SETL Newsletter Number 14

January 19, 1971
J. T. Schwartz

Additional syntactic extensions (not recorded in revised version of SETL paper)

1. Collecting existential quantifier.

If an existentially quantified variable x in a quantified boolean expression is included in curly brackets { }, then when the expression is evaluated the set of all values satisfying the condition will be assigned to x. Thus

if $\exists\{x\} \in e \mid C(x)$ then block;

abbreviates

$x = \{y \in e \mid C(x)\}$; if x ne nl then block;

More generally, suppose these brackets are attached to one (and only one) existentially quantified variable x_j in an initial sequence of such variables within a boolean expression B. Suppose that among the existentially quantified variables x which follow x_j, certain others, namely x_k, x_ℓ, ..., are prefixed with the otherwise optional quantifier sign $\exists$. Then the evaluation of B will assign to x the set of all n-tuples belonging to a sequence which gives the value t to the total expression B. Note then for example that if we evaluate

$$\exists x_1 \in e_1, \{x_2\} \in e_2, x_3 \in e_3, \exists[x_4] \in e_4 \mid C(x_1,x_2,x_3,x_4),$$

then not only is an assignment made to x_4, but x_2 is assigned as follows:

$$\text{tups} = \{\langle y_1,y_2,y_3,y_4\rangle, y_1 \in e_1,\ldots,y_4 \in e_4 \mid C(y_1,y_2,y_3,y_4)\};$$

$$x_2 = \{\langle\langle *\underline{z}^2\rangle y, \langle *\underline{z}^4\rangle y\rangle, y \in \text{tups}\};$$

2. Upgraded i/o operator.

The proposed standard format read/print operators are too stiff. The following extensions are proposed. Please look over this point and let me have your opinion soon.

a) print and read lists:

print is generalized to

print $expr_1$, ..., $expr_n$;

which is equivalent to

print $expr_1$; print $expr_2$; ..., print $expr_n$;

Similarly, read is generalized to

read $name_1$, $name_2$, ..., $name_k$;

It will also be useful to allow these i/o statements to reference other strings than the standard input/output strings, with the same rules regarding end record characters, blanks, etc. applying to the general read as now applies to the special "read from input" statement. A read operation will treat all strings as if they are terminated by infinitely many er characters. The proposed syntactic forms are

name print $expr_1$, ..., $expr_n$;

which concatenates what would otherwise be output to the character-string second component of the value of name; the form of this value must be ⟨pointer, characterstring⟩, in which pointer designates a character within characterstring. Similarly

name read $name_1$, ..., $name_k$

reads according to the proposed conventions from the character string portion of a similar ordered pair, starting at the character designated by pointer; and advances pointer appropriately.

The form in which character strings will appear should be amended as follows: character strings will appear in their normal external form, except where they contain characters such as comma, or where without quote marks they might be mistaken for constants of some other sort. In such cases, quote marks will be used; within quote marks, the quote itself will be represented by a double quote.