SETL NEWSLETTER 171C
IMPLEMENTATION OF BASE ASSIGNMENTS

ART GRAND
APR 22, 1977

THIS NEWSLETTER DISCUSSES THE IMPLEMENTATION OF BASE ASSIGNMENTS. WE BEGIN BY EXAMINING A SIMPLE PROGRAM:

```
        BASE B1, B2;;

        REPR  MB1: MAP(↑B1) ↑B1;
              SB1: SET(↑B1);
              EB1: ↑B1;

              MB2: MAP(↑B2) ↑B2;
              SB2: SET(↑B2);
              EB2: ↑B2;
        END REPR;


(1)     <MB1, SB1, EB1> = <MB2, SB2, EB2>;
```

AT FIRST GLANCE THE ASSIGNMENT IN (1) SEEMS VERY EXPENSIVE, SINCE IT INVOLVES A SERIES OF CONVERSIONS FROM B2 BASING TO B1 BASING. NOTICE THAT IMMEDIATELY PRIOR TO (1) ALL OBJECTS DECLARED TO BE BASED ON B1 ARE DEAD. THUS THE COMPILER COULD INSERT AN ASSIGNMENT B1 = B2 JUST PRIOR TO (1) WITHOUT CHANGING THE SEMANTICS OF THE PROGRAM. THE RESULT WOULD BE TO ELIMINATE THE NEED FOR CONVERSIONS IN (1). SUCH COMPILER GENERATED ASSIGNMENTS ARE CALLED BASE ASSIGNMENTS.

WE GO ON TO A SOMEWHAT MORE COMPLEX CASE WHICH CONTAINS THE SAME REPRS, BUT CONTAINS THE ASSIGNMENTS

```
(2)         SB1 = OM;
                .
                .
                .
(3)         <MB1, EB1> = <MB2, EB2>;
```

CLEARLY THE SAME CONDITIONS HOLD ON B1, AND THE BASE ASSIGNMENT B1 = B2 MAY BE INSERTED PRIOR TO (3).

SO FAR BASE ASSIGNMENTS SEEM LIKE STANDARD ONE WORD ASSIGNMENTS, AND ARE ONLY UNUSUAL IN THAT THEY ARE GENERATED AT THE COMPILER. IN FACT THEY ARE SOMEWHAT MORE COMPLEX. THIS CAN BE SEEN BY TAKING A CLOSER THE VALUE OF SB1 AFTER EXECUTING (2) AND (3).

EVERY SETL VALUE HAS ASSOCIATED WITH IT A FLAG KNOW AS #IS→UNDEF# WHICH INDICATES THAT THE VALUE IS UNDEFINED. THUS THERE IS NO SINGLE REPRESENTATION OF OMEGA; ANY VALUE WITH ITS IS→UNDEF BIT SET IS TAKEN TO BE OMEGA. THIS ALLOWS THE ASSIGNMENT IN (2) TO BE IMPLEMENTED MERELY BY SETTING A FLAG IN THE VALUE OF Sb1.

THE TREATMENT OF OMEGA OUTLINED ABOVE ALLOWS US TO ENFORCE AN INVARIANT CONDITION ON THE SETL SYSTEM:

(4) ALL VARIABLES MUST HAVE A VALUE OF THE PROPER REPR AT ALL TIMES. THIS CONDITION HOLDS EVEN IF THE VARIABLE IS DEAD OR HAS BEEN ASSIGNED A VALUE OF OMEGA. (UNDECLARED VARIABLES ARE CONSIDERED TO BE DECLARED TYPE GENERAL, AND MAY THUS HAVE ANY VALUE)

A VARIABLE IS SAID TO HAVE A PROPER REPR IF THREE CONDITIONS HOLD:

1. ITS FORM FIELD MUST BE CORRECT.

2. ITS BASE ARRAY MUST CONTAIN POINTERS TO THE CORRECT BASES. IN PARTICULAR, SINCE SB1 IS DECLARED TO BE BASED ON B1, ITS BASE ARRAY MUST CONTAIN A POINTER TO THE CURRENT VALUE OF B1.

3. IF THE VALUE CONTAINS ANY OTHER POINTERS TO BASES, THEY MUST ALSO BE CORRECT. A LIST OF PLACES WHERE SUCH POINTERS CAN APPEAR IS GIVEN BELOW.

IT IS NOW OBVIOUS THAT AFTER PERFORMING THE ASSIGNMENT B1 = B2, THE VALUE OF SB1 NO LONGER HAS THE PROPER REPR. THUS AFTER EXECUTING THE ASSIGNMENT B1 = B2, WE MUST GO THROUGH THE VALUE OF SB1, REPLACING ALL POINTERS TO THE OLD VALUE OF B1 WITH POINTERS TO THE NEW VALUE OF B1. THIS PROCESS IS CALLED REBASING.

SUPPOSE B3 IS DECLARED TO BE A BASE(∂ B1). THEN THE BASE ASSIGNMENT B1 = B2 IS ONLY LEGAL IF B3 IS DEAD, I.E. IF IT IS FOLLOWED BY A BASE ASSIGNMENT TO B3. WE REQUIRE THAT B3 BE CONSISTENT WITH THE INVARIANT CONDITION DESCRIBED ABOVE. THIS MEANS THAT B3 MUST BE REBASED AFTER AN ASSIGNMENT TO B1. NOTE THAT THIS IMPLIES A CERTAIN ORDERING TO BASE ASSIGNMENTS. FOR EXAMPLE THE MULTPILE BASE ASSIGNMENT

<B1, B3> = <B2, NULL BASE>

WE MUST PERFORM THE ASSIGNMENT B1 = B2, THEN REBASE B3, AND FINALLY USE B3 AS A SAMPLE VALUE TO BUILD A NEW NULL BASE. AS WILL BE SEEN, THIS THIRD STEP CAN BE COMBINED WITH THE REBASING OPERATION.

IN SUMMARY, A BASE ASSIGNMENT B1 = B2 INVOLVES TWO STEPS. FIRST, MAKE THE ONE WORD ASSIGNMENT B1 = B2. THEN REBASE ALL VARIABLES V WHICH ARE BASED ON B1 AND ARE NOT IMMEDIATELY ASSIGNED VALUES BASED ON B2. NOTE THAT IN ORDER FOR THE BASE ASSIGNMENT TO BE LEGAL, THE V-S MUST ALL BE DEAD OR MUST BE KNOWN TO HAVE THE VALUE OMEGA. THUS THE REBASING ROUTINE CAN ASSIGN THEM ANY OMEGA VALUE OF THE PROPER REPR.

DETAILS OF REBASING

REBASING CAN SOMETIMES BE MINIMIZED, BUT IS USUALLY PERFORMED OFF LINE. THE REBASE ROUTINE TAKES TWO ARGUMENTS, THE CURRENT VALUE OF THE VARIABLE(OR BASE) ≠V≠ TO BE REBASED, AND ITS NEW BASE ARRAY. IT RETURNS AN OMEGA VALUE OF THE PROPER REPR. MORE SPECIFICALLY:

IF V HAS TYPE ELEMENT OF B, ITS NEW VALUE WILL BE A POINTER TO THE TEMPLATE BLOCK OF B WITH ITS ISUNDEF BIT SET.

IF V IS A TUPLE, ITS NEW VALUE WILL BE A NULL TUPLE WITH ITS ISUNDEF BIT SET.

IF V IS A SET, MAP, OR BASE, ITS NEW VALUE WILL BE A NULL SET OF THE PROPER TYPE WITH ISUNDEF SET.

ALL NULL VALUES CONTAIN A SAMPLE ELEMENT WHICH IS USED BY THE SYSTEM. FOR EXAMPLE, A NULL TUPLE(↓ INT2) WILL CONTAIN A ZERO-TH COMPONENT WHICH IS ACCESSIBLE ONLY TO THE SYSTEM AND WHOSE REPR IS SET(INT). THIS IMPLIES THAT REBASING OF A COMPOSITE OBJECT MUST BE A RECURSIVE PROCESS.

POINTERS TO BASES CAN ONLY APPEAR IN CERTAIN CONTEXTS. THE TYPE OF A SET OR TUPLE DETERMINES WHERE IT CAN CONTAIN POINTERS TO BASES. WE GIVE A LIST OF THEM HERE.

ALL COMPOSITE OBJECTS CONTAIN POINTERS TO BASES IN THEIR BASE ARRAYS.

IF A NULL TUPLE HAS COMPONENTS WHICH ARE BASED, THEN ITS SAMPLE COMPONENT WILL HAVE POINTERS TO BASES.

IF A NULL SET HAS ELEMENTS WHICH ARE BASED, THEN ITS SAMPLE ELEMENT HAS POINTERS TO BASES.

IF A NULL SET HAS TYPE SPARSE SET(↑B) THEN ITS SAMPLE ELEMENT MUST HAVE A POINTER TO B. IF A SET HAS TYPE SPARSE MAP(↑A) ↑B THEN ITS SAMPLE DOMAIN ELEMENT HAS A POINTER TO A AND ITS SAMPLE IMAGE HAS A POINTER TO B.

FINALLY, IF A SET IS A BASED SET OR MAP THEN ITS HASHTB FIELD POINTS TO THE HASH TABLE OF THE BASE.

OPTIMIZING BASE ASSIGNMENTS

BASE ASSIGNMENTS CAN BE OPTIMIZED BY RELAXING THE CONDITION STATED IN (4). THIS CONDITION HAS TWO PURPOSES.

CERTAIN NUBBINS SUCH AS F(X) ON LOCAL MAPS BECOME VERY EXPENSIVE IF THEY MUST TEST THEIR ARGUMENTS FOR OMEGA. RATHER THAN COMMITING THE SYSTEM TO CATCHING ALL ILLEGAL USES OF OMEGA, WE MERELY SAY THAT SUCH USES OF OMEGA WILL ALWAYS YIELD ORDERLY RESULTS. BY THIS WE MEAN THAT THEY WILL NEVER CAUSE A MACHINE ABORT WITH AN OBSCURE CORE DUMP; RATHER THEY WILL YIELD LEGAL SETL VALUES. THIS REQUIRES THAT F LOOK LIKE A LOCAL MAP EVEN IF IT IS SET TO OMEGA, IN OTHER WORDS ITS FORM MUST BE CORRECT. THIS CAN BE DONE WITHOUT EVER REBASING F.

CONVERSION FROM ONE REPR TO ANOTHER ARE ALWAYS DRIVEN BY A SAMPLE VALUE OF THE DESIRED REPR. FOR EXAMPLE, IF A AND B HAVE DIFFERENT MODES THEN THE ASSIGNMENT A = B REQUIRES A CONVERSION. THIS CONVERSION WILL BE DRIVEN BY SOME SAMPLE VALUE OF THE SAME MODE AS A; THE OBVIOUS THING IS TO USE THE OLD VALUE OF A AS A SAMPLE. THIS REQUIRES THE OLD VALUE TO HAVE THE PROPER REPR EVEN THOUGH IT IS DEAD.

IT IS OBVIOUS FROM THE ABOVE TWO PARAGRAPHS THAT THE ONLY VARIABLES WHICH NEED TO BE REBASED ARE THOSE WHICH ARE USED TO DRIVE CONVERSIONS. THUS THE NUMBER OF VARIABLES WHICH MUST BE REBASED AS PART OF A BASE ASSIGNMENT CAN BE REDUCED BY A COMBINATION OF LOCAL AND GLOBAL ANALYSIS. THIS IS ALREADY DONE BY THE SEMANTIC PASS ON A LOCAL BASIS.

FINALLY, SUPPOSE WE MODIFY ARE EXAMPLE IN (2) AND (3) BY ADDING A SECOND SET(↛ B1) WHICH IS ALSO DEAD AT LINE (3). IT IS NEVER NECESSARY TO REBASE BOTH SETS(↛B1). AT WORST WE MUST REBASE ONE AND THEN ASSIGN IT TO THE OTHER.

SCOPES OF BASES

BASES HAVE NAME SCOPES IN A PURELY SYNTACTIC SENSE, JUST LIKE MODE NAMES AND MACROS. A BASE DECLARED AT THE TOP OF A MODULE IS GLOBAL TO THE MODULE; A BASE DECLARED IN A PROCEDURE IS LOCAL. BASES ARE PUBLIC OR EXTERNAL IF THE VARIABLES BASED ON THEM ARE.

BASE ASSIGNMENT AND COMPILE TIME INITIALIZATION

ALL BASES ARE GIVEN INITIAL NULL VALUES BY THE COMPILER. THIS CONSTITUTES AN INITIAL BASE ASSIGNMENT. SINCE ALL VARIABLES ARE DEAD AT THIS POINT, THEY MUST ALL BE REBASED. SINCE THE COMPILER USES THE RUN TIME ENVIRONMENT, THIS IS DONE SIMPLY BY CALLING THE LIBRARY≠S REBASE ROUTINE.

BASE ASSIGNMENTS AND PARAMETER PASSAGE

LET US EXAMINE THE FOLLOWING CODE FRAGMENT IN WHICH BASED OBJECTS ARE PASSED AS PARAMETERS:

```
DEFINE P1;
    BASE B1;

    REPR SB1: SET(↗B1);
         EB1: ↗B1;;

    P2(SB1, EB1);
END P1;

DEFINE P2(SB2, EB2);
    BASE B2;

    REPR SB2: SET(↗B2);
         EB2: ↗ B2;;

 END P2;
```

HERE SB1 AND EB1 ARE THE ≠ACTUAL ARGUMENTS≠, AND SB2 AND EB2 ARE THE ≠FORMAL PARAMETERS≠. THE BINDING OF ARGUMENTS TO PARAMETERS IS EQUIVLENT TO THE ASSIGNMENT

```
<SB2, EB2> = <SB1, EB1>;
```

WE WOULD LIKE TO DETERMINE WHEN THE CONVERSIONS IMPLIED BY THIS ASSIGNMENT CAN BE REPLACED BY A BASE ASSIGNMENT B2 = B1. SUCH A BASE ASSIGNMENT IS POSSIBLE WHEN THERE ARE NO DECLARED VARIABLES LIVE ON B2 PRIOR TO THE BINDING OF ARGUMENTS. A SUFFICIENT, ALTHOUGH NOT NECESSARY CONDITION FOR THIS IS THAT THE ALL LOCAL VARIABLES BASED ON B2 BE DECLARED STACKED. IN THIS CASE B1 IS KNOWN AS AN ≠ACTUAL BASE≠ AND B2 IS KNOWN AS A ≠FORMAL BASE≠. ACTUAL BASES ARE PASSED AND ASSIGNED THROUGH THE NORMAL ARGUMENT PASSAGE MECHANISM.

THE BINDING OF ARGUMENTS TO PARAMETERS INVOLVES A SMALL AMOUNT OF INTERPROCEDURAL ANALYSIS TO DETERMINE WHETHER THE TYPES OF ARGUMENTS AND PARAMETERS MATCH, AND WHAT CONVERSIONS MUST BE PERFORMED. THIS ANALYSIS IS DONE BY THE ≠BINDER≠ PHASE OF THE COMPILER, WHICH IS ALSO RESPONSIBLE FOR MERGING SEPERATE MODULES, RESOLVING EXTERNAL VARIABLES, ETC. IN THIS SECTION WE OUTLINE THE TREATMENT OF ARGUMENTS AND PARAMETERS.

THE SEMANTIC PASS WILL DETERMINE WHICH BASES ARE FORMAL BASES. IT WILL MARK THEM BY ASSIGNING THEM PARAMETER NUMBERS, AND GENERATING ≠PARAMIN ASSIGNMENTS≠ FOR THEM AT THE START OF THE ROUTINE, JUST AS IT DOES FOR VARIABLES WHICH ARE FORMAL PARAMETERS.

THE CODE FOR THE CALL

```
P(X=)
```

WILL BE GENERATED AS

```
(4)     TEMP = X;
(5)     P(TEMP=);
(6)     X = TEMP;
```

X WILL BE TYPED BY THE SEMANTIC PASS USING BOTH REPRS AND LOCAL INFORMATION. THE BINDER WILL PROCESS THE CALL AND ASSIGN ≠TEMP≠ THE TYPE OF P≠S FORMAL PARAMETER. IF THIS MATCHS THE TYPE OF X, IT WILL FLAG THE ASSIGNMENTS (4) AND (6) AS NO-OPS. OTHERWISE THE CODE GENERATOR WILL RECOGNIZE THEM AS CONVERSIONS. LINE (4) IS REFERED TO AS ≠CONVERT IN≠ ASSIGNMENT, AND LINE (6) IS REFERED TO AS A ≠CONVERT OUT≠ ASSIGNMENT.

THE BINDER PROCESSES EACH CALL IN THREE STEPS:

STEP 1. SEE WHICH BASES ARE GOOD CANDIDATES TO BE USED AS ACTUAL BASES. THIS STEP WORKS AS FOLLOWS: ITERATE OVER THE LIST OF FORMAL PARAMETERS AND THE CORRESPONDING ACTUAL ARGUMENTS. IF THE I-TH PARAMETER AND THE I-TH ARGUMENT HAVE DIFFERENT FORMS, A CONVERSION WILL OBVIOUSLY BE REQUIRED. OTHERWISE COMPARE THEIR BASE ARRAYS. IF THE J-TH BASE OF THE PARAMETER IS A FORMAL BASE, THEN IT MAY BE THAT THE J-TH BASE OF THE ARGUMENT CAN BE USED AS THE J-TH ACTUAL BASE.

STEP 2. SEE IF THERE ARE ANY ACTUAL BASES MISSING. IF SO, GENERATE TEMPORARAY BASES FOR THEM.

STEP 3. SUPPLY THE REPRS FOR THE CONVERT-IN TEMPORARIES SUCH AS ≠TEMP≠ ABOVE. THE REPR OF THE I-TH CONVERT-IN TEMPORARY IS COMPUTED AS FOLLOWS: START WITH THE FORM AND BASE ARRAY OF THE I-TH FORMAL PARAMETER. IF THE PARAMETERS J-TH BASE IS A FORMAL BASE, REPLACE IT WITH THE CORRESPONDING ACTUAL BASE. ONCE THE REPR HAS BEEN BUILT, SEE IF IT MATCHES THE REPR OF THE I-TH ARGUMENT. IF SO, FLAG THE CORRESPONDING CONVERT-IN AND CONVERT-OUT ASSIGNMENTS AS NO-OPS.

LINKAGE TO PROCEDURE VARIABLES

CALLS TO PROCEDURE VARIABLES ARE IMPLEMENTED USING A VARIATION OF THE ABOVE SCHEME. SUPPOSE WE HAVE A PROCEDURE P WHICH BEGINS:

```
DEFINE P(S, X);

   BASE B;;

   REPR S: SET(↗ B);
        X: ↗ B;
   END;
        .
        .
        .
END P;
```

THE SEMANTIC PASS WILL COMPILE AN ADDITIONAL PROCEURE

```
DEFINE P1(S1, X1);

   REPR S1, X1: GENERAL;;

   P(S1, X1);
   RETURN;
END P1;
```

IT WILL REPLACE ALL PROCEDURE ASSIGNMENTS X = P WITH X = P1. THUS PROCEDURE VARIABLES WILL ALWAYS HAVE ARGUMENTS OF TYPE GENERAL, AND WILL NEVER REQUIRE ARGUMENT CONVERSIONS. INSTEAD THE BINDER WILL INSERT THE APPROPRIATE CONVERSIONS WHEN P1 CALLS P.

STACKED BASES

SUPPOSE B IS A BASE WHICH IS LOCAL TO SOME PROCEDURE P. FURTHERMORE SUPPOSE THAT ALL VARIABLES BASED ON B ARE STACKED LOCAL VARIABLES. THIS MEANS THAT ALL VARIABLES BASED ON B ARE DEAD ON ENTRY TO P. AS A RESULT IT IS POSSIBLE TO EXECUTE THE BASE ASSIGNMENT

```
B = NULL BASE;
```

EVERY TIME P IS ENTERED. THIS BASE ASSIGNMENT IS DESIRABLE FOR TWO REASONS.

FIRST, A STACKED VARIABLE CANNOT BE GIVEN A LOCAL REPRESENTATION UNLESS ITS BASE IS ALSO STACKED, SINCE THIS WOULD MEAN THAT DIFFERENT GENERATIONS OF THE VARIABLE COULD REFER TO THE SAME MEMBERSHIP BIT OR MAP VALUE FIELD OF THE SAME BASE.

SECOND, IT IS LIKELY THAT EACH INVOCATION OF P WILL ADD SOME ELEMENTS TO THE BASE WHICH WILL BECOME DEAD AT THE END OF THE PROCEDURE. IT IS MUCH FASTER TO RECOVER THE SPACE USED BY THESE ELEMENTS IF WE ALLOW AN ENTIRE STACKED BASE TO BECOME DEAD, RATHER THAN DOING BASE COMPACTION ON A STATIC BASE.

IT IS OF COURSE POSSIBLE TO HAVE STACKED AND STATIC VARIABLES ON THE SAME BASE. IN THIS CASE THE BASE CANOT BE STACKED, AND THE TREATMENT OF THE STACKED VARIABLES WILL BE SOMEWHAT INEFFICIENT.

THE DECISION WHETHER TO STACK A GIVEN BASE IS PURELY LOCAL, AND CAN BE MADE BY THE SEMANTIC PASS.

NOTE THAT BY DEFAULT VARIABLES ARE LOCAL STACKED. VARIABLES DECLARED GLOBALLY OR USED IN A MAIN PROGRAM DEFAULT TO GLOBAL STATIC.

LOCAL OBJECTS AND BASE ASSIGNMENTS

THE BASE ASSIGNMENT B1 = B2 IS ONLY LEGAL IF

1. THE ELEMENT BLOCKS OF B2 HAVE ROOM FOR ALL THE LOCAL MEMBERSHIP BITS, LOCAL MAP IMAGES, ETC. OF VARIABLES BASED ON B1.

2. THE LOCAL OBJECTS BASED ON B1 AND B2 USE DISJOINT PARTS OF EACH ELEMENT BLOCK.

AN EASY, SUFFICIENT CONDITION FOR THIS IS THAT THERE BE NO LOCAL OBJECTS ON B1. WE CURRENTLY PLACE THIS RESTRICTION ON FORMAL BASES.

A REMARK ON BASES AND BACKTRACKING

ASSUME THAT V IS A BACKTRACK VARIABLE BASED ON B. AN OK OPERATION WILL PRESUMABLY NOT SAVE V WHEN IT IS KNOWN TO BE DEAD. THUS THE CONDITIONS UNDER WHICH AN OK OPERATION SAVES V ARE PRECISELY THE CONDITIONS UNDER WHICH BASE ASSIGNMENTS TO B ARE PROHIBITED. AS A RESULT THERE SHOULD BE NO NEED TO SAVE A SPECIFIER FOR B DURING AN OK OPERATION.

THE CURRENT BACKTRACKING SCHEME USES SHARE BITS TO PREVENT SAVED VALUES FROM BEING USED DESTRUCTIVELY. SINCE LOCAL OBJECTS DO NOT USE THE SHARE BIT MECHANISM, BACKTRACK VARIABLES CANNOT BE GIVEN LOCAL REPRESENTATIONS.