

A Coarser, But Simpler and Considerably
More Efficient Copy Optimization Technique

The dynamic 'shared bit' copy elimination method envisioned for the new SETL compiler can be improved by adapting rather standard global optimization techniques. The present newsletter will outline these adapted techniques, which allow cases in which objects can be used destructively to be found by a highly efficient algorithm, although one that is not quite as precise as the 'valueflow function' method described in newsletter 164. The approach to be outlined in the present newsletter also allows us to detect cases in which copy optimizations can be moved out of loops, a question recently discussed by Schonberg and Liu in NL 175.

In the currently envisaged dynamic method, every SETL value c will carry a share bit b , which will be set whenever c can be the value, or part of the value, of more than one variable. The following rules are used to maintain share-bit settings:

- i.* On an assignment $x = y$ for which y is not dead, set the share bit of the value of y before transmitting it to x .
- ii.* When y is incorporated into a larger object s , e.g., by one of the operations s with y ; $s(x) = y$; $s(y) = x$ (if s is a map); $\{y, \dots\}$; or $\langle \dots, y, \dots \rangle$, set the share bit of y (unless y is dead.) Moreover, let the copy of the specifier for y which becomes incorporated into the compound objects s have its share bit set (so that when this same value is subsequently retrieved from y , the share bit of the retrieved variant will already be set).
- iii.* When y is transmitted to a subprocedure f as a read/write parameter, set the share bit of y if either y is simultaneously transmitted as the value of some other parameter,

or if y is global and accessed by some procedure which might be called by f , or if y is static and local to the procedure g calling f , but g might be called directly or indirectly by f .

iv. If y is transmitted to a subprocedure f as a read-only parameter, set the share bit of y if either y is simultaneously transmitted as the value of a write parameter, or if y is writable, global and accessed for writing by some procedure which might be called by f , or if y is static and local to and writable within the procedure g calling f , but g might be called directly or indirectly by f .

Suppose now that for each variable y we introduce an explicit logical 'shadow' variable $yshare$ whose value is always '1' when the share bit of y is set. Then this variable undergoes the following assignments:

i'. On a simple assignment $x = y$, set $yshare = 1$ unless y is dead, and $xshare = 1$. On an assignment $x = expn$, set $xshare = 1$ if $expn$ is a value-retrieving expression like $\exists s$ or $s(l)$, but set $xshare = 0$ if $expn$ is a value creating expression like $\{y\}$ or $y + z$.

ii'. When y is incorporated into a compound object s , e.g. by s with y or $s(x) = y$, execute $yshare = 1$.

iii'. When y is transmitted as an argument, execute $yshare = 1$ if this is indicated by rules (i-iv) above, and transmit $yshare$ as a read-write argument to the shadow share-bit variable associated with the parameter to which the argument y is being transmitted.

The logical assignments to the variables $yshare$ introduced by the preceding rules can be applied to calculate many of the values of this variable; for example, a standard constant propagation technique can be used. Every potentially destructive use of y should be regarded as a use of $yshare$ (which needs to be tested at the point of destructive use); live/dead analysis, applied to the share-bit variables, can also reveal that certain assignments to these bits are dead.

The following additional possibility is particularly significant:

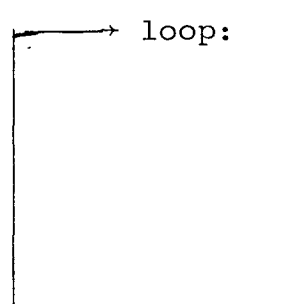
If we use an interval-based technique to calculate share-bit values, we will be able to classify every use of a share-bit variable into one of the following four categories; definitely 0; definitely 1; inherently indefinite, indefinite but would definitely be 0 if it were 0 on entry to the interval containing the occurrence. With every share-bit use of the fourth category, we can associate the entry block of the largest interval I (somewhere in a full program graph derivation sequence) such that the share-bit value would be zero within I if it were 0 on entrance to I. Then the corresponding value can be copied before entrance to I, and need never be copied within I.

The following example, which played a role in the evolution of the valueflow function method, illustrates the remarks made in the preceding paragraph:

```

                                s = t;    /* sshare is 1 here */
                                ...
                                ...
                                loop:
                                ...
                                ...    /* sshare is indefinite here, but */
                                ...    /* would be zero if it were 0 on loop */
                                ...    /* entrance */
                                s=s with x;
                                ...    /* sshare is 0 here */

```



Our analysis reveals that in this case the copy operation should be moved out of the loop.