This newsletter is a sequel to SETL Newsletter #102 ("Reduction in Strength Using Hashed Temporaries"). The reader is referred to that newsletter for a description of the intermediate code and macros assumed in this discourse.

The idea of test replacement is to allow the elimination of computations whose usefulness ends with strength reduction. For example, consider the loop:

\[
\begin{align*}
i &= 1 \\
\text{start: } x &= i \cdot c \\
\vdots \\
i &= i + 2 \\
\text{if } i \leq 100 \text{ go to start}
\end{align*}
\]

After strength reduction, this becomes:

\[
\begin{align*}
i &= 1 \\
t_i \cdot c &= i \cdot c \\
t_2 \cdot c &= 2 \cdot c \\
\text{start: } x &= t_i \cdot c \\
\vdots \\
i &= i + 2 \\
t_i \cdot c &= t_i \cdot c + t_2 \cdot c \\
\text{if } i \leq 100 \text{ go to start}
\end{align*}
\]

Now if \( i \) is dead on exit from the loop, as is the case with many induction variables, we can eliminate the instruction which increments \( i \) if we can eliminate the test of \( i \). We do this by testing \( t_i \cdot c \) instead, yielding the following loop.

\[
\begin{align*}
i &= 1 \\
t_i \cdot c &= i \cdot c \\
t_2 \cdot c &= 2 \cdot c \\
t_{100} \cdot c &= 100 \cdot c \\
\text{start: } x &= t_i \cdot c \\
\vdots \\
t_i \cdot c &= t_i \cdot c + t_2 \cdot c \\
\text{if } t_i \cdot c \leq t_{100} \cdot c \text{ go to start}
\end{align*}
\]
We have thus eliminated one instruction within the loop. This then is the purpose of test replacement.

The method we present here replaces all tests of induction variables $i$ with tests of temporaries $t_{i*c}$ if such a temporary exists. The useless increments will then be removed by a systematic dead computation elimination algorithm to be described in a later newsletter.

Suppose the test is of the form:

```plaintext
if $i > c$ go to label
```

We must do three things:

1. we must insert $t_{c*l}$ in the table of hashed temporaries;
2. we must initialize $t_{c*l}$ in the prolog of the strongly-connected region, and
3. we must replace the instruction by

```plaintext
if $t_{i*c} > t_{c*l}$ go to label
```

Note that steps (1) and (2) need only be undertaken if there is no temporary $t_{c*l}$.

The following SETL function performs steps (1) and (2) and returns the ordered pair of arguments for the modified branch instruction.

```plaintext
definef treplace (i,c,cl,prolog,plast,t);
/* i is the induction variable,
   c is the constant multiplier for which temporary $t(i,c)$ exists,
   cl is the comparison constant in the test,
   prolog is the set of instructions which are executed prior
to entry to the current scr,
   plast is the last instruction in the prolog,
   t is the table of temporaries */
/* first see if $t(cl,c)$ is in the temporary table */
if t(cl,c) $\in$ $\Omega$ then /* create new entry */
   t(cl,c) = newtemp; t(c,cl) = t(cl,c);
```
SETL 107-3

/* insert initialization in prolog */
insert(plast,t(cl,c), mul, <cl,c>, prolog);
/* the insert routine is defined in newsletter #102 */
end if;
/* now return new arguments for conditional branch */
return <t(i,c), t(cl,c)>;
end treplace;

Now the test replacement merely consists of searching the code in the strongly-connected region $scr$ for conditional branch instructions (brc) which compare an induction variable to a region constant. If the induction variable $i$ has an associated temporary $t_{i,c}$ in the region we can perform the reduction.

In order to determine if there is such a temporary we need a list of induction variables and associated constants for which temporaries have been created. Recall that the set $aands$ from SETL Newsletter #102 is the set of instructions which will be eliminated by reduction in strength. After the candidates are found we can form a list of induction variables and constants by inserting the instruction

$$ctemps = args[cands]$$

The set $ctemps$ will then contain all pairs $<i,c>$ for which a temporary $t_{i,c}$ is created by reduction in strength. With the availability of this set the test replacement algorithm looks like this.

define testreplace(scr,iv,rc,ctemps,prolog,plast,t);
/* $scr$ is the region being considered,
   $iv$ is the set of induction variables,
   $rc$ is the set of region constants,
   $ctemps$ is the list of constant multipliers of induction variables,
   $prolog$ is the prolog of the current $scr$,
   $plast$ is the last instruction of prolog,
   $t$ is the table of temporaries */
This routine can be incorporated into the strength reduction algorithm presented in SETL Newsletter #102 to produce the following SETL routine:

```setl
define streduce(prolog,plast,scr,rc);
/* prolog is the initialization block whose last instruction is plast,
   scr is the region and rc are the region constants which
   we assume are found in an earlier code-motion pass */
/* find induction variables */
   <iv,ivnodes> = findivars(scr,rc);
/* find candidates for reduction */
cands = findcands(scr,rc,iv);
/* create constant multipliers list for test replacement */
ctemps = args[cands];
/* find the affect relation */
affect = findaffect(ivnodes,iv,rc);
/* now pass through the candidates creating temporaries and
   inserting initializations and modifications */
```
(\forall at \in \text{cands}) \; x = \text{arg1}(at); \; c = \text{arg2}(at);

/* create the new temporaries as required */

(\forall y \in \text{affect}{x} \mid t(y,c) = \Omega)

\; t(y,c) = \text{newtemp}; /* compiler generated name */

/* initialization in prolog */

insert(plast,t(y,c),mul,<y,c>, prolog);
plast = next(plast);

/* double entries for const * const */

if \; y \in rc then \; t(c,y) = t(y,c);;

/* insert modifications to the new temporaries after 
instructions which set induction variables */

(\forall n \in \text{ivnodes} \mid \text{targ}(n) = y)

\; \text{newargs} = \text{if pair args}(n)

\; \text{then} \; <t(\text{arg1}(n),c),t(\text{arg2}(n),c)>

\; \text{else} \; <t(\text{arg1}(n),c)>;

/* the inserted instruction has the target \; t(y,c), the same 
operations as \; n, and \; \text{newargs} as its argumentns */

insert(n,t(y,c), op(n), newargs, scr);

end \; \forall n;

end \; \forall y;

/* now replace the candidate by a store operation */

<\text{op}(at),\text{args}(at)> = <\text{sto}, \; t(x,c)>;

end \; \forall at;

/* call test replacement program */
testreplace(scr,iv,rc,ctemps,prolog,plast,t);
end streduce;

The test replacement routine does not eliminate actual increments
of induction variables. It will, however, make many of these
increments "useless" and they can be eliminated by a dead
computation elimination algorithm. Such an algorithm will be
described in a later newsletter.

Acknowledgement: The author is grateful to John Cocke
for suggesting the methods described in this newsletter
and SETL Newsletter No. 102.