SETL Newsletter # 138A

J.T.Schwartz
August 12, 1974

Optimisation by Set Suppression

The 'iterator reduction'optimisation discussed in NL 138 is one of a class of 'algorithmic' optimisations; these are optimisations of a level high enough to be confortably expressible by SETL → SETL code transformations rather than by transformations applied during the translation of a SETL program into programs written in a less abstract language. There undoubtedly exist many optimisations of this class, and, even though many of them will occur too infrequently to justify their automatic treatment, they are worth recording, evern if only as hints to the algorithm designer. Newsletter 138 initiated, and the present newsletter continues, the task of recording these optimisations. In the present newsletter we focus our interest on situations in which a set s is formed simply to support some subsequent iteration, and where in fact the formation of s can be suppressed. If s is a large set, this may be quite advantageous.

A typical instance of this general situation might initially appear as follows: Suppose that a set s is formed by a body of code

(1)

```
...
s = nl;
(while ...)
        ...
        /* a quantity x has been calculated *
        x in s;
        ..
end while;
...
```

and then used in an iteration

(2)
```
...
(∀x∈s | C(x)) block end ∀;
...
```

Suppose also

a) that execution of the code (1) has no side effects; more precisely, that the only data item transmitted from the code (1) to the remainder of the program P in which (1) is embedded is the set s;

b) That, aside from its use in the iteration (2), s has no other use in P;

c) That the objects x sucessively added to s may be shown to be distinct from each other;

d) That no data object used in the code (1) is modified until after the iteration (2) is complete.

Then (1) and (2) may be transformed into equivalent code which does not require the set s to be formed expliclitly or stored. This is done as follows:

i. Move (1) to the position of (2), modifying both so that together they read

(3)
```
...
(while ...)
        ...
        /* a quantity x has been calculated */
        if C(x) then block; /* replaces x in s */
        ...
        ...
end while;
```

Suppose next that s appears in several iterations

(4) $(\forall x \in s \mid C_j(x))$ *block* end ∀; (j = 1, ...,n),

but that conditions (a) and (c) remain true. Suppose also that none of the iterations (4) modifies a variable used either by the code (2) or in some other iteration (4); and suppose that aside from its appearance in the iterations (4) s has no use. Then (1) may be combined together with all the iterations (4) into a body of code which has the following appearance

(5)

```
...
(while ...)
      ...
      /* a quantity x has been calculated */
      if C1(x) then block1;
      if C2(x) then block2;
         ...
      if Cn(x) then blockn;
      ...
end while;
```

Loops having the form (5) commonly occur in compilers, and may be considered to have an origin like that which has just been described.