CGTM 32
Nov. 1967

SYNTAX OF AN INTERACTIVE LANGUAGE

William E. Riddle

June 12, 1967

In the past few years, because of (and sometimes in spite of) implementation of time-sharing systems, many general and special purpose man-machine interactive systems have been designed and implemented. Each has had its special purpose -- engineering and design aides, teaching aid, limited general purpose computing utility, etc. Most have been approached in an ad-hoc manner, with the structure and implementation of the language processor dictated by the particular aims of each system. Thus most have been composed of a group of system routines, each pertaining to a particular function of the system.

During the same period, gains have been made in the area of syntax-directed compiling. Many compiling techniques have been developed which depend on the definition of a formal syntax for the language. Some techniques will even produce the specifications for a parsing routine for a compiler when a syntax specification for the language is given (or a full compiler if semantic information is also specified). These techiques are highly dependent on a formal syntax having been specified for the language.

This paper is an attempt to organize some of the common factors of interactive systems and to develop a formal syntax so that some of the current compiling techniques can be brought to bear on interactive systems.

Before presenting a full discussion of the syntax, a few general comments should be made. First, it is not meant that the developed syntax be used to generate a compiler for an interactive system. In fact, this would be a contradiction of terms, since by its real-time nature, an interactive system requires an interpreter. However, in either an interpreter or compiler, parsing of sentences in the source language is a primary function

and it is toward this end that the developed syntax is intended for use.

Second, the approach taken is to specify those general concepts which seem to be useful and appropriate for all types of interactive systems. (Due to my recent investigations of text-editor and picture processing systems, most examples will be drawn from these areas -- I can only hope that the bias toward these systems does not extend past the examples.) Because of this many syntactic types are defined as primitive to the system since they depend very heavily on the hardware available and the metjod of implementation. The developed syntax is therefore a skeleton upon which a full syntax for a particular problem-oriented interactive system could be developed.

The last philosophical point before taking up the subject at hand is concerned with expressive power of interactive languages. Designers who are most familiar with batch-processing systems have a tendency to carry language constructs which increase expressive power (and implementation difficulty) over into interactive languages. However, one important characteristic of interactive systems is the greatly reduced turn-around times. Thus, reducing the expressive power of the language and making the user give his instructions in a more elementary language, is no longer a burden to the user. Consider a quick example. Let there be a command, SEARCH, in a text editor, which allows the user to instruct the system to search the text for a string and tell him the number of each line containing it. Suppose he wants to find lines containing both the strings ABC and XYZ. Rather than include boolean algebra capabilities in the language, it is permissible, in an interactive environment, to require the user to give two instructions and then do the process of finding those lines common to both sets by himself.

Having these preliminary remarks out of the way, it remains to introduce the syntax. BNF is used, with

the added notation that primitive syntactic types (terminal symbols) are denoted by underscoring. For definitness of expression, it is considered that the user is working on a picture, which may take different forms in the different systems (i.e. in a text-editor, the picture would be one of the user's files).

Picture Composition:

Before discussing the process of effecting transformations upon a picture, the method of composing a picture should be outlined. What is meant here is the addition or insertion of new objects into an existing picture, and thus the process of starting an interactive session is merely considered to be a special case in which an object is inserted into a blank picture. An object is not confined to be a primitive structure in the system, but, as will be seen later, may be a structure composed of primitives to which the user has assigned a name.

There are two different types of objects which may be inserted into a picture. The first has as its distinguishing characteristic that its structure is defined (without resorting to already defined structures or primitives) coincident to the instruction commanding its insertion into the picture. In a graphical data processing application, such objects would be ones defined by light-pen tracking or use of a Rand-tablet [5] or Lincoln Wand [7]; essentially objects which are "free-form" in that their structure is defined by tracing the motion of some drawing device rather than by use of "building block" primitives in the system. For a text-editor, the inputting of new text via typewriter, card deck, or tape, into a file would be an example of this type of process.

The second type of object is one whose structure is predefined and which is referenced by name. A first instance of this would be the process of copying one part of the picture onto another point. In this instance, the name of the structure is a designational reference (i.e.

pointing at it with the light pen, see below for more). As a second case the user may insert one of the primitive structures defined for the system by referring to its name. In the final case, the user may call for an instance of some structure which he has defined and given a name (definitional capabilities are discussed more fully below).

The discussion of the first and last cases of predefined structures is deferred until some more basic definitions are given. Suffice it here to say that the names of the predefined objects in these two cases are ⟨designate⟩ and ⟨defined name⟩.

It remains to specify the second case of predefined structures, namely those composed of system primitives, and then give the syntax of the insertion operation. As an example, consider the primitives available in the system to be a line and an arc. These would have as parameters two points, and two points and a radius respectively. Lang, Polansky, and Ross [1] suggest that all primitives be written as a name which is considered to be a binary infix operator. Thus, assuming P_i is a designator of point i:

$$P_1 \text{ LINE } P_2$$

would be a line from point 1 to point 2 and:

$$R_1 \text{ ARC } P_1 \text{ TO } P_2$$

would be an arc from point 1 to point 2 with radius R_1. In this last example, ARC is a binary operator with R_1 and P_1 TO P_2 as its operands. The authors limitation to infix operators stems from their use of Ross' Algoritmic Theory of Languages and his associated processors (reference given in 1). However, this restriction has one redeeming side effect, concerning the concatenation of primitives. Consider the task of specifying a quadrilateral with apices at P1, P2, P3, and P4.

P1 LINE P2 LINE P3 LINE P4 LINE P1

is a much more convenient form than

P1 LINE P2, P2 LINE P3, P3 LINE P4, P4 LINE P1

Furthermore, the meaning of the first form is no less apparent when it is realized that some operands can serve dual

purposes as both the first operand of one operator and the second operand of another operator. Since this ease of expression is felt to be important, and absent from pre- or post-fix notation, the principle of infix operators should be kept, including the restriction that they be binary operators.

The final task of this section is to specify the syntax for picture composition. Assume for the moment a syntactic type Basis primitive unit which is, for example, a point on a picture. The two primitive structures felt important enough to mention explicitly are LINE and ARC.

<line> ::= LINE Basic primitive unit
<arc> ::= ARC Basic primitive unit TO Basic primitive unit
<primitive base> ::= Primitive type Basic primitive unit
Primitive type <Primitive>
<Line>
<arc>

A primitive type is one of those basic structures which the designer has decided to include in the definition of the system.

<Primitive> ::= Basic Prim unit <Primitive base>
<Primitive><Primitive Base>

This definition allows operands to be shared amoung operators.

<object> ::= coincidently defined object
<designate>
<defined name>
<primitive>
<compose command> ::= INSERT<object>

Structure Designation:

Now that there is a means for composing a picture, the problem arises of picking out a structure or sub-structure within it. The basic problem here is to allow the user to pick out various structures of differing complexity with a fair degree of ease.

To begin with, define Basic primitive unit as that primitive of the system which is the smallest recognizable unit. In a picture processing application this would probably be a point, whereas in a text editor, it would most likely be a character. The system also has defined primitive types, and the user should be able to designate one of these with a single command. These primitive types are composed of basic primitive units and are considered to be a full structure, wholly connected. For a picture processing application, these would be a line and an arc, as a minimum, and perhaps triangles, squares, or arcs of differing curvature, depending on the nature of the application. In a text editor, on the other hand, these primitive types would be words, lines, paragraphs, and pages.

The user should also be able to designate a list of primitive types by designating each in turn. This is really just an iteration control device, allowing the user to effect an action on a list of objects rather than typing the command to effect the action on each one.

Finally, the user should be able to designate a structure which he has defined. This could be handled by imposition of a rule stateing that a defined structure is considered to be wholly interconnected and an action effected on any of its parts is considered an action upon all of it. It is not clear, however, that the asymmetries which this introduces are necessary. Consider the following two cases. First, the user inserts an instance of a square which he has defined into the system. Pointing at one side he gives the delete command and the whole square is erased. Second, the user constructs a square out of four lines. Now pointing to one side and giving the erase command, only one side will be erased since the square is not a wholly interconnected structure. This asymmetry is eliminated by requiring the user to give the generic name of the structure as well as point to one section of it. Thus, in the first example above if he were to point to one side and give the generic name "line", then only that

side would be erased; however, pointing to any side and giving the generic name "square" would erase the total structure.

So far, it has been specified what generic names can be used in a designational expression. In order to specify a particular occurance of the structure named, the user must use a <u>designational action</u> such as pointing with the light pen or giving the identifying number for a line of text. So far, the syntax is as follows:

<generic designate> ::= <u>basic primitive unit</u>
<u>primitive type</u>
<defined name>

<generic group> ::= <generic designate>
<generic designate> <u>delim</u> <u>basic prim unit</u>
<generic designate> <u>delim</u> <u>primitive type</u>
<generic designate> <u>delim</u> <defined name>

This is just a list (possibly of only one element) of generic names.

<designate> ::= <generic ~~designate~~ group> <u>designational action</u>
<u>ALL</u>

A group of structures in the picture or all of the picture.

There is one further designational method which should be allowed. This is designation by means of some associative relations. Thus the user of a picture processing system may want to talk about the center of a circle or the intersection of two lines. Whereas the text editor user would want to designate all lines containing a certain string of characters. To allow these, first define <u>associative types</u> as words, primitive to a given system, which describe recognizable associations. These might be CIRCLE, INTERSECTION, etc. Then add

<designate> ::= <u>associative types</u> <generic group> <u>designational action</u>

Definitional Capabilities:

There are two important definitional capabilities which should be under user control. The first pertains to conventions which are in effect during an interactive session, while the second pertains to the ability to extend the system by defining structures.

With respect to the first, the conventions meant are those which are in effect due to their definition by the system's designer. These, for example, have to do with the words or characters which are used to denote commands. The user should be able to direct the system to accept the word INPUT as a command word for the compose command rather than (or perhaps even in addition to) the word INSERT. Other examples arise in text editing where tab control and formatting information should be under user control, with meaningfull default assumptions. The syntax for setting these options, or conventions, is as follows:

<set command> ::= SET option name = option value

With respect to the second definitional capability, the user should be able to define a name for a picture which he has constructed by giving the command:

<define command> :: = DEFINE AS name

This command would attach the name to the structure comprising the current picture. In text editing, this command would serve to assign a name to the file of text that the user has constructed. Once the name has been defined (and put as a member of an internal list of recognizable objects) the user may then use the name as a <defined name> to ask for an instance to be displayed, etc.

Closely allied to this second capability is the ability to define a characteristic of a structure (rather, assign a value to a characteristic of a structure). In this manner the user may impose constraints upon the picture. For example, in a picture processing system, the use may want to insure that some line is vertical or horizontal, or that two lines are equal, parallel, or

perpendicular. If we define constraint as a generic name for a group of constraints defined in a system, then

<impose command> ::= MAKE <designate> Constraint

Transformations:

The most important aspect of picture processing and text editing systems, and for that matter any interactive system, is the ability to effect transformations on structures which have been composed in the system. This section will give the general syntax of transformational commands and then outline the syntax more fully for each of the specific applications mentioned above.

Let trans command word be the set of command words which designate the legal transformations in the system. Then in general:

<transformation command> ::= trans command word <designate>

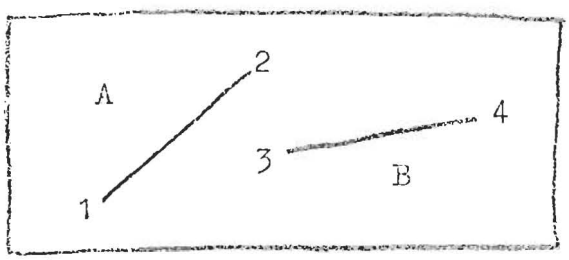
In trying to make this more specific with relation to one of the mentioned applications, about all that can be done (and need be done, really) is to specify the legal transformations. In regards to text editing, the transformations are straightforward. The user must be able to ERASE or REPLACE lines, paragraphs, pages, and even characters within a line. He would also want to DISPLAY any or all of a text. The important thing is that if <designates> are defined properly so the the user has flexibility in specifying just what string or group of strings he is considering, the user has a very powerful tool for correcting and editing text with only these three transformations.

Picture processing, on the other hand, has more basic transformations [4]:

trans command word ::= MERGE MOVE ROTATE REFLECT
ERASE DISPLAY EXPAND SHRINK
OUTPUT

all of which are self-explanatory save for the last three. EXPAND and SHRINK pertain to enlarging or diminishing the contents of some area of the picture (or the whole picture) which is usually considered to be a "window". Usually, the windows are defined to be squares delimited by a grid of fixed dimensions. With these capabilities the user may have control over the contents of the scope face and may enlarge some area to put in further detail, and then return it to normal size when finished. OUTPUT would call for the picture to be putout in hard copy form (microfilm, photogragh, plotter, etc.).

There are, however, some complications which arise in these transformations (for picture processing) which were pointed out by I.E. Sutherland [4]. Essentially, these arise from various associations, explicit and implicite, which may exist between structures in the picture. Consider the following picture:



If the ~~square~~ rectangle is erased, then the lines may or may not be erased depending on whether they are defined to be associated with the square by an INSIDE constraint. On the other hand, if the ends 2 and 3 of lines A and B respectively are merged then the lines could also be considered to be merged. And if lines A and B are merged then so should their end points. Even more complications arise with the MOVE command. Situations may arise where only one end of a line should be moved while the other end remains in position. But situations may also arise in which the whole line would be moved at once.

Interactions:

To complete the syntax, the following definitions of interactions are given.

<command> ::= <compose command>
<set command>
<define command>
<impose command>
<transformation command>

<interaction> ::= responce <command> sendchar
responce <anything> cancelchar

In this, responce is some signal given by the system to indicate that it is ready to accept another command, sendchar is a punctuation mark used by the user to indicate that the line is to be accepted as a command, and cancelchar is a punctuation mark sent by the user to indicate that he wishes this line to be ignored.

In addition to this definition of an interaction, it is important to allow question-answer interactions:

<interaction> ::= query answer

where query is a set of questions and answer is a set of their possible answers. Thus the user of a text editor may be asked if he really wants that file deleted before the file is destroyed. Or the user of a picture processing language may pick an alternative answer about any possible constraints upon a generated line by picking those characteristics which apply out of a list of possible characteristics.

Conclusions:

This report has presented the bare minimum of a syntax for an interactive language. It itself is not useful as the actual syntax for such a language but it does serve as a useful outline and checklist for the designer of an interactive system. Also, a useful syntax

may be built from this skeleton by filling in the primitives defined here with syntax definitions of those which are actually present in the system under consideration.

BIBLIOGRAPHY

1. Lang, C.A., Polansky, R.B., Ross, D.T., "Some Experiments With an Algorithmic Graphical Language", MIT Technical Memorandum, ESL-TM-220, August, 1965.

2. Sutherland, W.R., "On-Line Graphical Specification of Computer Procedures", MIT Technical Report 405, May 1966.

3. Stotz, Robert, "Man-Machine Console Facilities For Computer-Aided Design", SJCC, 1963, pp. 323,328.

4. Sutherland, I.E., "Sketchpad, A Man-Machine Graphical Communication System", SJCC, 1963, pp. 329,346.

5. Davis, M.R., Ellis, T.O., "The Rand Tablet: A Man-Machine Graphical Communication Device", FJCC, 1964, p 325 ff.

6. Reinfelds, J, et.al., "AMTRAM, a Remote-Terminal, Conversational-Mode Computer System", Proc. 21st Nat. Conf., ACM, 1966, PP. 469,478.

7. Roberts, L.G., "The Lincoln Wand", FJCC, 1966, pp223, 228.