty second film illustrating the execution of the recursive function MEMBER was produced for a total computation cost of $4.23 using 5.4 seconds of central processor time. The cost of film and processing was $7.50. The example could have been slowed down two or three times to make it easier to follow. This would only add a slight additional computation charge, as only the elapsed time and virtual memory usage would increase.

5. CONCLUSIONS

5.1 COMPARISON WITH EXISTING SYSTEMS

Few program animation systems have been developed, and it is difficult to compare ANTICS to those that exist. ANTICS' strong points, its cheapness and interactive capability, are not

found in any other systems. Also, existing systems have been designed to present subject areas quite different from ANTICS'. One thing that can be said is that ANTICS is not a general system in the same sense as the PL/I animation system described by Baecker [7]. ANTICS' primary use is to teach features of LISP. After LISP is mastered and students are familiar with the style of ANTICS' presentation, the system can be used to illustrate general properties of algorithms. The intimate details of the process of evaluating a LISP form have been illustrated with ANTICS by animating the application of an a-list version of the function EVAL, written in LISP, to a simple form.

5.2 PRODUCING FILMS

Simply filming the display screen with a movie camera running at normal speed will not produce an acceptable film because of stroboscopic effects. Automatically tripping the shutter at the start of each display frame is not totally satisfactory either. The shutter should stay open at least as long as the time required to display the most complex frame on the graphics terminal. Most movie cameras have a maximum shutter time of 1/40 second, which easily exceeded. This problem could possibly be avoided if the camera were capable of double exposing single frames. However, 35mm slides have produced excellent results; they may, in fact, be better for pedagogical purposes. A set of 50 slides, suitable for classroom presentation, is available at cost from the authors.

5.3 FUTURE DIRECTIONS

The success of ANTICS has inspired us to continue with its development. Next we intend to adapt it as far as possible to ordinary display terminals to make it more widely available to students, and to explore the possibility of producing videotapes directly. Far more ambitiously, we are considering the problem of automating an introductory LISP course. How should we organize our knowledge of LISP programming to communicate it using a graphics-based, computer-aided learning system? Ideally, such a system would have, in addition, many of the attributes of Winograd's proposed programmer's assistant [11]. Before proceeding with such automation one must ask if there exists a need and a role for such a system. In our opinion it should not, and probably could not, supplant the traditional forms of instruction. Finally, as various groups approach the concept of a LISP machine we should consider designing into such machines facilities that will allow the development of systems like ANTICS.

REFERENCES

1. J. McCarthy, M. I. Levin, et. al., LISP 1.5 Programmer's Manual, M.I.T. Press, 1962.

2. J. Halas and R. Manvell, The Technique of Film Animation, Communication Arts Books, New York, 1971.

3. S. Papert, Teaching Children Thinking, Papers of the IFIP World Conference on Computer Education, pp. I/73-I/78, Science Associates International, New York, 1970.

4. K. C. Knowlton, L6 Bell Telephone Laboratories Low Level Linked List Language, Two black and white sound films, Bell Telephone Laboratories, Murray Hill, N.J., 1966.

5. K. C. Knowlton, Computer produced movies, Science 150, 1965, 1116.

6. R. M. Baecker, Towards Animating Computer Programs: A First Progress Report, Proc. Third National Research Council Man-Computer Communications Seminar, pp. 4.1-4.10, National Research Council, Ottawa, 1973.

7. R. M. Baecker, Two systems which produce animated representations of the execution of computer programs, SIGCSE Bulletin 7, 1, 1975, 158-167.

8. F. R. A. Hopgood, Computer Animation Used as a Tool in Teaching Computer Science, Proc. 1974 IFIP Conf., pp. 889-892. North-Holland Publishing Co., Amsterdam.

9. W. Hall, B. Jervis and J. Jervis, GLISP - A LISP Based Graphic Language, University of British Columbia, Department of Computer Science, 1973.

10. M. S. Dionne, ANTICS - A System for Animating LISP Programs, M. Sc. Thesis, University of British Columbia Department of Computer Science, 1975.

11. T. Winograd, Breaking the complexity barrier again, SIGPLAN Notices 10, 1, 1975, 13-22.