SETL NEWSLETTER 197

SOME COMMENTS ON EXTENDING CODE MOTION AND EXPRESSION AVAILABILITY ALGORITHMS FOR THE SETL OPTIMIZER.

MICHA SHARIR
SEP 30 1977

IN THIS NOTE WE SHALL OUTLINE SEVERAL ASPECTS AND OBSERVATIONS RELEVANT TO CODE MOTION AND EXPRESSION AVAILABILITY ANALYSIS OF SETL PROGRAMS. WE SHALL ALSO SUGGEST SEVERAL POSSIBLE EXTENSIONS AND GENERALIZATIONS OF THESE ANALYSES, AND HINT ON POSSIBLE APPROACHES TO THESE PROBLEMS.

THE AIM OF THIS NOTE IS MERELY TO DOCUMENT AND CLARIFY THE PRESENT STATE OF THE DESIGN AND IMPLEMENTATION OF SUCH ALGORITHMS IN THE SETL OPTIMIZER. AS WILL BE NOTED BELOW, WE STILL HAVE TO MAKE A DECISION AS TO WHAT EXTENT WE WISH TO CARRY OUT THOSE OPTIMIZATIONS, AND HOW TO IMPLEMENT THEM EFFICIENTLY. SOME OF THE COMMENTS BELOW ARE SIMPLY GUIDING PRINCIPLES IN THE MAKING OF SUCH A DECISION.

1) POTENTIAL COMPUTATIONS FOR ELIMINATION AND MOTION.

ANY SETL OPERATION WITH A UNIQUE OVARIABLE, WHICH IS DETERMINED UNIQUELY AND IN A DETERMINISTIC FASHION FROM THE IVARIABLES, WITHOUT ANY SIDE-EFFECTS, EXCLUDING ASSIGNMENT-LIKE OPERATIONS, WILL BE CALLED AN EXPRESSION COMPUTATION. THE OVARIABLE NAME WILL BE A TEMPORARY NAME, UNIQUELY DETERMINED BY THE OPERATION AND THE IVARIABLE NAME. THUS, FOR EXAMPLE, CALLS, BRANCHES, ASSIGNMENTS, MAP AND TUPLE STORAGES, ETC, ARE NOT EXPRESSION COMPUTATIONS. A COMPUTATION, FOR OUR PURPOSE, WILL BE EITHER AN EXPRESSION COMPUTATION, OR A SIMPLE ASSIGNMENT, OR AN ASSIGNMENT-LIKE OPERATION. THESE COMPUTATIONS WILL BE THE ONLY POTENTIAL CANDIDATES FOR CODE MOTION AND COMMON SUBEXPRESSION ELIMINATION.

2) SAFETY CONSIDERATIONS OF MOVING EXPRESSION COMPUTATIONS.

WE IGNORE ALTOGETHER THE PROBLEM OF SAFETY OF MOTION OF EXPRESSION COMPUTATIONS. AS THE COMPILER IS DESIGNED, THERE WILL BE SEVERAL MODES OF EXECUTING SETL PROGRAMS. THE ONLY PERMITTED MODE FOR PROGRAMS TO WHICH THE CODE MOTION OPTIMIZATION WILL BE APPLIED IS THE ONE IN WHICH FATAL RUN TIME ERRORS, STEMMING FROM ILLEGAL ARGUMENTS OF AN OPERATION, WILL NOT CAUSE AN IMMEDIATE ABORTION OF THE JOB, BUT RATHER ALLOW THE EXECUTION TO BE CONTINUED, WITH THE OVARIABLE OF THE OPERATION RECEIVING AN ERROR VALUE, AS LONG AS THE PROGRAM FLOW IS WELL DEFINED. ANY COMPUTATION INVOLVING AN OBJECT WITH AN ERROR VALUE WILL YIELD AN ERROR

VALUE AS A RESULT. THE EXECUTION WILL BE ABORTED WHEN A BRANCH DEPENDING ON AN ERROR VALUE IS EXECUTED. THUS, IT IS SAFE TO MOVE E,G, A DIVISION OUT OF A LOOP, EVEN THOUGH IT MIGHT NOW HAVE A ZERO DIVISOR, PROVIDED THAT IN THIS CASE, THE ERROR VALUE OF THE QUOTIENT IS NEVER USED LATER ON. THUS, ANY EXPRESSION COMPUTATION CAN BE MOVED OUT OF A LOOP, OR EVEN INSERTED ANYWHERE IN THE PROGRAM, WITHOUT ANY HARM. (USUALLY, CODE MOTION ALGORITHMS DO NOT MOVE CODE OUT OF LOOPS, BUT RATHER INSERT THE CODE INTO THE TARGET BLOCK OF THE LOOP, AND LET THE COMMON SUBEXPRESSION ELIMINATION PHASE DO THE REST.) THUS, WHEN WE SAY THAT AN EXPRESSION COMPUTATION CAN BE MOVED, WE ALWAYS MEAN MOVED PROFITABLY, AND THIS WILL BE DEFINED BELOW.

3) PROFITABILITY OF CODE MOTION.

USUALLY, THE PROFITABILITY CRITERION FOR CODE MOTION OUT OF A LOOP, IS THAT EVERY MOVED COMPUTATION WILL BE UNCONDITIONALLY EXECUTED WITHIN THE LOOP FOR EVERY EXECUTION FLOW. THIS WILL ENSURE THAT THE NUMBER OF TIMES THIS COMPUTATION IS EXECUTED WILL NEVER INCREASE, AND WILL POTENTIALLY DECREASE SUBSTANTIALLY. HOWEVER, THE SETL STRUCTURED LOOPS ARE SUCH THAT THE TEST FOR LOOP TERMINATION IS PERFORMED AT THE BEGINNING OF EACH LOOP. THUS, CONSIDERING THE POSSIBILITY THAT SOME LOOPS MIGHT NOT BE EXECUTED AT ALL, FOR A PARTICULAR RUN OF THE PROGRAM, THIS CRITERION IS TOO STRONG, FOR EACH COMPUTATION WITHIN A LOOP MIGHT POTENTIALLY NEVER BE EXECUTED, AND SO THIS CRITERION WILL YIELD NEXT TO NO MOVABLE COMPUTATIONS AT ALL.

THUS, OUR COMPROMISE IS TO MOVE OUT OF A LOOP COMPUTATIONS WHICH ARE UNCONDITIONALLY EXECUTED, ASSUMING THAT THE LOOP IS EXECUTED AT LEAST ONCE. THUS, THOUGH WE MIGHT COMPUTE EXTRA EXPRESSIONS AFTER THE CODE MOTION, WE WILL GAIN CONSIDERABLY ON THE AVERAGE.

4) HIGHER LEVEL OF CODE MOTION AND AVAILABILITY.

LET US DEMONSTRATE, THROUGH SOME EXAMPLES, THE LEVEL OF OPTIMIZATION THAT WE MIGHT WANT TO ACHIEVE -

1) IN THE FOLLOWING CODE FRAGMENT

```
(*)
    T1 := F(B);
    A := T1;
    T2 := H(A);
END *;
```

WE WOULD LIKE TO DERIVE THAT ALL THREE COMPUTATIONS CAN BE MOVED OUT OF THE LOOP. THERE ARE SOME COMPLICATIONS AND LIMITATIONS

ABOUT MOVING THE ASSIGNMENT OUT OF THE LOOP, AS WILL BE EXPLAINED BELOW, BUT WE MIGHT WANT TO FIND SOME MECHANISM THAT WILL ALLOW US TO MOVE THE COMPUTATIONS OUT.

2) CONSIDER THE FOLLOWING UNOPTIMIZED CODE

```
(∀Y → S)
    F(Y) := G(X);
    H(Y) := P(G(X));
END ∀;
```

WHICH WOULD EXPAND TO

```
(∀Y → S)
    T1 := G(X);
    F(Y) := T1;
    T1 := G(X);
    T2 := P(T1);
    H(Y) := T2;
END ∀;
```

AND ANY STANDARD ALGORITHM WILL BE ABLE TO MOVE THE COMPUTATION OF T1 AND T2 OUT OF THE LOOP, HOWEVER, AFTER A MANUAL SEMI-OPTIMIZATION, THE ORIGINAL CODE MIGHT LOOK SOMETHING LIKE THIS:

```
(∀ Y → S)
    A := G(X);
    F(Y) := A;
    H(Y) := P(A);
END ∀;
```

WHICH WOULD BE EXPANDED INTO

```
(∀Y → S)
    T1 := G(X);
    A := T1;
    F(Y) := A;
    T3 := P(A);
    H(Y) := T3;
END ∀;
```

AND SINCE THE STANDARD ALGORITHM WILL REGARD THE ASSIGNMENT A := T1; AS A KILL OF A AND OF ALL THE DEPENDENT EXPRESSIONS, IT WILL NOT PICK UP THE FACT THAT THE COMPUTATION OF T3 CAN BE MOVED OUT OF THE LOOP, TOGETHER WITH THE ABOVE ASSIGNMENT. THUS, WE PUNISH THE PROGRAMMER FOR HIS ATTEMPT TO SOMEWHAT OPTIMIZE HIS CODE. THIS SITUATION COULD ALSO OCCUR IF A IS A USER DEFINED TEMPORARY, USED TO SPLIT A LONG COMPUTATION INTO SEVERAL SUBEXPRESSIONS. AGAIN, WE MIGHT WANT TO BE ABLE TO MOVE ALL THESE COMPUTATIONS OUT OF LOOPS.

3) ONE MIGHT ALSO WISH TO GENERALIZE AVAILABILITY, IN A SIMILAR FASHION, CONSIDER THE FOLLOWING EXAMPLE.

```
IF COND THEN
    B := F(A);
    X := G(B);
ELSE
    C := F(A);
    Y := G(C);
END IF;
```

AFTER EXPANDING, WE WILL HAVE

```
IF COND THEN
    T1 := F(A);
    B := T1;
    T2 := G(B);
    X := T2;
ELSE
    T1 := F(A);
    C := T1;
    T3 := G(C);
    Y := T3;
END IF;
```

HERE, THE EXPRESSION G(F(A)) IS ACTUALLY COMPUTED UNCONDITIONALLY IN THIS IF STATEMENT (STORED, HOWEVER, UNDER TWO DIFFERENT TEMPORARY NAMES). WE MIGHT WANT TO DETECT THIS INFORMATION, SO THAT WE CAN ELIMINATE ANY FURTHER COMPUTATION OF THIS EXPRESSION. HERE WE REFER TO A DEEPER ANALYSIS, TRYING TO ANALYZE FURTHER THE CURRENT VALUE THAT EXPRESSIONS AND USER VARIABLES HAVE AT CERTAIN PROGRAM POINTS.

5) SAFETY CONSIDERATIONS OF MOVING ASSIGNMENTS.

THE SAFETY CONSIDERATIONS FOR MOVING ASSIGNMENTS OUT OF LOOPS ARE QUITE DIFFERENT FROM SUCH CONSIDERATIONS FOR MOVING OUT EXPRESSION COMPUTATIONS, AS DESCRIBED ABOVE. INDEED, IN GENERAL, ONE CANNOT MOVE AN ASSIGNMENT LIKE X := T; OUT OF A LOOP, SINCE, IF THE LOOP HAPPENS NOT TO BE EXECUTED AT ALL, THEN X IS ASSIGNED A VALUE, WHICH WOULD NOT BE ASSIGNED TO IT OTHERWISE. THERE ARE SEVERAL WAYS TO OVERCOME THIS LIMITATIONS, WHICH MIGHT SEVERELY AFFECT ANY EXTENDED ALGORITHM, ATTEMPTING TO TRACE INFORMATION PAST ASSIGNMENTS.

ONE WAY IS TO ALLOW THAT ASSIGNMENT TO BE MOVED OUT OF THE LOOP, ONLY IF X IS DEAD UPON ANY EXIT FROM THE LOOP, SO THAT THE NEW VALUE OF X WOULD NOT AFFECT THE EXECUTION OF THE REST OF THE PROGRAM. THIS LIMITS SOMEWHAT THE POWER OF CODE MOTION,

BUT FOCUSES ON CASES WHERE X IS A USER DEFINED TEMPORARY, USED TO SPLIT A LONG CALCULATION IN TWO, AND IS NOT NEEDED ELSEWHERE IN THE PROGRAM.

ANOTHER WAY IS TO MOVE THIS ASSIGNMENT OUT OF THE LOOP, BUT REPEAT ALL TESTS WITHIN THE LOOP WHICH ARE EXECUTED BEFORE THE ASSIGNMENT, ALSO IN THE TARGET BLOCK OF THIS LOOP, TO MAKE SURE THAT IF THE STATEMENT WAS ORIGINALLY NOT EXECUTED, IT SHALL NOT BE EXECUTED AFTER THE MOTION AS WELL, THIS WAY IS VERY AWKWARD AND DIFFICULT TO IMPLEMENT, EVEN WITH STRUCTURED LOOPS, THE TESTS FOR LOOP TERMINATION MAY BE QUITE INVOLVED, AND DIFFICULT TO LOCATE (CONSIDER, E.G., QUIT STATEMENTS SCATTERED THROUGHOUT THE LOOP), AND IN MANY CASES, TO REPEAT THOSE TESTS WILL REQUIRE TO ALMOST DUPLICATE THE LOOP BODY IN THE TARGET BLOCK (WHICH WILL NOT BE NOW A BLOCK ANYMORE). ALSO, ATTEMPTING TO MOVE THESE ASSIGNMENTS AND TESTS OUT OF TWO OR MORE NESTED LOOPS MIGHT INCREASE THE COMPLEXITY OF THE CODE TREMENDOUSLY. THUS, THIS WAY IS CERTAINLY NOT RECOMMENDED.

A THIRD WAY, WHICH IS ALSO RELATED TO THE GENERALIZED AVAILABILITY SUGGESTED ABOVE, IS TO EMPLOY A SCHEME OF SUBSTITUTIONS OF ONE EXPRESSION INTO ANOTHER, EXPLICITLY IN THE CODE, SO THAT THERE IS NO NEED TO MOVE ASSIGNMENTS AT ALL. LET US EXAMINE THE EXAMPLES GIVEN IN 4).

IN THE FIRST EXAMPLE, WE CAN MODIFY THE CODE TO LOOK LIKE

```
(v)
    T1 := F(B);
    A := T1;
    T3 := H(T1);
END v;
```

NOW, THE COMPUTATIONS OF T1 AND T3 CAN BE SAFELY MOVED OUT OF THE LOOP. THE ASSIGNMENT REMAINS IN THE LOOP, AND, IF A WAS INDEED A USER DEFINED TEMPORARY, THEN A SUBSEQUENT PHASE OF DEAD CODE ELIMINATION WILL ELIMINATE THE ASSIGNMENT.

SIMILARLY, IN THE SECOND EXAMPLE, THE SEMI-OPTIMIZED CODE WILL BE TRANSFORMED INTO SOMETHING LIKE THE FOLLOWING.

```
(vY → S)
    T1 := G(X);
    A := T1;
    F(Y) := T1;
    T4 := P(T1);
    H(Y) := T4;
END v;
```

AND AGAIN, THE COMPUTATIONS OF T1 AND T4 CAN BE MOVED OUT OF THE LOOP.

SIMILAR SUBSTITUTIONS CAN BE PERFORMED IN THE THIRD EXAMPLE, MAKING THE COMPUTATION OF G(F(A)) AVAILABLE, AND STORED UNDER THE SAME NAME, ON BOTH BRANCHES OF THE FLOW.

HOWEVER, SUCH A SCHEME HAS ALSO SHORTCOMINGS, AS DEMONSTRATED IN THE FOLLOWING EXAMPLE.

```
IF COND THEN
    A := X(I);
    B := F(A);
    C := G(A);
ELSE
    A := X(J);
    B := G(A);
    C := F(A);
END IF;

F(A) := F(A) + B;
G(A) := G(A) + C;
```

WITHOUT SUBSTITUTIONS, F(A) AND G(A) ARE BOTH AVAILABLE AFTER EXECUTING THE IF STATEMENT, HOWEVER, AFTER SUBSTITUTING X(I) AND X(J) INSTEAD OF A, NOTHING BECOMES AVAILABLE, AS THE VALUE OF F(A), FOR EXAMPLE, IS NOW STORED UNDER TWO DIFFERENT NAMES.

THUS, THERE IS A CONFLICT BETWEEN CODE MOTION AND AVAILABILITY, SO THAT SUBSTITUTIONS MIGHT COMPLICATE AND INTERFERE WITH AVAILABILITY ANALYSIS, EVEN THOUGH THEY ARE IDEAL FOR CODE MOTION.

6) CONCLUDING REMARKS.

THE CLASSICAL, INTERVAL-ORIENTED ALGORITHM FOR COMMON SUBEXPRESSION ELIMINATION AND CODE MOTION, DUE ORIGINALLY TO J. COCKE, AND DESCRIBED IN DETAIL BY K. KENNEDY IN ≠ON PROGRAMMING≠, HAS ALREADY BEEN IMPLEMENTED INTO THE SETL OPTIMIZER, AS A DEFAULT OPTION. WE ARE NOT SURE THAT THE EXTENDED CODE MOTION AND AVAILABILITY ANALYSIS WILL REALLY BE EFFECTIVE, THE EXTRA CODE OPTIMIZATION THAT MIGHT BE PICKED UP BY SUCH EXTENDED ALGORITHMS, MIGHT BE QUITE MARGINAL, AND APPLICABLE ONLY IN RARE SITUATIONS. HOWEVER, IF ONE IS REALLY WILLING TO EXTEND THE CODE MOTION AND AVAILABILITY ALGORITHM, THEN THE ABOVE REMARKS SUGGEST SEVERAL POSSIBLE APPROACHES, AMONG WHICH WE FAVOR EITHER A FULL SCALE OR A LIMITED RANGE SUBSTITUTION SCHEME, DEPENDING ON THE DESIRED AMOUNT OF EXTRA OPTIMIZATION.