SETL Newsletter # 119

October 25, 1973
J. Schwartz

A Suggested Generalisation and Revision of the SETL Compound Operator Form.

The present SETL compound operator is at times irritatingly clumsy, and for this reason generalisations and modifications have been suggested from time to time. I would like to suggest such a modification, which seems to me to be advantageous. The proposed form is

(1) f[*expn: iterator*],

where *expn* is an expression $e(x_1,\ldots,x_n)$ involving free variables $x_1,\ldots,x_n$, and *iterator* is an iterator, such as

(2) $x \in e_1$, $\min(x_1) \leq x_2 \leq \max(x_1)$, $x_3 \in e_3(x_1,x_2)$,... built upon these same variables. Moreover (and this involves a semantic step beyond the present compound operator form) f is either a pair

(3) < binaryoperator, initialobject >

or a triple

(4) < binaryoperator, initialobject, monadicmapping >.

If f has the form (4), (1) signifies the value *val* calculated by the code

```
val = initialobject;
( ∀ iterator )
        val = binaryoperator(val, monadicmapping(e(x1,...,xn)))
  and ∀;
```

If f has the simple form (3), then *monadicmapping* is understood to be the identity map. We allow any expression with a suitable pair or triple as value to appear in place of *f* in (1).
Some examples:

If *makeunitset* always maps x into {x}, then the SETL set-former {*expn, iterator* } is defined by

(5) < +, nl, makeunitset > [expn; iterator],

and can be understood as a syntactic abbreviation for (5). The (presently missing, and sometimes missed) 'tuple former' which corresponds to the set former can be written

(6) < +, nult,makeunittuple > [expn: iterator]

where *makeunittuple* maps x into <x>. For (6), the abbreviation

(7) < expn: iterator >

might reasonably be allowed.

The mathematical $\prod_{2=1}^{n} a(i)$ would be

< *,1 > [a(i): 1 ≤ i ≤ n],

or, if prod = < *,1 >,would be

prod [a(i): 1 ≤ i ≤ n].

This revision might be of substantial benefit in connection with a larger family of user-definable object types, in that it defines a syntactic and semantic framework allowing compound-object formers of new, user specified types to be introduced.

Note that from a deeper point of view the virtue of the compound operator is that it makes certain iterative constructions usable directly as expressions.