SETL NEWSLETTER # 206

ON NAME SPLITTING IN SETL OPTIMIZATION

A. Grand
M. Sharir
February 5, 1978

NAME SPLITTING IS ONE OF THE FINAL PHASES OF THE SETL OPTIMIZER. IT IS PERFORMED AFTER THE TYPE-FINDING AND AUTOMATIC DATA STRUCTURE SELECTION PHASES HAVE COMPUTED A MAP ≠OI¬REPR≠, DEFINED ON VARIABLE OCCURENCES, FOR EACH OCCURENCE VO, OI¬REPR(VO) IS A REPR FOR VO THAT HAS BEEN SELECTED BY THESE PHASES, CONJOINED WITH USER-SUPPLIED REPR INFORMATION.

HOWEVER, THE MAP OI¬REPR IS NOT OF DIRECT USE TO THE CODE GENERATOR, WHICH DEALS WITH SYMBOL TABLE ENTRIES RATHER THAN WITH OCCURENCES.

OF COURSE, IF ALL OCCURENCES OF THE SAME VARIABLE HAVE THE SAME REPR, THEN WE CAN TRANSMIT THIS REPR TO THE CODE GENERATOR. UNFORTUNATELY, THERE ARE MANY COMMON CASES (MAINLY, BUT NOT EXCLUSIVELY, IN NON-REPRED PROGRAMS) WHERE THIS IS NOT THE CASE. FOR EXAMPLE, CONSIDER

EXAMPLE 1.

```
        READ X;
        Y := X + 1;
```

OR EVEN:

EXAMPLE 2.

```
(1)     READ X;
(2)     IF COND THEN
(3)         (∨)
(4)             Y := X + 1;
(5)         END ∨;
(6)     ELSE
(7)         (∨)
(8)             Z := X + ≠1≠;
(9)         END ∨;
(10)    END IF;
```

EVEN IF WE FULLY REPR EXAMPLE 2., THE ONLY REPR THAT X CAN HAVE IS TYPE GENERAL, SO THAT THE ADDITIONS AT LINES (4) AND (8) WILL BE SLOWED DOWN DUE TO TYPE CHECKS AND CONVERSIONS. MOREOVER, IT WILL BE IMPOSSIBLE TO EMIT IN-LINE CODE FOR THESE ADDITIONS. IT IS THEREFORE OF INTEREST TO NOTE THAT BY ANALYZING OCCURENCES, RATHER THAN VARIABLES, TYPE-FINDING WILL REVEAL THAT TYPE(X1 := X AT LINE 1) := GENERAL, TYPE(X4) := INTEGER AND

TYPE(X8) := CHARACTERS. WE CAN THEN MAKE USE OF THIS INFORMATION BY SPLITTING THE VARIABLE X INTO THREE DIFFERENT SYMBOL TABLE ENTRIES, XA, XB, XC, HAVING THE REPRS GENERAL, INTEGER AND CHARACTERS RESPECTIVELY. IN THIS WAY WE SEPARATE THE NECESSARY TYPE CHECKS AND CONVERSIONS FROM THE ACTUAL ADD INSTRUCTIONS (4) AND (8), AND THUS OPEN UP THE POSSIBILITY OF GENERATING EFFICIENT CODE FOR THEM. WHEN THIS IS DONE, CONVERSIONS MUST BE MADE EXLICIT IN THE CODE. WE CAN TRY TO INSERT THESE CONVERSIONS AT OPTIMAL PLACES BY MOVING THEM OUT OF LOOPS, IF POSSIBLE.

LET US VISUALIZE THESE CONVERSIONS AS ASSIGNMENTS OF ONE SPLIT VARIABLE TO ANOTHER. THE SECOND EXAMPLE WOULD THEREBY BE TRANSFORMED INTO THE FOLLOWING CODE:

EXAMPLE 2A.

```
        READ XA;
        IF COND THEN
            XB := XA;
            (∨)
                Y := XB + 1;
            END ∨;
        ELSE
            XC := XA;
            (∨)
                Z := XC + ≠1≠;
            END ∨;
        END IF;
```

WE CALL THE TRANSFORMATION FROM (2) TO (2A) ≠NAME-SPLITTING≠.

ANOTHER AND MORE IMPORTANT CASE IN WHICH NAME SPLITTING IS REQUIRED IS THE INSERTION OF ≠LOCATE≠ INSTRUCTIONS WHICH PUT BASED ELEMENTS INTO THEIR BASES. CONSIDER THE FOLLOWING EXAMPLE:

EXAMPLE 3.

```
(1)     (∨)
(2)         X := X + 1;
(3)     END ∨;
(4)     S WITH X;
```

SUPPOSE THAT S4 HAS THE REPR SET(→B) (CHOSEN AUTOMATICALLY OR MANUALLY). IN THIS CASE IT IS DISADVANTAGEOUS TO REPR BOTH X2 AND X4 AS →B, FOR ONLY THE LAST CREATED VALUE OF X2 HAS ACTUALLY TO BE INSERTED INTO THE BASE B. WE EXPECT THE AUTOMATIC DATA STRUCTURE SELECTION PHASE TO COME UP WITH OI¬REPR(X2) := INTEGER, OI¬REPR(X4) := →B . THE HEURISTIC NAME SPLITTING SHOULD THEREFORE AIM TO TRANSFORM THIS CODE INTO:

EXAMPLE 3A.

```
(∨)
    XA := XA + 1;
END ∨;
XB := XA;
S WITH XB;
```

WHERE XA, XB ARE SPLIT SYMBOL TABLE ENTRIES FOR X, HAVING THE REPRS INTEGER AND →B RESPECTIVELY, AND THE ASSIGNMENT XB := XA ACTUALLY SIGNIFIES A ≠LOCATE≠ OF THE VALUE OF XA IN B. SUCH A ≠LOCATE≠ COMPUTES A BASE POINTER FOR XA, INSERTING IT INTO B IF NECESSARY, AND ASSIGNS THIS POINTER TO XB.

IN THE FOREGOING EXAMPLE, NO MOTION OF LOCATES IS NEEDED. HOWEVER, IN THE FOLLOWING EXAMPLE

EXAMPLE 4.

```
X := X + 1;
(∨)
    S WITH X;
END ∨;
```

CODE MOTION IS PROBABLY ADVANTAGEOUS, SINCE IT WILL TRANSFORM THIS CODE INTO

EXAMPLE 4A.

```
XA := XA + 1;
XB := XA;
(∨)
    S WITH XB;
END ∨;
```

WITH SIMILAR XA AND XB.

HAVING CONVINCED OURSELVES THAT A NAME-SPLITTING MECHANISM IS NECESSARY, LET US NOW DESCRIBE A NAME SPLITTING ALGORITHM AND THE CONVERSION-MOTION ALGORITHM IT USES IN DETAIL.

DEFINITION: LET V BE A PROGRAM VARIABLE, AND LET R BE A REPR OF SOME OF ITS OCCURENCES, WE DEFINE A SPLIT VARIABLE OF V WITH THE REPR R, AS A PAIR [V, R]. LET ≠SPLIT¬NAME≠ BE THE MAP SENDING EACH OCCURENCE VO INTO THE PAIR [OI¬NAME(VO), OI¬REPR(VO)], AND LET ≠EQREPR≠ BE THE EQUIVALENCE RELATION INDUCED BY THIS MAP AS A QUOTIENT MAP.

FOR EVERY SPLIT VARIABLE OF V WE INTRODUCE A NEW SYMBOL TABLE ENTRY VA, WITH THE UNDERSTANDING THAT IF V IS NOT REALLY SPLIT, THEN VA WILL BE THE ORIGINAL ENTRY OF V. AFTER SPLITTING, EACH VARIABLE OCCURENCE WILL THEN BE REGARDED AS AN OCCURENCE OF

AN APPROPRIATE SPLIT VARIABLE, AND THE CODE WILL BE MODIFIED TO SHOW THESE SPLIT VARIABLES RATHER THAN THE VARIABLES ORIGINALLY OCCURING.

ONE IMPORTANT OBSERVATION IS THAT TWO SPLIT VARIABLES OF THE SAME ORIGINAL VARIABLE ARE NEVER LIVE SIMULTANEOUSLY, AND SO THEY CAN SHARE STORAGE. THIS FACT WILL BE USED BY THE CODE GENERATOR, AND ALSO BY THE NAME-SPLITTING ALGORITHM ITSELF.

THE NAME SPLITTING ALGORITHM CONSISTS OF THE FOLLOWING STEPS:

STEP 1) PERFORMS ACTUAL NAME SPLITTING AND COLLECTS ALL LINKS BETWEEN OCCURENCES OF A SINGLE VARIABLE WHICH HAVE BEEN SPLIT.

STEP 2) INSERTS CONVERSIONS AND CHECKS INTO THE CODE.

STEP 1) IS PERFORMED IN A STRAIGHT-FORWARD WAY, BY ITERATING OVER BFROM. WHENEVER WE ENCOUNTER A LINK BETWEEN TWO OCCURENCES WITH DIFFERENT OI-REPR VALUES, WE AUGMENT A WORKPILE OF SUCH LINKS, AND ADD NEW ENTRIES TO THE SYMBOL TABLE, IF NECESSARY. THE VARIABLE NAMES APPEARING IN OCCURENCES ARE REPLACED BY THE NAMES OF THE CORRESPONDING SPLIT VARIABLES.

STEP 2) IS MORE COMPLICATED, AND, LIKE ANY OTHER CODE MOTION ALGORITHM, RAISES PROBLEMS OF SAFETY AND PROFITABILITY, AS WELL AS A FEW ADDITIONAL MORE SPECIFIC PROBLEMS. THE APPROACH DESCRIBED BELOW AIMS TO ENSURE A MODERATE LEVEL OF PROFITABILITY, BUT MAY NOT PRODUCE OPTIMAL CODE IN SEVERAL EXTREME CASES. HOWEVER, IT IS RATHER SIMPLE, AND WILL GENERALLY PRODUCE QUITE ACCEPTABLE CODE.

LET VO BE A VARIABLE OCCURENCE, FOR WHICH THERE EXISTS VO1 → BFROM≤VO≥ SUCH THAT NOT (VO .EQREPR VO1). THE SAFEST PLACE TO INSERT A CONVERSION/TEST OF THE FORM OF VO IS JUST BEFORE THE INSTRUCTION CONTAINING VO. INDEED, THE TYPE FINDER AND THE AUTOMATIC DATA STRUCTURE SELECTION PHASES FUNCTION IN SUCH A WAY AS TO ENSURE THAT THE INSTRUCTION ORIGINALLY CONTAINING VO, BEFORE NAME-SPLITTING, IS EQUIVALENT TO THE CONVERSION/TEST FOLLOWED BY THE SAME INSTRUCTION BUT WITH THE SPLIT VARIABLE REPLACING THE ORIGINAL VARIABLE.

CONVERSION/TEST INSTRUCTIONS ARE REPRESENTED IN THE FINAL CODE WE ENVISAGE AS ASSIGNMENTS OF ONE SPLIT VARIABLE TO ANOTHER. SINCE ALL THE MEMBERS OF A GROUP OF VARIABLES SPLIT FROM A SINGLE ORIGINAL VARIABLE SHARE STORAGE, SUCH AN ASSIGNMENT IS SIMPLY A CONVERSION OF THE VALUE SPECIFIED AT THIS COMMON LOCATION (OR PERHAPS JUST A TEST THAT THIS VALUE HAS A DESIRED FORM), AND A REPLACEMENT OF THE OLD VALUE SPECIFIER WITH THE NEW ONE.

HOWEVER, VO MAY HAVE TWO OR MORE OCCURENCES, VO1, VO2 → BFROM≤VO≥, SUCH THAT VO1, VO2 ARE TO BE REPLACED BY DIFFERENT SPLIT

VARIABLES, SO THAT IF WE INSERT THE CONVERSION JUST BEFORE V0, ITS IVARIABLE MUST HAVE A FORM DOMINATING THOSE OF V01 AND V02 (IN THE TYPE-LATTICE SENSE). IN THIS CASE, WE GENERATE A NEW DUMMY SPLIT VARIABLE OF THE VARIABLE OF V0, HAVING THIS DOMINATING FORM, AND MAKE IT THE NAME OF THE IVARIABLE OF THE CONVERSION. THIS OPERATION IS EASILY SEEN TO WORK PROPERLY, SINCE ALL THESE SPLIT VARIABLES SHARE STORAGE.

IN SOME CASES WE ALSO HAVE ANOTHER ALTERNATIVE; NAMELY - TO PUSH THE CONVERSION UPWARD TOWARDS V01 AND V02, CONTINUING TO MOVE IT UPWARD UNTIL THE CONVERSION ITSELF CAN BE SPLIT INTO TWO CONVERSIONS HAVING AS IVARIABLES THE SPLIT VARIABLES OF V01 AND V02 RESPECTIVELY. THIS IS SHOWN IN THE FOLLOWING EXAMPLE:

EXAMPLE 5.

```
        F := ≤ [1,2] ≥;
        GO TO L1;
L2
        F := [1,2];
L1
        PRINT F(1);
```

HERE F HAS THREE SPLIT VARIABLES, FA (MAP), FB (TUPLE) AND FC(GENERAL). THE FIRST APPROACH SKETCHED ABOVE WILL TRANSFORM THIS INTO

EXAMPLE 5A.

```
        FA := ≤ [1,2] ≥;
        GO TO L1;
L2
        FB := [1,2];
L1
        FC := FD;
        PRINT FC(1);
```

WHERE FD IS A SPLIT VARIABLE OF F HAVING GENERAL TYPE (IN THIS PARTICULAR EXAMPLE, FD IS IDENTICAL WITH FC). THE SECOND APPROACH SKETCHED ABOVE WILL TRANSFORM THE CODE INTO

EXAMPLE 5B.

```
        FA := ≤ [1,2] ≥;
        FC := FA;
        GO TO L1;
L2
        FB := [1,2];
        FC := FB;
L1
        PRINT FC(1);
```

HOWEVER, THERE ARE CASES WHERE THE SECOND ALTERNATIVE WILL FAIL, AS IN THE FOLLOWING EXAMPLE:

EXAMPLE 6.

```
        F := ≤ [1,2] ≥;
        GO TO L1;
L2
        F := [1,2];
L1
        . . .
        . . .
        IF COND THEN
            F + ≤ [3,4] ≥;
        ELSE
            F + [3,4];
        END IF;
```

HERE F HAS ONLY TWO SPLIT VARIABLES FA(MAP) AND FB(TUPLE), BUT, AS CAN BE CHECKED, THERE IS NO PLACE IN THE CODE IN WHICH WE CAN INSERT AN EXPLICIT CONVERSION OF ONE SUCH SPLIT VARIABLE INTO ANOTHER, WITHOUT CAUSING A POSSIBLE ABORT, WHICH MIGHT NOT HAVE OCCURED IN THE ORIGINAL PROGRAM.

THUS, OUR ALGORITHM WILL HAVE TO MAKE USE OF THE FIRST ALTERNATIVE FOR AT LEAST CERTAIN CASES, AND WE PROPOSE TO DROP THE SECOND ALTERNATIVE ALTOGETHER, FOR THE FOLLOWING REASONS:

A) SITUATIONS SUCH AS THOSE SHOWN IN EXAMPLE 5. ARE RARE. IN MOST CASES, VO WILL BE LINKED ONLY TO ONE SPLIT VARIABLE, AND SO INSERTED CONVERSIONS WILL HAVE A SPECIFIC IVARIABLE ANYWAY.

B) EVEN WHEN SITUATIONS SUCH AS THAT SHOWN IN EXAMPLE 5. DO HAPPEN, THE GAIN FROM HAVING A SPECIFIC IVARIABLE IN INSERTED CONVERSIONS (NOTE THAT CONVERSIONS FROM TYPE GENERAL ARE SLOWER, SINCE THEY INVOLVE A BRANCH ON THE ACTUAL FORM OF THE VARIABLE) DOES NOT JUSTIFY THE SIGNIFICANT INCREASE IN THE COMPLEXITY OF THE ALGORITHM REQUIRED TO IMPLEMENT THE SECOND APPROACH.

HOWEVER, CERTAIN QUITE COMMON CASES DO CALL FOR SOMETHING LIKE OUR SECOND APPROACH. THESE ARE CASES IN WHICH ALL OCCURENCES OF A VARIABLE WITHIN A LOOP HAVE THE SAME SPLIT VARIABLE, BUT SOME OF THEM ARE ALSO LINKED TO OCCURENCES OUTSIDE THE LOOP, HAVING A DIFFERENT SPLIT VARIABLE. IF IN THESE CASES WE LEAVE CONVERSIONS INSIDE A LOOP, THEY MAY HAVE TO BE CONVERSIONS FROM TYPE GENERAL, AND IN ANY CASE THEY WILL IMPLY REDUNDANT TESTS THAT A VARIABLE IS IN THE CORRECT FORM, FROM THE SECOND ITERATION ONWARD. THUS, THE POSSIBILITY OF MAKING THE IVARIABLE OF AN INSERTED CONVERSION MORE SPECIFIC WHILE MOVING CONVERSIONS OUT OF LOOPS WILL BE TAKEN INTO CONSIDERATION IN OUR ALGORITHM.

AS WITH ANY KIND OF CODE MOTION, MOVING A CONVERSION OUT OF A LOOP IS IN GENERAL NOT SAFE, FOR THE MODIFIED PROGRAM MAY ABORT IN CERTAIN SITUATIONS IN WHICH THE ORIGINAL PROGRAM WOULD NOT (E.G. THE LOOP MAY BE BYPASSED).

EVEN IF WE FOLLOW (AND INDEED WE WILL) THE APPROACH OF THE STANDARD CODE MOTION PHASE OF THE OPTIMIZER (SEE NL. 197), I.E. - ASSUME THAT CODE MOTION WILL BE PERFORMED ONLY ON PROGRAMS THAT WILL RUN IN A SPECIAL EXECUTION MODE, IN WHICH OPERATIONS WITH ILLEGAL ARGUMENTS DO NOT CAUSE A PROGRAM ABORT, BUT PRODUCE AN ERROR VALUE, WE STILL FACE A SAFETY PROBLEM, CHARACTERISTIC OF ASSIGNMENTS; NAMELY - IF WE MOVE AN ASSIGNMENT OUT OF A LOOP AND IT IS ILLEGAL, THEN EVEN IF THE PROGRAM DOES NOT ABORT, ASSIGNING THE ERROR VALUE TO THE OVARIABLE WILL KILL THE PREVIOUS VALUE SPECIFIER OF THAT VARIABLE, WHICH WOULD BE RETAINED, HAD THE ASSIGNMENT BEEN LEFT IN THE LOOP, AND THE LOOP NEVER EXECUTED. FOR EXAMPLE:

EXAMPLE 7.

```
        READ V;
        LENV := 0;
        IF TYPE V = TUPLE THEN LENV := #V; END IF;

        (∀ I := 1 ... LENV)
            X := V(I);
        END ∀;

        IF LENV = 0 THEN X := V; END IF;
```

ASSUMING THAT THE INPUT V IS EITHER AN INTEGER OR A TUPLE OF INTEGERS, THE ABOVE CODE WILL NOT ABORT, AND AT ITS END, X WILL BE ASSIGNED AN INTEGER VALUE. SUPPOSE THAT THE TYPE INFORMATION SUGGESTS THAT WE SPLIT V INTO THREE SPLIT VARIABLES, VA(GENERAL), VB(TUPLE) AND VC(INTEGER). THE CODE CAN THEN BE SAFELY TRANSFORMED INTO THE FOLLOWING CODE:

```
        READ VA;
        LENV := 0;
        IF TYPE VA = TUPLE THEN LENV := #VA; END IF;

        (∀ I := 1 ... LENV)
            VB := VA;
            X := VB(I);
        END ∀;

        IF LENV = 0 THEN VC := VA; X := VC; END IF;
```

NOTE THAT THE ASSIGNMENT VB := VA; CAN NOT BE MOVED OUT OF THE LOOP, EVEN IN THE SPECIAL EXECUTION MODE DESCRIBED ABOVE, FOR IF WE DO MOVE IT OUT, AND V HAPPENS TO BE AN INTEGER, THEN THIS

CONVERSION WILL RESULT IN AN ERROR VALUE, STORED AT THE COMMON LOCATION OF ALL THE SPLIT VARIABLES OF V, SO THAT THE INTEGER VALUE OF V IS DESTROYED, AND X WILL BE ASSIGNED AN ERROR VALUE, INSTEAD OF THE INPUT INTEGER VALUE OF V.

NOTE, HOWEVER, THAT NOT EVERY CONVERSION CAN FAIL. A CONVERSION IS UNCONDITIONALLY SAFE, IF THE FORM OF ITS IVARIABLE IS MORE SPECIFIC THAN, OR EQUIVALENT TO, THE FORM OF ITS OVARIABLE. FOR EXAMPLE - CONVERSION TO GENERAL IS ALWAYS SAFE; CONVERSION FROM INTEGER TO AN ELEMENT OF B, WHERE B IS A BASE OF INTEGERS, IS ALWAYS SAFE, AS WELL AS THE INVERSE CONVERSION. THIS SUGGESTS THAT WE EXTEND THE TYPE ORDER TO A PSEUDO-ORDER RELATION (REFLEXIVE, TRANSITIVE, BUT NOT NECESSARILY ANTI-SYMMETRIC) ≠.LE≠ DEFINED ON FORMS IN THE FOLLOWING WAY: LET ≠TYPE¬OF≠ DENOTE THE FUNCTION THAT COMPUTES THE TYPE OF A GIVEN FORM, IN A RECURSIVE MANNER, REPLACING ≠ELEMENT-OF-BASE≠ DESCRIPTORS BY THE MODE OF THE CORRESPONDING BASES. THEN FORM1 .LE. FORM2 IFF TYPE¬OF(FORM2) DOMINATES TYPE¬OF(FORM1) IN THE STANDARD TYPE LATTICE. THUS, IT CAN BE EASILY SEEN THAT A CONVERSION VA := VB; IS ALWAYS SAFE IFF FORM(VB) .LE FORM(VA).

IN VIEW OF THE OBSERVATIONS MADE IN THE PRECEEDING PARAGRAPHS, WE SHALL USE THE FOLLOWING RATHER SIMPLE, THOUGH SOMEWHAT WEAK, CRITERION TO DETERMINE WHETHER A CONVERSION CAN BE MOVED OUT OF A LOOP:

LET V0 BE A VARIABLE OCCURENCE, LINKED BY BFROM TO OTHER SPLIT VARIABLES. IF THE FOLLOWING CONDITION IS SATISFIED,

```
(*)     ∨ V01 → BFROM≤V0≥ ↑ ( OI¬REPR(V01) .LE OI¬REPR(V0)
          OR (∨ V02 → FFROM≤V01≥ ↑ OI¬REPR(V02) .LE OI¬REPR(V0)))
```

THEN A CONVERSION TO SPLIT¬NAME(V0), INITIALLY PLACED JUST BEFORE THE INSTRUCTION CONTAINING V0, CAN BE MOVED OUT OF ITS LOOP(S). OTHERWISE, IT MUST REMAIN AT ITS INITIAL LOCATION.

THE HEURISTIC BASIS OF THIS APPROACH MAY BE STATED AS FOLLOWS: GIVEN THAT WE DO NOT HAVE ANY MORE DETAILED DATA-FLOW INFORMATION, WE MUST ASSUME THAT IF A LOOP CONTAINING V0 IS NOT EXECUTED, THEN THERE MAY MAY BE A PATH FROM SOME V01 → BFROM≤V0≥ TO ANOTHER V02 → FFROM≤V01≥, WHICH PASSES THROUGH THE TARGET BLOCK OF THIS LOOP (INTERVAL), (THE TARGET BLOCK OF AN INTERVAL IS A SPECIAL BASIC BLOCK, CREATED BY THE OPTIMIZER, WITH THE PROPERTY THAT IT IS THE ONLY BASIC BLOCK OUTSIDE THIS INTERVAL THAT IS A PREDECESSOR OF THE INTERVAL HEAD. WE CREATE THIS BLOCK SO THAT CODE MOVED OUT OF THIS INTERVAL CAN BE INSERTED INTO IT), AND IF WE M VE A CONVERSION TO THE FORM OF V0 OUT OF THAT INTERVAL, INTO ITS BASIC BLOCK, THIS CONVERSION CAN CUT THE ABOVE PATH FROM V01 TO V02. THUS, TO BE SURE THAT NO HARM WILL BE DONE, WE MUST BE SURE THAT IF THIS CONVERSION FAILS, THEN THE CONVERSION TO SPLIT¬NAME(V02), PLACED JUST BEFORE V02, WOULD HAVE FAILED ALSO. CONDITION (*) IS PRECISELY EQUIVALENT TO THAT ASSERTION.

ONCE HAVING GIVEN THE PRECEEDING SOLUTION TO THE SAFETY PROBLEM, WE USE A RATHER LIBERAL CRITERION FOR PROFITABILITY, AND ASSUME THAT IT IS ALWAYS PROFITABLE TO MOVE A CONVERSION FROM THE LOOP-PART OF AN INTERVAL TO ITS TARGET BLOCK.

NOTE THAT THE CRITERION FOR PROFITABILITY SET BY THE STANDARD CODE MOTION PHASE OF THE OPTIMIZER IS STRICTER, AND DEMANDS THAT A COMPUTATION MUST BE UNCONDITIONALLY EXECUTED INSIDE AN INTERVAL, PROVIDED THAT THE LOOP OF THAT INTERVAL IS EXECUTED AT LEAST ONCE. NOTE ALSO THAT OUR CODE MOTION CRITERIA REFLECT OUR RELUCTANCE TO RE-PERFORM A FULL SCALE DATA-FLOW ANALYSIS, WHICH WOULD ALLOW US TO SHARPEN THE CRITERIA OF MOTION, AND IMPROVE THE LOCATION-OPTIMIZATION OF CONVERSIONS. WE MIGHT BE WILLING TO PERFORM SUCH AN ANALYSIS IF FUTURE EXPERIMENTATION WITH SETL OPTIMIZATION REVEALS SIGNIFICANT INEFFICIENCIES IN THE MOTION OF CONVERSIONS, BUT PRESENTLY IT SEEMS LIKELY THAT OUR ALGORITHM WILL ACHIEVE GOOD RESULTS IN MOST CASES.

THERE IS YET ONE MORE PROBLEM THAT THE MOTION OF CONVERSIONS RAISES. IN THE GENERAL CASE, IT IS RATHER DIFFICULT TO DETERMINE A PROPER REPR FOR AN IVARIABLE OF AN INSERTED CONVERSION OPERATION. LET VO BE SOME VARIABLE OCCURENCE OF A VARIABLE V WHICH IS LINKED BY BFROM TO OTHER OCCURENCES OF V HAVING SPLIT VARIABLES WHICH ARE DIFFERENT FROM SPLIT-NAME(VO), SO THAT A CONVERSION TO THE FORM OF VO OUGHT TO BE INSERTED BEFORE VO IS USED. SUPPOSE THAT OUR ALGORITHM HAS DETERMINED TO MOVE THAT CONVERSION TO THE TARGET BLOCK B OF SOME INTERVAL INT CONTAINING VO, IN COMPLIANCE WITH ALL THE ABOVE CRITERIA OF CONVERSION-MOTION. THEN THE IVARIABLE OF THAT CONVERSION WILL HAVE AN OBVIOUS FORM IFF ALL OCCURENCES IN BFROM≤VO≥ THAT CAN REACH B HAVE THE SAME FORM. OUR PROBLEM IS TO DETERMINE, WITHOUT A FULL DATA FLOW ANALYSIS, WHICH OCCURENCES IN BFROM≤VO≥ CAN REACH B. IF VO1 → BFROM≤VO≥ IS NOT CONTAINED IN INT, THEN IT MUST CERTAINLY REACH B, BUT IF VO1 IS INSIDE INT, (AND SINCE VO IS IN THE LOOP-PART OF INT THERE WILL BE AT LEAST ONE SUCH OCCURENCE), THEN THERE IS NO SIMPLE CRITERION TO DETERMINE WHETHER VO1 CAN ALSO REACH VO THROUGH B. MOREOVER, THIS OCCURENCE WILL BE EQREPR TO VO, WHEREAS OTHER OCCURENCES IN BFROM≤VO≥ WILL NOT BE, AND SO, IF WE CAN NOT IGNORE THESE INSIDE LINKS IN DETERMINING THE REPR OF THE IVARIABLE OF THE CONVERSION, WE WILL ALWAYS HAVE TO REPR IT AS A GENERAL TYPE, WHICH IS CERTAINLY UNDESIRABLE.

IT IS RATHER DIFFICULT TO FIND A GENERAL NECESSARY AND SUFFICIENT CONDITION TO DETERMINE THE ABSENCE OF SUCH INSIDE LINKS. THE CONDITION THAT OUR ALGORITHM WILL USE (CONDITION (**), SEE BELOW) IS SOMEWHAT PESSIMISTIC, AND IS ONLY SUFFICIENT, BUT IT GIVES AN ADEQUATE ANSWER IN MOST CASES, THOUGH IT MAY CHOOSE, IN SOME RATHER RARE CASES, A GENERAL IVARIABLE FOR A CONVERSION UNNECESSARILY.

LET US NOW SKETCH THE CONVERSION INSERTION PHASE OF THE NAME-SPLITTING ALGORITHM:

THE FIRST PHASE OF THE ALGORITHM WILL HAVE COMPUTED A WORKPILE OF OCCURENCES THAT ARE LINKED BY BFROM TO DIFFERENT VARIABLES OF THE SAME SPLIT GROUP.

```
(∀ VO → WORKPILE)

     IF VO DOES NOT SATISFY THE SAFETY CONDITION (*) THEN

          IF ALL VO1 → BFROM≤VO≥ HAVE THE SAME FORM THEN
               VI := SPLIT¬NAME(ARB BFROM≤VO≥);
          ELSE
               VI := [OI¬NAME(VO), GENERAL];
          END IF;

          INSERT BEFORE THE INSTRUCTION OF VO THE CONVERSION
          SPLIT¬NAME(VO) := VI;

     ELSE

          COMPUTE INTSEQ, THE SEQUENCE OF INTERVALS CONTAINING
          VO, STARTING AT ITS BASIC BLOCK.

          FIND THE LARGEST INDEX J SUCH THAT INT := INTSEQ(J)
          DOES NOT CONTAIN ANY OCCURENCE IN THE SET
          ≤ VO1 → BFROM≤VO≥ ↑ NOT (VO1 .EQREPR VO)≥
          AND INTSEQ(J-1) IS IN THE LOOP-PART OF INT (THAT IS,
          HEAD(INT) CAN BE REACHED FROM INTSEQ(J-1) ALONG A
          PATH WHOLLY CONTAINED IN INT).

          (INT IS COMPUTED IN TWO STEPS; FIRST, FIND THE LARGEST
          INDEX K SUCH THAT INTSEQ(K) DOES NOT CONTAIN ANY
          OCCURENCE IN THE ABOVE SET, THEN, FIND THE LARGEST INDEX
          J <= K, SUCH THAT INTSEQ(J-1) IS IN THE LOOP-PART OF
          INTSEQ(J). SET INT := INTSEQ(J).)

          IF ALL VO1 → BFROM≤VO≥ ↑ INT DOES NOT CONTAIN VO1
          HAVE THE SAME FORM, AND THE FOLLOWING CONDITION IS
          SATISFIED,

       (**)   THERE EXISTS M <= ↓INTSEQ SUCH THAT ALL OCCURENCES
              VO1 → BFROM≤VO≥ THAT ARE OUTSIDE INT, ARE CONTAINED
              IN INTSEQ(M) BUT NOT IN INTSEQ(M-1)

          THEN
               VI := SPLIT¬NAME(ARB (THE ABOVE SET));
          ELSE
               VI := [OI¬NAME(VO), GENERAL];
          END IF;
```

```
          INSERT AT THE END OF THE TARGET BLOCK OF INT THE
          CONVERSION   SPLIT-NAME(VO) := VI;  UNLESS THERE
          IS ALREADY SUCH A CONVERSION IN THAT BLOCK, IN WHICH
          CASE BOTH CONVERSIONS ARE MERGED INTO ONE.

     END IF;
END ∀;
```

LET US FIRST JUSTIFY THE USE OF CONDITION (**) IN OUR ALGORITHM. WE ASSUME THAT THE ANALYZED PROGRAM HAS THE PROPERTY THAT ALL THE VARIABLES ARE INITIALIZED BEFORE THEY ARE EVER USED (LOCAL VARIABLES ARE INITIALIZED AT THE ENTRY TO THEIR PROCEDURE, AND GLOBAL VARIABLES AT THE ENTRY OF THE MAIN PROGRAM). THUS, ANY (STATIC) EXECUTION PATH LEADING FROM THE PROGRAM ENTRY TO A USE OF A VARIABLE, MUST CONTAIN A DEFINITION OF THAT VARIABLE.

PROPOSITION: UNDER THE ABOVE ASSUMPTION, LET VO → WORKPILE, AND LET INTSEQ, INT BE AS COMPUTED IN THE ABOVE ALGORITHM FOR VO. IF CONDITION (**) HOLDS FOR VO, THEN THERE CANNOT EXIST VO2 → BFROM≤VO≥ WHICH IS INSIDE INT AND REACHES THE TARGET BLOCK OF INT.

PROOF: LET V BE THE VARIABLE OF VO. OBSERVE THAT THERE EXISTS A V-FREE PATH LEADING FROM HEAD(INTSEQ(M-1)) TO VO (OBVIOUS), BUT THERE DOES NOT EXIST A V-FREE PATH LEADING FROM HEAD(INTSEQ(M)) TO VO, FOR OTHERWISE, BY OUR ASSUMPTION, THERE SHOULD BE AN OCCURENCE IN BFROM≤VO≥ OUTSIDE INTSEQ(M), CONTRARY TO CONDITION (**).

SUPPOSE THAT SUCH VO2 EXISTS. IT FOLLOWS THAT ANY V-FREE PATH FROM VO2 TO TB(INT) (THE TARGET BLOCK OF INT) IS CONTAINED IN INTSEQ(M-1). HENCE, THERE EXISTS SOME K <= M-1 SUCH THAT THIS PATH IS WHOLLY CONTAINED IN INTSEQ(K), BUT NOT WHOLLY CONTAINED IN INTSEQ(K-1). INTSEQ(K) STRICTLY CONTAINS INT, BECAUSE TB(INT) IS CONTAINED IN INTSEQ(K) AND IS NOT CONTAINED IN INT. THIS IMPLIES THAT INTSEQ(K-1), WHICH CONTAINS INT, IS IN THE LOOP-PART OF INTSEQ(K). ALSO, BY CONDITION (**) AND THE DEFINITION OF INT, INTSEQ(K) EXCLUDES ALL OCCURENCES VO1 → BFROM≤VO≥ THAT ARE NOT EQREPR TO VO. THE LAST TWO OBSERVATIONS IMPLY THAT THE INDEX J CONSTRUCTED BY OUR ALGORITHM FOR VO MUST BE >= K, SO THAT INTSEQ(K) IS CONTAINED IN, OR EQUAL TO INT, WHICH IS A CONTRADICTION.

Q. E. D.

REMARK: THIS CONDITION IS EVIDENTLY SATISFIED IF ONLY ONE OCCURENCE IN BFROM≤VO≥ IS NOT EQREPR TO VO AND IS OUTSIDE SOME INTERVAL CONTAINING VO AND ALL OTHER OCCURENCES IN BFROM≤VO≥. THIS WILL HAPPEN IN THE OVERWHELMING MAJORITY OF CASES. EVEN WHEN THIS IS NOT THE CASE, SUCH OCCURENCES ARE MORE LIKELY TO OCCUR IN THE SAME LOOP NESTING LEVEL AND NEAR EACH OTHER, SO THAT CONDITION (**) IS VERY LIKELY TO HOLD.

LET US NOW SHOW THAT THE CONVERSION INSERTION ALGORITHM DOES ELIMINATE ALL LINKS BETWEEN DIFFERENT SPLIT VARIABLES.

THEOREM: AT THE END OF THE CONVERSION INSERTION ALGORITHM, IF WE REPLACE THE VARIABLE NAMES BY THEIR ORIGINAL, UNSPLIT NAMES, AND RE-COMPUTE THE BFROM MAP, THEN, FOR EACH OCCURENCE VO, AND EACH VO1 ∈ BFROM≤VO≥, VO1 EQREPR VO, UNLESS VO IS OF TYPE GENERAL.

PROOF: WE FIRST INTRODUCE SEVERAL AUXILIARY NOTATIONS THAT WE WILL USE IN THE PROOF. LET US ENUMERATE WORKPILE AS ≤VO1, VO2 ,.., VON≥, IN THE ORDER IN WHICH OUR ALGORITHM ITERATES OVER THIS SET. LET INTJ BE THE INTERVAL INT COMPUTED AT THE J-TH ITERATION (INTJ := OM IF VOJ DOES NOT SATISFY (*)). LET CONVJ DENOTE THE CONVERSION INSERTED INTO THE CODE AT THE J-TH ITERATION, AND LET OVCJ (RESP. IVCJ) DENOTE THE OVARIABLE (RESP. THE IVARIABLE) OF THAT CONVERSION. LET VJ DENOTE THE VARIABLE OF THE OCCURENCE VOJ, J := 1 ,.. N. LET ≠INSIDE≠ DENOTE A RELATION BETWEEN INTERVALS, SUCH THAT INTA INSIDE INTB IFF EACH BASIC BLOCK OF INTA IS ALSO A BASIC BLOCK OF INTB. LET US ALSO USE THE SAME NOTATION FOR A RELATION BETWEEN VARIABLE OCCURENCES AND INTERVALS, DEFINED SO THAT IF VO IS A VARIABLE OCCURENCE, THEN VO INSIDE INT MEANS THAT THE BASIC BLOCK CONTAINING VO IS INSIDE INT. ≠BFROM≠ WILL DENOTE THE ORIGINAL BFROM MAP, AND ≠NEW-BFROM≠ WILL DENOTE THE RE-COMPUTED MAP AT THE END OF THE ALGORITHM, ASSUMING THAT THE ORIGINAL VARIABLE NAMES ARE RESTORED, BUT WITH THE INSERTED CONVERSIONS STILL PRESENT IN THE CODE.

LEMMA A: IF THERE IS A PATH LEADING FROM SOME CONVJ TO ANOTHER OCCURENCE VO OF THE VARIABLE VJ, FREE OF OTHER (ORIGINAL) OCCURENCES OF THAT VARIABLE, THEN THERE EXISTS VOP INSIDE INTJ, VOP ∈ BFROM≤VO≥ * BFROM≤VOJ≥ AND VOP .EQREPR VOJ.

PROOF: INTJ /= OM, FOR IF VOJ DOES NOT SATISFY (*), THEN NO SUCH PATH CAN EXIST. FIGURE (1) BELOW ILLUSTRATES THE SITUATION

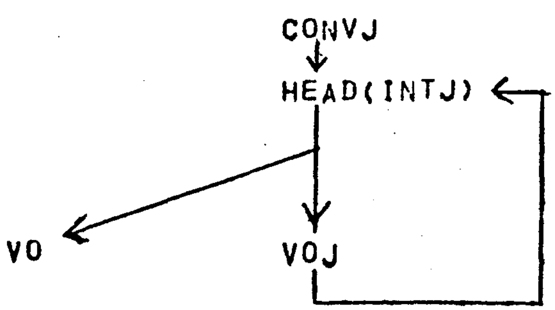


FIGURE (1)

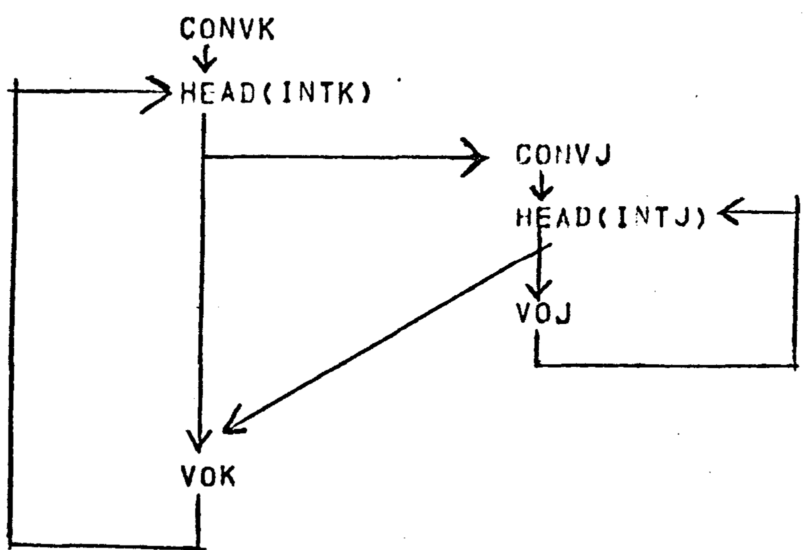
LET P1 BE A CYCLIC EXECUTION PATH OF THE FORM HEAD(INTJ) ... VOJ ..., HEAD(INTJ), SUCH THAT ITS INITIAL SUBPATH FROM HEAD(INTJ) TO VOJ IS VJ-FREE (SUCH A PATH EXISTS BECAUSE VOJ IS IN THE LOOP-PART OF INTJ, AND CONVJ CAN REACH VOJ THROUGH

HEAD(INTJ)), AND LET P2 BE A VJ-FREE PATH FROM HEAD(INTJ) TO VO. LET P := P1 + P2. P IS NOT VJ-FREE, FOR VOJ LIES ON IT. LET VOP BE THE LAST OCCURENCE OF VJ ON P (BEFORE VO). SINCE P2 IS VJ-FREE, VOP IS INSIDE INTJ, AND THE TERMINAL SUBPATH OF P, VOP ..., HEAD(INTJ) ..., VO IS VJ-FREE, HENCE VOP → BFROM≤VO≥ * BFROM≤VOJ≥, AND BY THE DEFINITION OF INTJ VOP .EQREPR VOJ.

Q. E. D.

WE NOW PROCEED WITH THE PROOF OF THE THEOREM, IN THE FOLLOWING STEPS

(1) FOR EACH K := 1 ... N, NEW¬BFROM≤VOK≥ CONTAINS ONLY OCCURENCES EQREPR TO VOK. INDEED, IF INTK = OM, THEN NEW¬BFROM≤VOK≥ = ≤OVCK≥, AND THIS OCCURENCE IS EQREPR TO VOK BY DEFINITION. OTHERWISE, BY THE DEFINITION OF INTK, EACH OCCURENCE IN NEW¬BFROM≤VOK≥ MUST BE EITHER OVCK, OR AN ORIGINAL OCCURENCE OF VK INSIDE INTK, EQREPR TO VOK, OR AN CVARIABLE OVCJ OF ANOTHER CONVERSION, INSERTED INSIDE INTK. THUS, IT IS SUFFICIENT TO SHOW THAT NEW¬BFROM≤VOK≥ CANNOT HAVE ANY MEMBER OF THE FORM OVCJ. INDEED, SUPPOSE THAT THERE EXISTS A CONVERSION CONVJ SUCH THAT VJ = VK, CONVJ INSIDE INTK AND THERE IS A VK-FREE PATH FROM CONVJ TO VOK, AS ILLUSTRATED IN FIGURE (2) BELOW



FIGURE(2)

(WITH NO LOSS OF GENERALITY WE MAY ASSUME THAT NO OTHER CONVERSION OF VK APPEARS ALONG THIS PATH.) WE HAVE TO CONSIDER TWO CASES: EITHER VOK NOT EQREPR VOJ, OR ELSE THESE OCCURENCES ARE EQREPR. FIGURE (2) ASSUMES THAT CONVJ, AND CONSEQUENTLY ALL INTJ, ARE STRICTLY INSIDE INTK. HOWEVER, IT IS ALSO POSSIBLE THAT INTJ = INTK AND CONVJ SUCCEEDS CONVK IN THE SAME TARGET BLOCK.

ASSUME FIRST THAT VOK NOT EQREPR VOJ, THEN, IN THE CONFIGURATION OF FIGURE (2), IT FOLLOWS FROM LEMMA A THAT THERE EXISTS VO → BFROM≤VOK≥, VO INSIDE INTJ AND VO EQREPR VOJ. HENCE, VO INSIDE INTK AND IS NOT EQREPR TO VOK, CONTRADICTING THE DEFINITION OF INTK. A SIMILAR ARGUMENT SHOWS THAT THE OTHER CONFIGURATION MENTIONED ABOVE IS ALSO IMPOSSIBLE IN THIS CASE.

NOW, ASSUME THAT VOK EQREPR VOJ. WE CLAIM THAT IN THIS CASE INTJ = INTK. INDEED, IF NOT, THEN OBSERVE FROM FIGURE (2) THAT INTJ MUST BE IN THE LOOP PART OF INTK. HENCE, BY THE DEFINITION OF INTJ, THERE MUST EXIST VO → BFROM≤VOJ≥, VO INSIDE INTK BUT OUTSIDE INTJ, AND VO NOT EQREPR VOJ (FOR IF NO SUCH OCCURENCE EXISTS, THEN THE FACT THAT INTJ IS IN THE LOOP-PART OF INTK IMPLIES THAT A LARGER INTERVAL THAN INTJ COULD HAVE BEEN CHOSEN IN THE J-TH ITERATION OF OUR ALGORITHM). IT ALSO FOLLOWS FROM FIGURE (2) THAT VO → BFROM≤VOK≥, BECAUSE VO CAN REACH VOK ALONG THE CONCATENATION OF THE VK-FREE PATH LEADING FROM VO TO HEAD(INTJ) WITH THE TERMINAL SUBPATH OF THE VK-FREE PATH LINKING CONVJ WITH VOK, WHICH STARTS AT HEAD(INTJ). BUT VO IS OBVIOUSLY NOT EQREPR TO VOK. THIS CONTRADICTS THE DEFINITION OF INTK, AND SO INTJ = INTK. BUT IN THIS CASE, SINCE VOJ EQREPR VOK, THE ALGORITHM WOULD HAVE MERGED CONVK WITH CONVJ, THUS THE ASSERTION IS PROVED.

(2) LET VO BE AN ORIGINAL OCCURENCE OF SOME VARIABLE V, NOT PLACED IN WORKPILE. WE CLAIM THAT IF CONVJ IS ANY CONVERSION OF V, FROM WHICH, AT THE END OF THE ALGORITHM, THERE MAY EXIST A V-FREE PATH TO VO, THEN THE OUTPUT VARIABLE OVCJ OF CONVJ IS TO VO. INDEED, IT FOLLOWS FROM LEMMA A THAT THERE EXISTS VOP → BFROM≤VO≥ SUCH THAT VOP EQREPR VOJ. HENCE, EITHER OVCJ EQREPR VO, OR ELSE VOJ IS NOT EQREPR TO VO, AND HENCE VOP IS NOT EQREPR TO VO, THUS VO MUST INITIALLY HAVE BELONG TO THE WORKPILE. THIS IS A CONTRADICTION, AND IT FOLLOWS THAT NEW-BFROM≤VO≥ CONTAINS ONLY OCCURENCES EQREPR TO VO.

(3) IT NOW REMAINS TO PROVE THE THEOREM FOR THE IVARIABLES OF THE CONVERSIONS. LET IVCK BE SUCH AN IVARIABLE. IF IT IS OF TYPE GENERAL, THEN THERE IS NOTHING TO PROVE. OTHERWISE, BY THE DEFINITION OF IVCK, EACH OCCURENCE IN NEW-BFROM≤IVCK≥ IS EITHER EQREPR TO IVCK, OR ELSE IS AN OVARIABLE OF SOME OTHER CONVERSION. THE SECOND CASE IS ILLUSTRATED AS FOLLOWS:

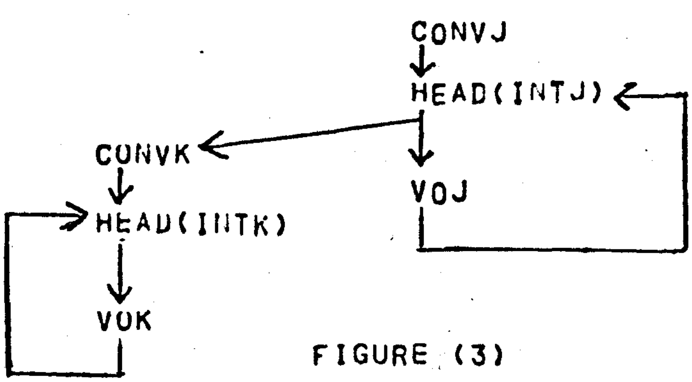


FIGURE (3)

(HERE NO RELATIONSHIP OF INCLUSION BETWEEN THE TWO INTERVALS IS IMPLIED. NOTE THAT FIGURE(3) EXCLUDES THE CASE WHERE CONVK AND CONVJ ARE IN THE SAME TARGET BLOCK, FOR THIS CASE HAS BEEN SHOWN, IN STEP (1), TO BE IMPOSSIBLE.)

SINCE VOK CAN BE REACHED ALONG A VK-FREE PATH FROM CONVK, IT FOLLOWS FROM LEMMA A THAT THERE EXISTS VOP → BFROM≤VOK≥ WHICH REACHES VOK THROUGH CONVK, SUCH THAT VOP INSIDE INTJ AND VOP EQREPR VOJ. HENCE, IVCK EQREPR VOP EQREPR VOJ EQREPR OVCJ, SO THAT NEW¬BFROM≤IVCK≥ INDEED CONTAINS ONLY OCCURENCES EQREPR TO IVCK, AND THIS CONCLUDES THE PROOF OF THE THEOREM.

Q. E. D.

REMARKS:
----

(1) OUR ALGORITHM IS INTRA-PROCEDURAL IN NATURE. THE BFROM MAP, HOWEVER, IS INTER-PROCEDURAL, INDICATING FOR EACH LINK THE RC¬PATH(S) THROUGH WHICH THIS LINK IS MATERIALIZED. WE SHALL INTERPRET BFROM AS AN INTRA-PROCEDURAL MAP, IN THE SAME WAY AS WE DID IN COPY OPTIMIZATION, AS FOLLOWS:

ASSUME THAT RC¬PATHS ARE COMPACTED, SO THAT COMPLETE CALLS ARE DELETED FROM THEM. LET VO BE A VARIABLE OCCURENCE AND LET [P, VO1] → BFROM≤VO≥. THEN [P, VO1] IS INTERPRETED AS VO1, IF P = NULL¬PATH. ELSE, IF P TERMINATES AT A CALL POINT, [P, VO1] IS INTERPRETED AS A DUMMY OCCURENCE JUST AFTER THE ENTRY TO THE CURRENTLY ANALYZED ROUTINE, AND IF P TERMINATES AT A RETURN POINT, [P, VO1] IS INTERPRETED AS A DUMMY OCCURENCE JUST AFTER THE CORRESPONDING CALLING INSTRUCTION.

(2) NOTE THAT, THOUGH WE BASE OUR ALGORITHM ON INTERVAL ANALYSIS, THE ROUTINE FLOW GRAPH NEED NOT BE REDUCIBLE (COMPARE WITH COPY OPTIMIZATION, NL.195). FOR EXAMPLE, WHEN CHECKING CONDITION (**), WE SHALL MAKE USE OF A ROUTINE THAT COMPUTES THE INDEX OF THE SMALLEST INTERVAL IN INTSEQ, CONTAINING VO AND SOME VO1 → BFROM≤VO≥. IF NO SUCH INTERVAL EXISTS, THE ROUTINE WILL RETURN #INTSEQ + 1 (THIS WILL NOT HAPPEN, HOWEVER, IN REDUCIBLE FLOW GRAPHS).

(3) OUR ALGORITHM SELECTS THE GENERAL TYPE FOR AN IVARIABLE OF A CONVERSION LINKED TO MORE THAN ONE SPLIT VARIABLE. THIS IS DONE IN ORDER TO SIMPLIFY THE DESCRIPTION OF THE ALGORITHM, BUT IS NOT ALWAYS THE BEST CHOICE. FOR EXAMPLE, IF SUCH AN IVARIABLE IVCK IS LINKED TO TWO OCCURENCES, HAVING THE REPRS SET(INT), SET(CHAR), THEN A BETTER CHOICE WOULD HAVE BEEN TO REPR IVCK AS SET(GENERAL). THIS WILL MAKE THE CONVERSION SOMEWHAT FASTER.

IN THIS ALTERNATIVE APPROACH, THE FORM OF EACH SUCH IVARIABLE IVCK IS COMPUTED AS A DISJUNCTION OF THE REPRS OF ALL OCCURENCES

TO WHICH IVCK IS LINKED. SUCH A DISJUNCTION MUST SATISFY THE CONDITION THAT NO CONVERSION WILL BE REQUIRED FROM ANY OF THE FORMS OF THE LINKED OCCURENCES TO THE MORE GENERAL FORM OF IVCK (THUS THE DISJUNCTION OF →B AND INT MUST BE GENERAL, EVEN IF B IS A BASE OF INTEGERS). THE ABOVE THEOREM NOW READS AS FOLLOWS:

THEOREM C: UNDER THE SAME HYPOTHESIS AS IN THEOREM B, FOR EACH VARIABLE OCCURENCE VO AND EACH VO1 → BFROM≤VO≥, VO EQREPR VO1 IF VO IS NOT AN IVARIABLE OF AN INSERTED CONVERSION; IF IT IS, THEN THE FORM OF VO INCLUDES THAT OF VO1, IN THE SENSE THAT THE ASSIGNMENT ≠SPLIT¬NAME(VO) := SPLIT¬NAME(VO1);≠ IS A NO-OP.

THE PROOF OF THEOREM C GOES IN MUCH THE SAME WAY AS THE PROOF OF THEOREM B, WITH A SLIGHT MODIFICATION OF STEP (3).