SETL NEWSLETTER 187A
MORE ON INTER-PROCEDURAL DATA FLOW ANALYSIS

MICHA SHARIR
M A Y 1977

THIS NEWSLETTER PRESENTS AN ALGORITHM FOR INTER-PROCEDURAL DATA FLOW ANALYSIS. THE ALGORITHM COMPUTES A MAP ≠REACH≠ WHICH IS DEFINED ON THE SET ≠ENTRIES¬OF¬RETURNS≠ OF ALL PROGRAM POINTS WHICH LIE EITHER AT AN ENTRY TO A PROCEDURE, OR JUST AFTER A RETURN FROM ONE. FOR EACH SUCH INSTRUCTION INST, REACH(INST) IS THE SET OF ALL PAIRS < OI , STR > WHERE OI IS A VARIABLE OCCURENCE IN THE PROGRAM, AND STR IS A PROPER ≠RC-STRING≠ OF RETURNS AND CALLS (CF, NL187) FOR WHICH THERE EXISTS A PROGRAM PATH LEADING FROM THE OCCURENCE OI TO INST, FREE OF OCCURENCES OF THE SAME VARIABLE. NOTE THAT A SIMILAR MAP IS USUALLY COMPUTED FOR EACH BASIC BLOCK IN A PROGRAM, HOWEVER, SINCE WE WISH TO CONCENTRATE UPON THE INTER-PROCEDURAL ASPECTS OF THE FLOW ANALYSIS, AND SINCE IT IS EASY TO COMPUTE THE REACH MAP FOR ALL THE BLOCKS ONCE WE HAVE IT FOR ALL ENTRIES AND RETURNS, WE IGNORE OTHER INSTRUCTIONS TEMPORARILY. ANOTHER IMPORTANT FACTOR TO CONSIDER IS THE AMOUNT OF SPACE OCCUPIED BY THIS MAP, SHOULD IT BE COMPUTED FOR EACH BASIC BLOCK, AN ISSUE THAT WILL BE DISCUSSED LATER.

WE ASSUME THAT THE PROCEDURE VARIABLES HAVE ALREADY BEEN DETERMINED AND THAT THE CALL GRAPH HAS BEEN COMPUTED. WE REPRESENT THE CALL GRAPH AS A MAP ≠CALLEDGE≠ WHOSE ELEMENTS ARE THE PAIRS < CALLING¬INSTRUCTION , CALLED¬PROCEDURE >. NOTE THAT THIS MAP IS SINGLE-VALUED IF THERE ARE NO PROCEDURE VARIABLES. LET ≠RETURNEDGE≠ BE THE INVERSE OF CALLEDGE.

LET US ALSO INTRODUCE THE SET ≠EXITS¬OR-CALLS≠ WHICH, IN A SENSE, IS COMPLEMENTARY TO ENTRIES¬OR¬RETURNS; IT IS THE SET OF ALL PROGRAM POINTS WHICH LIE IMMEDIATELY BEFORE EITHER A PROCEDURE CALL OR AN EXIT FROM A PROCEDURE.

IT SHOULD BE NOTED THAT INTER-PROCEDURAL DATA FLOW ANALYSIS IS MEANINGFUL ONLY FOR VARIABLES THAT APPEAR IN MORE THAN ONE PROCEDURE. THESE ARE ESSENTIALLY THE GLOBAL VARIABLES. (AND POSSIBLY ALSO THE FORMAL PARAMETERS OF PROCEDURES, IF WE MAKE THE Q1¬CODE REPRESENT DIRECT ASSIGNMENTS OF ACTUAL ARGUMENTS TO FORMAL PARAMETERS BEFORE A PROCEDURE CALL, AND CORRESPONDING INVERSE ASSIGNMENTS AFTER A RETURN FROM A PROCEDURE.)

THUS, THE INTER-PROCEDURAL ALGORITHM THAT WE ARE GOING TO DESCRIBE WILL DEAL ONLY WITH GLOBAL VARIABLES AND FORMAL PARAMETERS AND WILL IGNORE THE LOCAL VARIABLES. FROM NOW ON, THE TERM VARIABLE REFERS ONLY A GLOBAL VARIABLE OR A FORMAL PARAMETER.

OUR ALGORITHM USES THREE MAPS WHICH CAN BE OBTAINED BY A STRICTLY INTRA-PROCEDURAL ANALYSIS. THESE MAPS ARE DEFINED ON THE SET EXITS-OR-CALLS OR ON THE PRODUCT OF ENTRIES-OR-RETURNS WITH IT.

REACHIN (INST) - THE SET OF ALL VARIABLE OCCURENCES OI IN THE PROCEDURE OF INST FROM WHICH INST CAN BE REACHED VIA A PATH FREE BOTH OF OCCURENCES OF THE VARIABLE V OF OI AND OF PROCEDURE CALLS AND RETURNS. HERE, INST IS A MEMBER OF EXITS-OR-CALLS.

PCFP (PROCEDURE CONTROL FLOW PATHS) - CONTAINS ELEMENTS OF THE FORM [INS1,INS2] WHERE INS1 IS IN ENTRIES-OR-RETURNS, INS2 IS IN EXITS-OR-CALLS, AND THERE IS A PATH LEADING FROM INS1 TO INS2 WHICH IS FREE OF PROCEDURE CALLS AND RETURNS. THUS, BOTH INS1 AND INS2 ARE IN THE SAME PROCEDURE.

NOREACH (INS1 , INS2) - IS NON-VOID ONLY IF [INS1,INS2] → PCFP. IT IS THEN THE SET OF ALL VARIABLE NAMES SUCH THAT ON ANY PATH FROM INS1 TO INS2, WHICH IS FREE OF PROCEDURE CALLS AND RETURNS, THESE VARIABLES ARE USED OR MODIFIED.

THE INTRA-PROCEDURAL ANALYSIS THAT WILL PRODUCE THESE MAPS WILL MAKE USE OF AN INTRA-PROCEDURAL VERSION OF BFROM WHICH IS VERY CLOSE TO THE FINAL FORM OF THIS MAP. MORE PRECISELY, LET I BE AN IVARIABLE OCCURENCE IN A PROCEDURE P. WE DEFINE AUX-BFROM(I) TO BE THE SET CONTAINING

A) ALL OCCURENCES OI OF THE SAME VARIABLE IN P, FROM WHICH I CAN BE REACHED VIA A PATH FREE BOTH OF OTHER OCCURENCES OF THIS VARIABLE AND OF PROCEDURE CALLS AND RETURNS
B) ALL POINTS IN ENTRIES-OR-RETURNS WITHIN P, FROM WHICH I CAN BE REACHED BY A PATH SUBJECT TO THESE SAME CONSTRAINTS.

THE INTRA-PROCEDURAL ANALYSIS MIGHT AS WELL COMBINE THE COMPUTATION OF THE AUX-BFROM MAP WITH THE COMPUTATION OF THE ACTUAL BFROM MAP OF LOCAL VARIABLES, SINCE THESE COMPUTATIONS ARE VERY SIMILAR IN NATURE. TO ACHIEVE THIS MERGING OF BOTH COMPUTATIONS, WE HAVE TO REGARD A PROCEDURE CALL AS A USE OF ALL GLOBAL VARIABLES AND FORMAL PARAMETERS OF THE CALL, AND REGARD A RETURN FROM A PROCEDURE AS A DEFINITION OF ALL THESE VARIABLES. IN THIS WAY, PROCEDURE CALLS BECOME ≠TRANSPARENT≠ TO ALL NON-GLOBAL, NON-PARAMETER VARIABLES. IT THUS BECOMES POSSIBLE TO CALCULATE BOTH MAPS IN A SINGLE COMMON ALGORITHM. NOTE THAT IN THIS APPROACH, PROCEDURE CALLS DO NOT BREAK BASIC BLOCKS OR CHANGE THE INTRA-PROCEDURAL FLOW.

LET US ALSO DEFINE BFROM¬EXIT([VAR,INS]), WHERE VAR IS ANY VARIABLE NAME APPEARING IN P, AND INS IS ANY POINT IN EXITS¬OR¬CALLS WITHIN P, TO BE THE SET CONTAINING

A) ALL OCCURENCES OI OF VAR IN P, FROM WHICH INS CAN BE REACHED VIA A PATH FREE BOTH OF OTHER OCCURENCES OF VAR AND OF PROCEDURE CALLS AND RETURNS,

B) ALL POINTS IN ENTRIES¬OR¬RETURNS WITHIN P, FROM WHICH THERE EXISTS A PATH LEADING TO INS, AND SUBJECT TO THIS SAME CONSTRAINT

IF WE INSERTED A DUMMY USE I OF VAR AT INS, THIS SET WOULD BE IDENTICAL TO AUX¬BFROM(I))

IN TERMS OF THIS MAP, REACHIN≤INS≥ IS SIMPLY THE INTERSECTION OF THE SET OF OCCURENCES WITH THE UNION OF BFROM¬EXIT([VAR,INS]) TAKEN OVER ALL VARIABLES VAR WHICH APPEAR IN THE PROCEDURE THAT CONTAINS INS, AND VAR IS A MEMBER OF NOREACH≤INS1,INS2≥ IF AND ONLY IF BFROM¬EXIT([VAR,INS2]) DOES NOT CONTAIN INS1 (SO THAT EVERY PATH FROM INS1 TO INS2 CONTAINS AN OCCURENCE OF VAR).

MOREOVER, AFTER THE COMPUTATION OF REACH, IT IS STRAIGHT FORWARD TO CALCULATE THE ACTUAL BFROM MAP. INDEED, IF I IS AN IVARIABLE, THEN TO CONSTRUCT BFROM(I) ALL ONE HAS TO DO IS TO DELETE FROM AUX¬BFROM(I) ALL ELEMENTS OF ENTRIES¬OR¬RETURNS, AND REPLACE EACH DELETED ELEMENT INS BY THE SET ≤ [OI,STR] → REACH≤INS≥ ↑ VARID(OI) = VARID(I) ≥ ONE SHOULD ALSO ADJOIN THE NULL RC¬STRING TO EACH OF THE ORIGINAL VARIABLE OCCURENCES IN AUX¬BFROM(I).

WE SHALL ALSO REQUIRE A MAP EDGES¬FROM WHICH RESULTS FROM A DIRECT PROCESSING OF THE CALL GRAPH. IT CONTAINS ELEMENTS OF THE FORM [INS, [INS1,FLOW]] , WHERE INS → EXITS¬OR¬CALLS, INS1 → ENTRIES¬OR¬RETURNS, AND THERE IS A DIRECT PROCEDURE BOUNDARY CROSSING FROM INS TO INS1. FLOW IS A TRIPLE CHARACTERIZING THIS CROSSING, AND WILL EVENTUALLY BECOME PART OF AN RC¬STRING.
THE PRESENT FORM OF FLOW IS A TRIPLE [INS, Q, DIR], WHERE INS IS A CALLING INSTRUCTION, Q IS THE CALLED PROCEDURE, AND DIR DENOTES WHETHER THIS FLOW IS A CALL OR A RETURN.

HERE IS THE RECURSIVE DEFINITION OF REACH, WRITTEN IN TERMS OF THESE MAPS:

```
REACH ≤INST≥ = ≤ [OI,[FLOW]] : [INS,[INST,FLOW]] → EDGES¬FROM,
                               OI → REACHIN ≤INS≥ ≥

                + ≤ [OI,PROP¬CONCAT(STR,FLOW)] : [INS1,INS2] → PCFG,
                   [INS2,[INST,FLOW]] → EDGES¬FROM,
                   [OI,STR] → REACH≤INS1≥ ↑ VARID(OI) NOTIN
                                          NOREACH≤INS1,INS2≥ ≥
```

PROP-CONCAT CONCATENATES [FLOW] TO THE PROPER RC-STRING STR, IF THEY ARE COMPATIBLE, WITH THE UNDERSTANDING THAT IF THEY ARE NOT, THEN THE CORRESPONDING PAIR IS NOT INCLUDED IN THE SET ABOVE.

LET US EXPLAIN THE ABOVE EQUATION. WE TAKE ALL TRIPLETS INS1,INS2,INST SUCH THAT INS1,INST ARE IN ENTRIES-OR-RETURNS, INS2 IS IN EXITS-OR-CALLS, THERE IS A PATH FREE OF PROCEDURE CALLS AND RETURNS FROM INS1 TO INS2, AND THERE IS A DIRECT PROCEDURE BOUNDARY CROSSING FROM INS2 TO INST. THEN THE OCCURENCES AVAILABLE AT INST ARE, FIRST, THOSE WHICH REACH DIRECTLY INS2, MODIFIED BY THE SINGLETON RC-STRING [FLOW], AND, SECOND, THOSE OCCURENCES WHICH REACH INS1, MAY REACH INS2 FROM INS1 WITHOUT BEING USED OR MODIFIED IN BETWEEN, AND WHOSE RC-STRING IS COMPATIBLE WITH [FLOW], WITH A PROPERLY MODIFIED RC-STRING.

TO DERIVE A MORE EFFICIENT NON-RECURSIVE ALGORITHM FROM THE RECURSIVE RELATIONSHIP SHOWN ABOVE, WE USE THE USUAL WORKPILE METHOD. DURING THE COMPUTATION OF REACH, THE RC-STRINGS ARE REPRESENTED AS A TUPLE OF FLOW TRIPLES, TO MAKE IT EASY TO PROCESS THEM. LATER ON WE CALL A COMPRESSION ROUTINE TO TRIM THEM DOWN TO A MANAGEABLE SIZE.

IT MAY BE CONVENIENT TO MAKE USE OF A SIMPLE METHOD TO COMPRESS THE RC-STRINGS. THIS CAN BE DONE AS FOLLOWS.
EVERY FLOW TRIPLET [INS,Q,DIR] IS HASHED TO A K - BIT CODE, WHERE K IS RATHER SMALL (≤ 6, SAY). WE ALSO REQUIRE THAT THE HASH CODE OF [INS,Q,CALL] AND [INS,Q,RET] WILL BE EACH THE INVERSE OF THE OTHER, RELATIVE TO A CERTAIN BINARY OPERATION, DENOTED BY +, DEFINED ON THE SET ≤ 0,1 ... 2**K - 1 ≥. (A LIKELY CHOICE FOR THAT OPERATION IS EITHER AN EXCLUSIVE OR, OR ADDITION MODULO 2**K.) THEN, EACH RC-STRING STR WILL BE HASHED TO THE CODE [+: J := 1 ... +STR] CODE(STR(J))

THIS HASHING ALLOWS US TO PRE-CALCULATE ALL THE STANDARD OPERATIONS ON RC-STRINGS, AND STORE THEM IN SEVERAL TABLES, SO THAT DURING ANY FURTHER ALGORITHM ALL RC-STRING OPERATIONS, AS, FOR EXAMPLE, CHECKING FOR COMPATIBILITY, PROPER CONCATENATION, DETERMINATION OF INVERSES, MAXIMUM, MINIMUM ETC. (CF. NL 187) CAN BE PERFORMED AS TABLE LOOK-UPS. OBVIOUSLY, SOME INFORMATION WILL BE LOST BY SUCH A HASHING, BUT IF THE CALL GRAPH OF THE INPUT PROGRAM IS NOT VERY COMPLICATED, WE CAN EXPECT TO RETAIN RATHER ACCURATE TRACE BACK INFORMATION.

HERE IS A SETL VERSION OF THE INTERPROCEDURAL DATA FLOW ALGORITHM THAT WE HAVE JUST DESCRIBED:

```
   CONST CALL,RET;  $ FLOW DIRECTION MNEMONICS

   DEFINE COMPREACH;

$ STEP 1.     I N I T I A L I Z A T I O N

   EDGES¬FROM := NL;

   (∀INS → CALLS)
$ CALLS IS THE SET OF ALL PROCEDURE CALLS
        (∀Q → CALLEDGE≤INS≥)
            INS1 := ENTRY(Q);
$ ENTRY(Q) IS THE ENTRY TO PROCEDURE Q
            FLOW := [INS,Q,CALL];
$ NOTE THAT FLOW CONTAINS ALSO THE PROCEDURE CALLED
$ OR RETURNED FROM, SINCE IN CASE OF PROCEDURE VARIABLES
$ THE CALLING INSTRUCTION ITSELF IS NOT SUFFICIENT TO
$ IDENTIFY THE CALLED PROCEDURE
            EDGES¬FROM WITH [INS,[INS1,FLOW]];
        END ∀Q;
   END ∀INS;

   (∀INS → EXITS)
        Q := PROC¬OF(INS);
$ PROC¬OF(INS) IS THE PROCEDURE CONTAINING INS
        (∀INS1 → RETURNEDGE≤Q≥)
            FLOW := [INS1,Q,RET];
            EDGES¬FROM WITH [INS,[INS1,FLOW]];
        END ∀INS1;
   END ∀INS;

   WORKPILE := NL;

$ IN GENERAL, WORKPILE CONTAINS NEW FRAGMENTS OF REACH
$ THAT WERE NOT YET USED TO UPDATE THE REACH AT OTHER INSTRUCTIONS.

   (∀INS2 → EXITS¬OR¬CALLS)
            AUXREACH1 := REACHIN ≤INS2≥;

        (∀[INS3,FLOW] → EDGES¬FROM ≤INS2≥)
            WORKPILE≤INS3≥ := ≤ [OI,[FLOW]] : OI → AUXREACH1 ≥;
$ THE INITIAL ELEMENTS OF WORKPILE (AND REACH) ARE THE OCCURENCES
$ IN EACH  REACHIN≤INS2≥, WITH A SINGLETON RC¬STRING [FLOW],
$ INDICATING THE IMMEDIATELY FOLLOWING PROCEDURE BOUNDARY CROSSING
$ FROM INS2 TO INS3. THESE MODIFIED OCCURENCES ARE NOW AVAILABLE
$ AT INS3, WHICH IS IN ENTRIES¬OR¬RETURNS.
        END ∀;
   END ∀INS2;

   REACH := WORKPILE;
```

```
$ STEP 2, P R O P A G A T I O N   U S I N G   W O R K P I L E

     (WHILE WORKPILE NE NL)
          [INS1,SOME] := ARB WORKPILE;
          REACH-FRAG := WORKPILE ≤INS1≥;
          WORKPILE LESSF INS1;
$ EXTRACT AN INSTRUCTION INS1 WITH ALL ITS NEW REACH ELEMENTS
$ FROM WORKPILE.

          (∀INS2 ∈ PCFP ≤INS1≥)

               (∀[INS3,FLOW] ∈ EDGES-FROM ≤INS2≥)
                    AUXREACH := NL;
$ AUXREACH IS THE NEW REACH FRAGMENT BEING CONSTRUCTED AT INS3.
$ INS1 - INS2 - INS3  IS THE FLOW FROM THE LAST UPDATED INSTRUCTION
$ INS1 TO THE NEW ONE INS3 ( BOTH IN ENTRIES-OR-RETURNS ) VIA INS2
$ ( IN EXITS-OR-CALLS ). [INS1,INS2] ∈ PCFP (PROCEDURE CONTROL FLOW
$ PATHS), AND INS2 LEADS TO INS3 BY A DIRECT PROCEDURE BOUNDARY
$ CROSSING (SEE THE INFORMAL EXPLANATION ABOVE).

                    (∀[OI,STR] ∈ REACH-FRAG
                              | VARID(OI) NOTIN NOREACH≤INS1,INS2≥)
$ TAKE THE OCCURENCES IN THE OLD REACH FRAGMENT WHICH MAY
$ REACH INS3
                         STR1 := PROP-CONCAT(STR,FLOW);
$ IF COMPATIBLE, CONCATENATE [FLOW] TO THE RIGHT OF THE
$ PREVIOUS RC-STRING. OTHERWISE RETURN OM.
                         IF STR1 NE OM THEN
                              AUXREACH WITH [OI,STR1]; END IF;
                    END ∀;

                    WORKPILE≤INS3≥ + AUXREACH - REACH≤INS3≥;
$ ADD TO THE WORKPILE ONLY NEW OCCURENCES
                    REACH≤INS3≥ + AUXREACH;
$ AUGMENT WORKPILE ( = NEW REACHES) AND REACH ITSELF AT INS3
               END ∀;
          END ∀INS2;
     END WHILE;


     RETURN;
     END COMPREACH;
```

```
   DEFINEF PROP¬CONCAT(STR,FLOW);

$ THIS IS THE PROPER CONCATENATION ROUTINE FOR RC¬STRINGS
$ IT RETURNS OM IF THE CONCATENATION IS IMPROPER

$ STR  IS A RC¬STRING. THAT IS, IT IS A TUPLE OF FLOW TRIPLES,
$ WHERE ALL THE RETURN COMPONENTS PRECEED ALL THE CALL
$ COMPONENTS.  FOR EACH RC¬STRING THERE CORRESPONDS
$ A FLOW PATH IN THE INPUT PROGRAM, SUCH THAT THE COMPONENTS OF THE
$ STRING REPRESENT, IN LEFT TO RIGHT ORDER, ALL PROCEDURE CONTROL
$ TRANSFERS ALONG THIS PATH, OMITTING COMPLETE CALLS. WE ALSO REQUIRE
$ THAT A RC¬STRING SHOULD NOT CONTAIN THE SAME TRANSFER TWICE.
$ FLOW  IS A NEW FLOW COMPONENT TO BE ADDED TO THE RIGHT HAND SIDE
$ OF STR.
$ THIS CONCATENATION IS CALLED PROPER IF THE NEW STRING SATISFIES,
$ OR CAN BE TRANSFORMED INTO A STRING WHICH SATISFIES, ALL THE
$ CONSTRAINTS IMPOSED ON RC¬STRINGS.


   IF NOT (∃J := 1 ... ↓STR ↑ STR(J)(3) NE RET) THEN
        J := ↓STR + 1;
   END IF;
$ J IS THE FIRST PLACE IN STR OF A CALL COMPONENT

   TRANSF := FLOW(1:2);
   DIR := FLOW(3);
$ SEE EDGES¬FROM CONSTRUCTION FOR THE STRUCTURE OF FLOW

   BEGIN
        IF DIR EQ CALL THEN
             IF (∃I := J ... ↓STR ↑ STR(I)(1:2) EQ TRANSF) THEN
                  RETURN OM;
                  ELSE RETURN STR ↑↑ [FLOW];
             END IF;
$ IF A CALL, CHECK IF THERE IS ALREADY SUCH A CALL
$ IN STR, IF SO, THEN THE APPENDING IS IMPROPER. OTHERWISE
$ RETURN THE CONCATENATED STRING.

             ELSEIF (∃I := 1 ... J-1 ↑ STR(I)(1:2) EQ TRANSF) THEN
                  RETURN OM;
$ IF A RETURN, AND ALREADY APPEARS AS A RETURN IN STR THEN IMPROPER
                  ELSEIF (∃I := J ... ↓STR ↑ STR(I)(1:2) EQ TRANSF)
$ IF IT APPEARS ALREADY AS A CALL
                       THEN RETURN STR(1:I-1);
$ SUPPRESS THE END OF THE STRING FROM THE CALL TO THE RETURN
$ SINCE THIS END IS EMBEDDED IN THE COMPLETE CALL JUST DETECTED
                       ELSEIF J > ↓STR THEN
$ IF IT DOES NOT APPEAR IN THE STRING, AND THERE ARE ONLY
$ RETURNS IN THE STRING, THEN
```

```
                              RETURN STR ++ [FLOW];
$ RETURN THE CONCATENATED STRING,
                              ELSE RETURN OM;
$ OTHERWISE, IMPROPER CONCATENATION.
        END IF;
    END;

    END PROP¬CONCAT;
```