SETL Newsletter # 164A. J.T. Schwartz
April 7, 1976

'Copy on Assignment' Optimization in SETL.

This newsletter responds to NL 164 (by R. Dewar) in which it is observed that

a. A shared bit is useful even in the presence of global copy analysis, and inexpensively implementable;

b. The shared bit mechanism essentially dominates the copy optimization scheme presently under development, reducing its effect to a modest 'eliminate tests' level;

c. More useful information can be gathered by calculating assignment- related (or, more generally, 'incorporation'-related, see below) information.

We shall sketch a method for gathering this information.

Definitions: A simple assignment a = x, or a use of x which makes its value part of a larger composite object (e.g. a = {x}, or a = <x,y>, or a(i) = x) is called an *incorporation* of x. A potentially destructive use of x, e.g. a = x with y, is called a (potential) *modification* of x.

Any incorporation of x (other than the atypical case of a simple assignment) forms a new object a almost any of whose subsequent uses is likely either to reference the object x or to create a reference to x. Thus the shared bit of x will have to be set on incorporation unless x is close to being dead at its point of incorporation. Actually, only a somewhat weaker condition needs to be imposed, namely that no incorporation or modification of x can be reached by going forward from this point, without a reassignment of x being encountered first.

This condition can be calculated using essentially the standard 'live variable' technique. Call a use of x an *miuse* (modifying or incorporating use) if it is either a modification or an incorporation of x; and call x *milive* at a point p if there exists a path forward from p to an miuse which goes through no assignment to x. Then if we apply a standard 'live' algorithm simply ignoring uses of x other than miuses, milive information will be obtained.

For the case of a simple assignment, which is symmetric in its left and right sides, the condition that both right and left hand variables are live should be used.

In NL 164, an additional distinction is suggested. If at one of its incorporations the variable x is not only live but *lively*, in the sense that every path forward from the incorporation must encounter an miuse before an assignment to x or a program exit, then it is better to copy x at the incorporation point than to set the shared bit, since copying is inevitable and by not setting the shared bit we avoid creation of an object that may force multiple subsequent copying. The condition that x be lively can be computed by an easy algorithm, having exactly the live variable structure, but dual to it, in the following sense: treat assignment to x and program exits as if they were uses of x, and miuses of x as if they were assignments; then apply a standard 'live' algorithm which will calculate a condition c at each program point. The boolean negative of this condition c is the condition that x be lively.

The test-elisions on modification suggested in NL 164 can be made available using the *orthis* functions in the following way. Suppose that every assignment has already been classified as a 'set shared bit' or a 'dont set bit' assignment.

Suppose also that immediately following each incorporation, e.g., a = {x}, at which the shared bit must be set, we insert an auxiliary'special assignment' x = x, and that the *crthis* function is computed after these auxiliary assignments have been inserted. Then, given any ivariable occurence i of x which may be a modification of x, we look back to all the ovariables in crthis(i). If all of these will have set the shared bit of their ovariable, then x needs to be copied unconditionally; if none of them will have set the shared bit of their ovariable, the copying can be avoided unconditionally; and if some but not all of them will have set the shared bit, then the shared bit must be tested.

Variations in the Presence of Basing.

A set that has been declared as a base cannot be copied when modified, since all the other objects declared to be based on it must always point to the current copy of the base. Thus a set declared as a base can never be shared. If a base set b is incorporated into another object, then copying can only be avoided if all the objects based on b are dead at the point of incorporation. It should also be noted that since the pattern of incorporations which set share bits is changed(more specifically, diminished) when some of the sets in a program are declared to be bases, such declarations can also diminish the set of ivariable uses at which copies are necessary.