SETL Newsletter 107 May 20, 1973

Linear Function Test Replacement Ken Kennedy

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:

```
        i = 1
start:  x = i*c
          ⋮
        i = i + 2
        if i ≤ 100 go to start
```

After strength reduction, this becomes:

$$
\begin{array}{ll}
 & i = 1 \\
 & t_{i*c} = i*c \\
 & t_{2*c} = 2*c \\
\text{start:} & x = t_{i*c} \\
 & \vdots \\
 & i = i+2 \\
 & t_{i*c} = t_{i*c} + t_{2*c} \\
 & \text{if } i \le 100 \text{ go to start}
\end{array}
$$

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*c} instead, yielding the following loop.

$$
\begin{array}{ll}
 & i = 1 \\
 & t_{i*c} = i*c \\
 & t_{2*c} = 2*c \\
 & t_{100*c} = 100*c \\
\text{start:} & x = t_{i*c} \\
 & \vdots \\
 & t_{i*c} = t_{i*c} + t_{2*c} \\
 & \text{if } t_{i*c} \le t_{100*c} \text{ go to start}
\end{array}
$$

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:

$$\text{if } i \geq cl \text{ go to label}$$

We must do three things:

(1) we must insert t_{cl*c} in the table of hashed temporaries;

(2) we must initialize t_{cl*c} in the prolog of the strongly-connected region, and

(3) we must replace the instruction by

$$\text{if } t_{i*c} \geq t_{cl*c} \text{ go to label}$$

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

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

```
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) eq Ω then /* create new entry */
   t(cl,c) = newtemp;  t(c,cl) = t(cl,c);
```

```
    /* 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 *cands* 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        */
```

```
    /* search for replaceable tests */
    (∀a ∈ scr | op(a) eq brc)
    if((arg1(a) is i) ∈ iv) and
      ((arg2(a) is cl)∈ rc) and
      (∃c ∈ ctemps{i})
    then
      <arg1(a),arg2(a)> = treplace(i,c,cl,prolog,plast,t);
    else if((arg1(a) is cl) ∈ rc) and
           ((arg2(a) is i)  ∈ iv) and
           (∃c ∈ ctemps{i})
    then
      <arg2(a),arg1(a)> =treplace(i,c,cl,prolog,plast,t);
    end if;
    end  testreplace;
```

This routine can be incorporated into the strength reduction algorithm presented in SETL Newsletter #102 to produce the following SETL routine:

```
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 */
```

```
(∀at ∈ cands) x = arg1(at);  c = arg2(at);
/* create the new temporaries as required */
    (∀y ∈ affect{x} | t(y,c) = Ω)
       t(y,c) = 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 ∈ rc then t(c,y) = t(y,c);;
/* insert modifications to the new temporaries after
    instructions which set induction variables */
    (∀n ∈ ivnodes | targ(n) eq y)
       newargs = if pair args(n)
          then <t(arg1(n),c),t(arg2(n),c)>
          else <t(arg1(n),c)>;
/* the inserted instruction has the target t(y,c), the same
    operations as n, and newargs as its argumetns */
          insert(n,t(y,c), op(n), newargs, scr);
       end ∀n;
   end ∀y;
/* now replace the candidate by a store operation */
   <op(at),args(at)> = <sto, t(x,c)>>;
end ∀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.