1. COPY OPTIMIZATION IN SETL

SETL NEWSLETTER NUMBER 164

ROBERT B. K. DEWAR

4/13/76

THIS NEWSLETTER DISCUSSES THE PROBLEM OF OPTIMIZING COPIES IN SETL (I.E. PREVENTING COPIES WHERE THEY ARE UNECESSARY). IT IS ARGUED THAT A STATIC APPROACH IS INSUFFICIENT AND THAT THE USE OF REFERENCE COUNTS IS REQUIRED EVEN IF THE OPTIMIZER PROVIDES MAXIMUM INFORMATION. AN EFFICIENT APPROACH TO THE MAINTENANCE OF REFERENCE COUNTS IS DESCRIBED. THE INFORMATION REQUIRED FROM THE OPTIMIZER TO REDUCE COPIES WITH REFERENCE COUNTS IN USE IS ALSO DESCRIBED.

1.1. MODELS OF REQUIRED COPIES

THE THREE KINDS OF STATEMENTS IN SETL WHICH ARE INVOLVED IN DISCUSSION OF COPYING ARE:

1) SIMPLE ASSIGNMENTS, INCLUDING ACQUIRING A VALUE AND MAKING IT A PART OF A COMPOSITE.

2) ASSIGNMENTS TO SUBPARTS OF A COMPOSITE. WE WILL CALL THESE MODIFICATIONS.

3) COMPUTATIONS WHICH INVOLVE MODIFYING AN OPERAND. FOR EXAMPLE A = B WITH C.

   THIS THIRD TYPE WILL BE CONSIDERED A COMBINATION OF THE FIRST TWO BY WRITING A = B WITH C AS:

```
A = B           /* AN ASSIGNMENT */
A WITH C        /* A MODIFICATION */
```

THUS WE HAVE TO DEAL WITH ASSIGNMENTS AND WITH MODIFICATIONS.

THERE ARE TWO CANONICAL WAYS OF IMPLEMENTING OR DESCRIBING THE COPYING WHICH MUST BE PERFORMED AS FOLLOWS:

1) COPY ON ALL ASSIGNMENTS. NEVER COPY ON MODIFICATIONS.

2) COPY ON ALL MODIFICATIONS. NEVER COPY ON ASSIGNMENTS.

NOTE THAT SINCE A = B WITH C INVOLVES BOTH AN ASSIGNMENT AND A COPY, THAT EITHER OF THESE APPROACHES LEADS TO A COPY IN THIS INSTANCE.

UNDER THE CONSTRAINT OF IMPLEMENTING COPYING WITH ONE OF THESE CANONICAL APPROACHES, THERE ARE PROGRAMS WHICH CLEARLY FARE BETTER WITH ONE APPROACH OR THE OTHER:

```
(@1<N<100) A=B;; A(X)=B; $ FASTER WITH METHOD 2

A=B; (@1<N<100) A(X)=B;; $ FASTER WITH METHOD 1
```

AND THERE ARE PROGRAMS WHERE IT IS IMPOSSIBLE TO TELL:

```
(@1<N<J) A=B;; (@1<N<K) A(X)=B;; $ J>K???
```

THIS LAST POSSIBILITIY SHOWS THE DIFFICULTY OF MAKING A CLEAR CHOICE BETWEEN THESE TWO METHODS ON THE BASIS OF THE PROGRAM AT HAND. MAKING A CHOICE THROUGHOUT THE SYSTEM DEPENDS ON THE EMPIRICAL QUESTION OF WHICH KIND OF PROGRAM IS MORE COMMON.

1.2. REFERENCE COUNTS

THE CURRENT SYSTEM USES REFERENCE COUNTS TO DECREASE THE NUMBER OF COPIES REQUIRED. THERE ARE SOME ERRORS IN THE WAY THIS IS DONE, BUT THESE COULD BE FIXED SO THAT THE SCHEME WOULD BE WATERTIGHT.

THERE ARE TWO WAYS OF LOOKING AT THE USE OF REFERENCE COUNTS:

1) BASICALLY COPIES ARE DONE ON ASSIGNMENTS. HOWEVER, THE COPY IS DELAYED BY MANIPULATING REFERENCE COUNTS TO INDICATE THAT A COPY SHOULD REALLY HAVE BEEN PERFORMED. IF IT IS DISCOVERED LATER THAT THE COPY SHOULD HAVE BEEN DONE, THEN IT IS DONE AT THE POINT OF DISCOVERY.

THUS WE HAVE COPY ON ASSIGNMENTS, BUT WITH A MECHANISM FOR AVOIDING COPIES WHICH ARE NOT NECESSARY.

2) BASICALLY COPIES ARE DONE ON MODIFICATIONS. HOWEVER, REFERENCE COUNTS ARE KEPT SO THAT AT THE POINT OF MODIFICATION IT IS POSSIBLE TO DETECT THE FACT THAT THE COPY IS NOT REQUIRED IN THIS CASE.

THESE TWO VIEWS ARE EQUIVALENT, BUT IT IS USEFUL TO DISTINGUISH THEM WHEN IT COMES TO CONSIDERING THE FURTHER EFFECT OF THE OPTIMIZER.

1.3. SHARED BIT MECHANISM

THE MECHANISM PROPOSED FOR THE NEW SETL INTERPRETOR (NON-OPTIMIZED VERSION) USES SHARE BITS INSTEAD OF REFERENCE COUNTS.

THE SHARED BIT IS A REFERENCE COUNT WHICH HAS VALUES OF EITHER 1 OR 0. IT HAS A SIMILAR FUNCTION TO THE USE OF FULL REFERENCE COUNTING, BUT SINCE 1 IS THE MAXIMUM VALUE (AND MIGHT MEAN MORE THAN 1 IN FACT), IT IS NEVER POSSIBLE TO REDUCE REFERENCE COUNTS (TURN OFF THE SHARED BIT). THIS MEANS THAT IN A SEQUENCE:

```
A = B;
A(1) = 1;
B(1) = 1;
```

THE SECOND AND THIRD LINES BOTH PERFORM COPIES WHERE A FULL REFERENCE COUNTING SYSTEM WOULD ONLY NEED THE SECOND COPY (ASSUMING NO OTHER USES OF THE VALUE ARE ACTIVE).

IN GENERAL THE SHARED BIT MECHANISM MAY CAUSE AS MANY AS TWICE THE NUMBER OF COPIES THAT A FULL REFERENCE COUNT MECHANISM WOULD REQUIRE. IN PRACTICE THE NUMBER OF COPIES IS SOMEWHERE BETWEEN ONE AND TWO TIMES THIS NUMBER.

THE ADVANTAGE OF THE USE OF SHARED BITS IS THAT THEY CAN BE MANIPULATED VERY MUCH FASTER THAN REFERENCE COUNTS BY USING THE FOLLOWING SCHEME.

NORMALLY REFERENCE COUNTS ARE KEPT WITH THE ACTUAL VALUE WHICH MIGHT GET COPIED. THIS IS DONE SO THAT A REFERENCE WHICH BECOMES DISCONNECTED (BY EITHER REASSIGNMENT OR COPYING) MUST REDUCE THE COUNT BY ONE. USING THE SHARED BIT MECHANISM, THE COUNT IS NEVER REDUCED. THIS MEANS THAT IT CAN BE KEPT IN THE REFERENCES THEMSELVES. SINCE THE SHARED BITS ARE NEVER TESTED ON SHORT ITEMS AND HAVE NO EFFECT, THEY MAY BE SET WITHOUT TESTING TO SEE WHETHER AN ITEM IS SHORT OR LONG.

THE SEQUENCE FOR AN ASSIGNMENT A = B BECOMES:

```
(SET SHARED BIT OF VALUE SPECIFIER FOR B)
(ASSIGN B TO A, I.E. JUST MOVE SPECIFIER)
```

ON THE IBM 370, THE INSTRUCTION SEQUENCE IS:

```
OI    (SHARED BIT OF B)       *******
LM    1,2,B
STM   1,2,A
```

ON THE CDC 6600, THE INSTRUCTION SEQUENCE IS (ASSUMING THE SHARED BIT TO BE THE LAST BIT OF THE SPECIFIER):

```
SA1   B
SX2   B7                      *******
IX6   X1+X2
SA6   A1                      *******
SA6   A
```

IN EACH CASE, THE ASTERISKS FLAG THE EXTRA INSTRUCTIONS FOR

SETTING THE SHARED BIT COMPARED WITH A STRAIGHT ASSIGNMENT. THE COST IS SMALL IN TIME AND THE RESULTING CODE IS STILL SHORT ENOUGH TO BE GENERATED INLINE.

THIS SEQUENCE IS TO BE COMPARED WITH SOMETHING LIKE THE FOLLOWING FOR FULL REFERENCE COUNTS:

```
        (IF A IS SHORT ITEM, GOTO L1)
        (IF REF COUNT OF VALUE OF A = MAX, GOTO L1)
        (DECREMENT REFERENCE COUNT OF VALUE OF A)
  L1:   (IF B IS SHORT ITEM, GOTO L2)
        (IF REF COUNT OF VALUE OF B = MAX, GOTO L2)
        (INCREMENT REFERENCE COUNT OF VALUE OF B)
  L2:   (COPY VALUE SPECIFIER OF B TO A)
```

NOT ONLY IS THIS SEQUENCE MUCH SLOWER AND MORE COMPLEX, BUT IT IS QUITE IMPRACTIAL TO GENERATE INLINE CODE FOR SUCH A COMPLICATED SEQUENCE.

SINCE THE NUMBER OF ASSIGNMENTS IS VERY MUCH GREATER THAN THE NUMBER OF COPIES REQUIRED BY A REFERENCE COUNT SYSTEM, THE USE OF SHARED BITS IS THOUGHT TO BE MUCH MORE EFFICIENT.

1.4. GLOBAL OPTIMIZATION

WE NOW CONSIDER THE INFORMATION WHICH COULD BE PROVIDED BY A GLOBAL OPTIMIZATION SCHEME TO HELP REDUCE THE NUMBER OF COPIES.

AN INITIAL APPROACH IS TO ASSUME ONE OF THE ORIGINAL CANONICAL SCHEMES TOGETHER WITH AN ATTEMPT TO IDENTIFY STATICALLY THOSE INSTANCES WHERE COPYING CAN ALWAYS BE OMITTED.

ASSUMING THE APPROACH OF COPYING ON ASSIGNMENTS, THE CONDITIONS FOR OMITTING THIS COPY ARE AS FOLLOWS:

A = B

COPY CAN BE OMITTED IF ANY OF THE FOLLOWING HOLD:

1) B IS DEAD

2) THE VALUE B IS ABANDONED BEFORE A IS MODIFIED

3) THE VALUE A IS ABANDONED BEFORE B IS MODIFIED

4) THE VALUES OF A AND B ARE BOTH ABANDONED.

AN ABANDED VALUE IN THIS SENSE IS ONE WHICH BECOMES DEAD WITHOUT BEING MODIFIED OR SHARED BY ANOTHER POINTER.

THESE CONDITIONS ARE DETECTABLE BY SUITABLE FLOW ANALYSIS AND ASSIGNMENTS MEETING THESE CONDITIONS COULD BE FLAGGED.

IF THE APPROACH OF COPYING ON MODIFICATIONS IS ADOPTED, THEN THE CONDITIONS FOR OMITTING THE COPY ARE AS FOLLOWS:

F(A) = B

COPY CAN BE OMITTED IF:

1) THERE IS NO OTHER LIVE USE OF THE VALUE F AT THE TIME THE ASSIGNMENT IS EXECUTED.

AGAIN THIS INFORMATION CAN BE OBTAINED BY FLOW ANALYSIS AND IN FACT THIS APPROACH IS THE ONE THAT HAS BEEN EXAMINED IN PREVIOUS WORK ON COPY OPTIMIZATION IN CONJUNCTION WITH SETL.

NEITHER OF THESE SCHEMES IS SATISFACTORY AS CAN BE SEEN FROM THE PROGRAM EXAMPLES GIVEN EARLIER:

(@1<N<100) A=B;; A(X) = C

A=B IS NOT AN ASSIGNMENT FOR WHICH THE COPY CAN BE OMITTED STATICALLY SINCE IT IS NEEDED SOME OF THE TIME (THE LAST TIME THROUGH THE LOOP).

A=B; (@1<N<100) A(X)=C;;

A(X)=C IS NOT A MODIFICATION FOR WHICH THE COPY CAN BE OMITTED STATICALLY SINCE IT IS NEEDED SOME OF THE TIME (THE FIRST TIME THROUGH THE LOOP).

ALTHOUGH THESE PARTICULAR EXAMPLES COULD BE HANDLED BY SPECIAL ANALYSIS WHICH OPENED UP THE LOOP TO TREAT THE FIRST (OR LAST) ITERATION SPECIALLY, IT IS CLEARLY POSSIBLE TO CONSTRUCT EXAMPLES WHERE NO SPECIAL STATIC TREATMENT WOULD BE ADEQUATE.

IT IS THUS CLEAR THAT THE SHARED BIT MECHANISM IS REQUIRED EVEN IF THE OPTIMIZER PROVIDES THE BEST POSSIBLE STATIC COPY INFORMATION. WE THEREFORE CONSIDER THE INTERACTION BETWEEN THE STATIC COPY INFORMATION AND THE SHARED BIT MECHANISM.

FOR A FIRST STAGE, LET US SUPPOSE THAT THE OPTIMIZER FLAGS ASSIGNMENTS WHERE THE COPY IS NOT REQUIRED (UNDER THE ASSUMPTION OF COPYING ON ASSIGNMENTS) AND ALSO FLAGS MODIFICATIONS WHERE THE COPY IS NOT REQUIRED (UNDER THE ASSUMPTION OF COPYING ON MODIFICATIONS). WE THEN HAVE THE FOLLOWING:

FOR ASSIGNMENT: IF THE ASSIGNMENT IS FLAGGED AS NOT REQUIRING A COPY, THEN THE SETTING OF THE SHARED BIT IS OMITTED SINCE IT COULD NOT BE NEEDED. THIS

MIGHT SAVE A COPY WHICH WOULD OTHERWISE BE PERFORMED UNECESSARILY.

FOR MODIFICATION: IF THE MODIFICATION IS FLAGGED AS NOT REQURING A COPY, THEN THE SHARED BIT NEED NOT BE TESTED. THIS IS STRICTLY A LOCAL HELP AND SAVES ONLY THE TEST (SINCE THE RESULT OF THE TEST, IF PERFORMED, WOULD ALWAYS INDICATE THAT NO COPY WAS REQUIRED).

A MORE SOPHISTICATED SCHEME IS AS FOLLOWS:

FOR ASSIGNMENTS, THE OPTIMIZER DISTINGUISHES THREE CASES:

1) A COPY IS REQUIRED SINCE ALL PATHS FROM THE ASSIGNMENT INVOLVE MODIFYING ONE OF THE VALUES WHILE THE OTHER IS STILL LIVE. IN THIS CASE, THE COPY IS ACTUALLY DONE ON THE ASSIGNMENT SINCE THE SHARED BIT MECHANISM CANNOT PREVENT IT AND MIGHT CAUSE A SECOND UNECCESSARY COPY IF BOTH VALUES ARE MODIFIED.

2) A COPY IS DEFINITELY NOT REQUIRED. AS ABOVE, WE SIMPLY OMIT THE SETTING OF THE SHARED BIT.

3) A COPY MAYBE REQUIRED DEPENDING ON WHICH OF SEVERAL PATHS IS TAKEN (OR THERE IS A PATH FOR WHICH COMPLETE INFORMATION IS NOT AVAILABLE). IN THIS CASE, THE SHARED BIT MECHANISM IS USED.

IT IS ALSO POSSIBLE TO DISTINGUISH THE THREE CASES FOR MODIFICATIONS:

1) COPY IS NEVER REQUIRED SINCE ALL PREVIOUS ASSIGNMENTS WERE TYPE 1) (COPY REQUIRED). IN THIS CASE, THE TEST OF THE SHARED BIT TO SEE WHETHER A COPY IS REQUIRED CAN BE OMITTED AND THE OPERATION PERFORMED WITHOUT A TEST.

2) COPY IS ALWAYS REQUIRED SINCE ALL PREVIOUS ASSIGNMENTS SET THE SHARED BIT. IN THIS CASE, THE TEST CAN BE OMITTED AND A COPY PERFORMED.

3) COPY MAYBE REQUIRED SINCE THE STATE OF THE SHARED BIT CANNOT BE PREDICTED.

1.5. RELATIVE VALUE OF INFORMATION

THERE ARE TWO COMPLETELY DIFFERENT COPY OPTIMIZATIONS WHICH THE OPTIMIZER IS THUS CALLED UPON TO PERFORM:

1) ANALYZE ASSIGNMENTS

2) ANALYZE MODIFICATIONS

THE FIRST OF THESE IS MUCH MORE IMPORTANT SINCE THE FLAGGING OF ASSIGNMENTS (EITHER BY THE SIMPLE TWO CASE MECHANISM OR THE THREE CASE MECHANISM) CAN SAVE COPY OPERATIONS AS WELL AS SAVING INLINE CODE FOR ASSIGNMENT OPERATIONS.

THE ANALYSIS OF MODIFICATIONS IS MUCH LESS IMPORTANT SINCE AT BEST IT SAVES SIMPLE TESTS. FURTHERMORE, IT IS UNLIKELY THAT MODIFICATION OPERATIONS CAN BE PERFORMED IN LINE (WITH THE POSSIBLE EXCEPTION OF ASSIGNMENTS TO TUPLES WHICH ARE KNOWN TO BE TUPLES), WHICH MEANS THAT INLINE CODE WILL NOT BE SAVED.

MOST OF THE EFFORT SO FAR HAS GONE INTO ANALYSIS OF MODIFICATIONS. IT APPEARS TO BE MISDIRECTED AND A STUDY OF THE DIFFICULTIES OF PROVIDING COMPLETE INFORMATION ON ASSIGNMENTS SHOULD BE COMMENCED IMMEDIATELY.

1.6. LOCAL OPTIMIZATION

IN THE ABSENCE OF GLOBAL ANALYSIS, WE CAN EXAMINE WHETHER ANY OF THE REQUIRED GLOBAL INFORMATION CAN BE OBTAINED BY PURELY LOCAL MEANS USING INFORMATION AVAILABLE TO THE SEMANTIC PASS.

ONE COMMON CASE INVOLVES TEMPORARIES:

IF WE HAVE:

```
A = TEMP;
```

IT WILL OFTEN BE THE CASE THAT TEMP IS DEAD AND KNOWN TO BE DEAD BY THE SEMANTIC PASS. THIS MEANS THAT THIS ASSIGNMENT CAN BE FLAGGED AS NOT REQUIRING A COPY.

ON THE OTHER HAND

```
TEMP = A;
```

USUALLY DOES REQUIRE A COPY (IN THE SENSE DESCRIBED).

1.7. BASING COMPLICATIONS

THE INTRODUCTION OF BASE SETS WHERE THERE ARE POINTERS TO AN OBJECT FROM OUTSIDE INTRODUCES ADDITIONAL COMPLICATIONS.

A BASE SET CANNOT BE COPIED ON MODIFICATION SINCE THIS WOULD INVALIDATE POINTERS TO THE OBJECT. THIS MEANS THAT THE

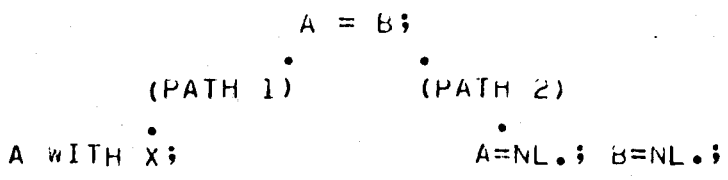
ASSIGNMENT:

```
A = B;  $ B IS A BASE SET
```

MUST CAUSE AN ACTUAL COPY IF A COPY IS REQUIRED. THIS COPY CANNOT BE REPLACED BY THE SHARED BIT MECHANISM.

THIS MAKES THE FLAGGING OF ASSIGNMENTS WHICH DO NOT REQUIRE A COPY MORE IMPORTANT IN THIS CASE. FAILURE TO FLAG AN ASSIGNMENT WILL ALWAYS CAUSE AN ACTUAL UNNEEDED COPY (IN THE NORMAL CASE IT CAUSES AN UNNECESSARY SETTING OF THE SHARED BIT WHICH MAY OR MAY NOT CAUSE AN EXTRA COPY).

IF THE OPTIMIZER IS EXAMINING MODIFICATIONS, THEN IT CAN TAKE ADVANTAGE OF THIS KNOWLEDGE AS SHOWN BY THE FOLLOWING:



IF B IS NOT A BASE, THEN WE HAVE THE FOLLOWING:

A=B;	FLAGGED AS MAYBE REQUIRING A COPY, SINCE COPY WILL BE REQUIRED ON PATH 1 BUT NOT ON PATH 2. THE RESULT WILL BE TO SET THE SHARED BIT.
A WITH X;	FLAGGED AS REQURING A COPY SINCE THE ONLY PATH IS FROM A=B.

IF B IS A BASE, THEN WE HAVE:

A=B;	FLAGGED AS REQUIRING A COPY, SINCE A COPY MUST BE PERFORMED IF THERE IS ANY PATH REQURING A COPY.
A WITH X;	FLAGGED AS NOT REQUIRING A COPY SINCE THE PATH FROM A=B DID A COPY.

THIS REFINEMENT IS NOT REQUIRED, ALTHOUGH IT IS CLEAR THAT IN ITS ABSENCE, IT IS WISER TO TREAT THE (COPY REQUIRED) FLAGGING OF A MODIFICATION AS (COPY MAY BE REQUIRED) SINCE THE SHARED BIT MAY BE OFF AND A COPY COULD BE AVOIDED.

TABLE OF CONTENTS

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 *erthis* 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.