KfK 4979
Februar 1992

ARBUS: A FORTRAN Tool for Generating Program Tree Structure Diagrams

C. Ferrero, M. Zanger
Institut für Neutronenphysik und Reaktortechnik
Projekt Heißdampfreaktor-Sicherheitsprogramm

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE
Institut für Neutronenphysik und Reaktortechnik
Projekt Heißdampfreaktor-Sicherheitsprogramm

KfK 4979

ARBUS: A FORTRAN Tool for Generating Program Tree Structure Diagrams

C. Ferrero *) M. Zanger

*) Hauptabteilung Ingenieurtechnik, presently delegated to Institut für Neutronenphysik und Reaktortechnik.

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

ABSTRACT

The FORTRAN77 stand-alone code ARBUS has been designed to aid the user by providing a tree structure diagram generating utility for computer programs written in FORTRAN language. This report is intended to describe the main purpose and features of ARBUS and to highlight some additional applications of the code by means of practical test cases.

ARBUS: Ein Werkzeug zur Generierung des zu einem FORTRAN-Rechenprogramm gehörigen Aufrufbaumdiagrammes

ZUSAMMENFASSUNG

Das FORTRAN77-Rechenprogramm ARBUS wurde entwickelt, um dem Benutzer die graphische Darstellung des Aufrufbaumdiagramms bzw. der Aufrufstruktur der einzelnen Unterprogramme in einem beliebigen FORTRAN-Programm zu ermöglichen. In diesem Bericht wird auf die Zielsetzung und die Hauptmerkmale von ARBUS eingegangen. Außerdem werden einige Anwendungen des Codes anhand von praktischen Beispielen erläutert

TABLE OF CONTENTS

1. PURPOSE

ARBUS is a labor-saving software tool to be used by programmers and analysts to verify and document complex computer programs in the frame of the software development cycle. It provides automatic technical documentation through its ability to generate hierarchical structures of any FORTRAN program unit available in form of a sequential file. This is especially valuable in the maintenance of software, where programs are often changed by someone not acquainted with the source code.

The user can specify by input the name of the program unit to be used as tree structure starting point and the branching depth related to it. The first of these two options is available also in /1/ and in /2/, the second only in /1/. However the utilities described there can only print a tree structure on an output listing or data set, thus limiting considerably not only the readability of the issued description but also the max. printable width of the calling structure.

These drawbacks are partly removed in ARBUS since this program generates a plot of the tree structure with the black-and-white VERSATEC plotting device of Kernforschungszentrum /3/, what enables the user to get hierarchical code structures up to the 20th calling level. Moreover ARBUS, - unlike the programs in /1/ and /2/ -, was expressly designed for the special task to produce compact and clearly arranged graphical analysis reports and the authors endeavoured to make it so flexible and user-friendly as possible.

ARBUS development benefitted from the experience gathered with a previous tool /4/ whose major shortcoming was to ignore the presence of FUNCTION subprograms in the input source code, thus preventing from a complete and correct analysis of it.

The tree structure diagram gives a quick graphical representation of the complexity of the source code calling hierarchy. It is also handy for establishing overlay and segmentation schemes, to remove redundancies and for restructuring the calling arrangement.

ARBUS processes programs written in standard FORTRAN 66 and FORTRAN 77 /5/, provided these have been already compiled successfully. Non-compilable sources may cause ARBUS execution to abend.

2. FEATURES OF ARBUS

ARBUS reads a sequential file containing the FORTRAN source code with any number of modules (henceforth also called procedures for the sake of analogy with structured programming languages like PASCAL or C). While reading it processes the source code in accordance with a simple set of input instructions specified by the user in the related Job Control Language (JCL) as described in 2.1 . Up to three analysis reports may be issued by running ARBUS:

two printable invocation reports, one showing the selected calling tree (calling tree report or invocation bands report) and the other listing in the first section all the modules called by each individual program unit and in complementary manner in the second section also every module by which each individual program unit is called (invocation summary or simply protocol report).

The invocation summary report can be used to obtain a complete listing of the calling structure of a given program. It highlights modules that are not invoked and modules that do not call other modules. Procedures that are referenced but not defined in the source code are identified. This holds for SUBROUTINE but not for FUNCTION units because for the latter to be detected a considerable amount of tedious programming work would be additionally needed. The inclusion of such a feature in ARBUS would moreover reduce the time effectiveness in running the program, thus it was decided to ignore invocations of FUNCTIONs which do not exist in the source list.

The third output file consists of a brief statistics report where general informative data about the source code under investigation are summarized (e.g. number of subroutines, number of functions, max. call depth etc.). A detailed description of this report is given later in section 2.2 .

Last but not least ARBUS can provide a calling tree plot which reflects essentially the structure of the calling tree report but makes use of the VERSATEC graphical package in order to obtain compact and elegant presentation effects (see Figs. 1-5).

2.1 CODE INPUT DESCRIPTION

An executable version of ARBUS has been created and can be run by the following JCL:

```
//AIT496Z JOB (0496,935,P0000),FERRERO,NOTIFY=AIT496,MSGCLASS=H,
// REGION=4000K
//BLOCK EXEC F7CLG,PARM.C='LANGLVL(77),NOFIPS'
//*************************************************
//*
//C.SYSIN DD DSN=AIT496.BAUM.FORT(ARBUS),DISP=SHR
//*
//*************************************************
//G.SYSIN DD *
3
AIT496.TESTBSP1.FORT
MAIN
20
AIT496.CALLTAB.DATA
AIT496.TREE.DATA
AIT496.STATIS.DATA
//*************************************************************
//***                                                       ***
//***  PLOT - PARAMETER:                                    ***
//***  -----------------                                    ***
//***                                                       ***
//***  XMAX ..... MAXIMAL ERLAUBTE PLOTLAENGE IN INCH       ***
//***                                                       ***
//*************************************************************
//G.PLOTPARM DD *
 &PLOT XMAX=600.0 SPACE=30 &END
//PLOT EXEC SVPLOT
```

Let us consider the G.SYSIN instructions block. In the first line an integer number must be entered ranging from 0 to 3. These numerical values correspond to the following menu options:

- 0 - only the program statistics file will be created
- 1 - the program statistics and the tree diagram file will be created
- 2 - the program statistics file and the tree diagram VERSATEC plot will be created
- 3 - the program statistics file, the tree diagram file and additionally the tree diagram VERSATEC plot will be created

In the second line the name of the data set containing the program to be analysed must be entered (in our example: AIT496.TESTBSP1.FORT) The data set organization must be

sequential. For an input source code that contains different procedures with the same name, the first one encountered while processing is printed/plotted; however all of them are analysed and printed in the invocation summary file together with their subordinate modules (which may be different). Both unnamed and named BLOCK DATA subprograms are as well listed in the invocation summary file.

The item "MAIN" of the third line in the list specifies the name of the program unit to be used as the starting point of the tree structure. If no PROGRAM statement is used as the entry source statement of the main program, then the default name MAIN is assigned, this name appearing throughout the two ARBUS printable reports whenever the name of the module is used. The main program is supposed to be unique. If that is not the case, then the execution is stopped while processing the first executable statement of the next main program and the message " YOU HAVE MORE THAN ONE MAIN PROGRAM " is printed out on the standard output unit FT06F001.

If the user wants to have the invocation tree starting from the main program and the name he declared in the JCL doesn't correspond to the actual one in the code to be analysed, ARBUS runs normally but uses the correct name in the printable output data sets and in the VERSATEC tree plot. This implies that when the misspelled name of a SUBROUTINE or of a FUNCTION is coded in the input,-i. e. a non existing procedure is declared-, then the call tree start point assumed in ARBUS is the main program, although this might have not been the true intention of the user.

In the fourth line the selected maximum branching depth of the invocation tree must appear. If the entry value is greater than 20 then ARBUS sets it automatically to 20, which corresponds to the max. plottable depth compatible with the VERSATEC paper sheet width (for comparison: the corresponding max. depth on printer is 7). If the expected depth is lower than the actual one or the physical mapping of the tree is such that the max. width of the paper/plot output is exceeded, a tree diagram truncated at the level selected by the user or respectively imposed by the paper margin will be generated. A dash following a framed procedure name indicates that the tree is to be continued at that point.

The last three lines of the G.SYSIN list must contain the names of the report files to be issued by ARBUS. In order of appearance in the list we have:

- the invocation summary report (in our example: AIT496.CALLTAB.DATA)
- the invocation tree report (in our example: AIT496.TREE.DATA)
- the program statistics report (in our example: AIT496.STATIS.DATA)

The user is supposed to allocate these three data sets with the (recommended) DCB parameters RECFM=FB, LRECL=132, BLKSIZE = 3960 and with a cautiously high number of secondary disk tracks for "CALLTAB" and "TREE" prior to running ARBUS. No assessment can be made here about the total amount of allocation space required by them except that it should be somewhat proportional to the expected complexity of the analysed program. This is of no concern for the statistics file which has a fixed (=36) number of output lines, no matter what the input source code is.

It should be noticed that regardless of the menu selection number the invocation summary file and the statistics file are always generated whereas the invocation tree file is produced only with the option numbers 2 and 3. As a consequence, when 0 or 1 are instead selected only two data set names must be entered in the G.SYSIN list and the G.PLOTPARM cards may be omitted.

2.2 CODE OUTPUT DESCRIPTION

When the user is not yet acquainted with the source code he wants to analyse it is suggested to choose the option "0" for issuing only the statistics file. The information contained there enables him to make the right decision about the way he wants to proceed further in his analysis. In fact ARBUS calculates the sizes (length and width) in meters and inches of the tree diagram plot and the length (in number of lines) of the call tree report. These quantities are printed in the statistics file.

This feature is of paramount importance for the user when he is not able to guess in advance how long the tree he wants to plot/print is (for some source codes a total length of even more than 100 m may result!). Therefore before making use of the plotting/printing option it is strongly recommended to make a pre-run requesting only the statistics output. In case the sizes foreseen by ARBUS are too large for the physical/plotting systems, the user is suggested to lower the value of the calling depth and to start again ARBUS until acceptable values are attained. He can then easily reconstruct the whole program tree by running repeatedly ARBUS taking lower-level procedures as starting points for the remaining branchings to be visualized.

In the statistics file the user can find also important information items about the analysed source code like the number of SUBROUTINEs, FUNCTIONs, BLOCK DATA, ENTRY points and EXTERNALs included in it. Also the number of subroutines invoked but not available in the source code is written. Additionally all source lines are scanned and sorted in actual program statements with (possibly) continuation lines and comment lines. The amounts of each kind of line are issued in the statistics file along with their sum, which must be equal to the total amount of source program lines.

The invocation bands file (in the previous section called AIT496.CALLTAB.DATA) shows the module calling hierarchies identified by ARBUS by means of a graphical tree structure representation. The user should be aware of the fact that if a procedure is called more than once within a given program unit, then for sake of compact visualization only the first call occurring in the FORTRAN source is mapped onto the tree diagram print.

The invocation summary file contains a table listing the CALLEEs (lower-level routines called by the program unit) and CALLERs (upper-level routines calling the program unit) for each program unit. The symbol "-----" indicates that there are no CALLERs or CALLEEs. These are typically the cases of the main program that is expected not to be called by any other module and of BLOCK DATA modules which do not call other modules. Any other

modules which are shown as not being invoked should be audited for possible inconsistencies with the rest of the analysed library.

It should be noted that in this file neither the statements where the module reference occurs nor the number of times one module is called within the whole program unit under investigation are listed. Such kind of documentation was considered to be beyond the scope of ARBUS reporting activities and therefore intentionally ruled out of the work schedule of the authors.

The invocation tree VERSATEC plot exhibits the same structure of the invocation bands file but provides a more compact and elegant graphical representation of it. It is recommended to make use of this output option (=2) in order to save paper costs.

Diagnostic messages resulting from the ARBUS analysis are written on the FT06F001 output listing. A preliminary input parameters check is performed by the ARBUS subroutine FEHLER. This subroutine surveys also the array subscripts history in ARBUS while processing the input source code which might exceed the max. permissible parameter values declared in ARBUS (s. Table 1). Once the error is detected a warning is printed which identifies the error cause and explains the problem recovery action to be undertaken (normally resetting a higher value for the upper subscript of one vector/matrix field in ARBUS main program).

2.3 EXAMPLES

In this section two sample problems are given which highlight the ARBUS abilities described in the previous sections. The following examples contain the JCL's needed to run ARBUS, the source lists of the input programs and the related output data sets. It was believed that although the sample sources TESTBSP1.FORT and TESTBSP2.FORT are quite short, they are representative for testing the functionality of ARBUS, because they contain all the features which ARBUS is supposed to cover and document. In the first example the whole program tree diagram of TESTBSP1.FORT has been generated and printed out in the protocol data set. The second example differs from the first in that several ARBUS runs were made and every time only a part of the call tree diagram of TESTBSP2.FORT was printed/plotted (s. Figs. 1-4) with different start points.

Example No. 1

```
//AIT496Z JOB (0496,935,P0000),FERRERO,NOTIFY=AIT496,MSGCLASS=H,
//    REGION=4000K,TIME=(1,0)
// EXEC F7CLG,PARM.C='LANGLVL(77),ASTER,SOURCE,NOMAP,NOLIST'
//*
//C.SYSIN DD DSN=AIT496.BAUM.FORT(ARBUS),DISP=SHR
//*
//G.SYSIN DD *
1
AIT496.TESTBSP1.FORT
MAIN
20
AIT496.CALLTAB.DATA
AIT496.TREE.DATA
AIT496.STATIS.DATA
```

```
**************************************************************
```

```
C=============================================================C
C                                                             C
C           T E S T B E I S P I E L  1                        C
C                                                             C
C    DIESES PROGRAMM DIENT LEDIGLICH DEM ZWECK, DIE           C
C    FUNKTIONALITAET DES PROGRAMMES  A R B U S  ZU ZEIGEN !!! C
C                                                             C
C=============================================================C
      BLOCKDATA
      INTEGER IFELD(5)
      COMMON IFELD
      END
C-------------------------------------------------------------
      BLOCK DATA TEST
```

```
      INTEGER NFELD(5)
      COMMON / FELD / NFELD
      DATA NFELD / 5,4,3,2,1 /
      E N D
C-----------------------------------------------------------------
      EXTERNAL QUAD, SUM, SIN,    ! DIES IST EIN KOMMENTAR
     &         COS                ! SOLCHE KOMMENTARE ERLAUBT DAS
      INTEGER IZAHL(5)            ! PROGRAMM ARBUS !!!
      EXTERNAL TAN, WURZEL, SINHYP
      WRITE (*,*) 'GEBEN SIE EINE REAL-ZAHL EIN :'
      READ  (*,*) DZAHL
      WRITE (*,*) 'BITTE WAEHLEN SIE :'
      WRITE (*,*) '1 - SIN, 2 - COS, 3 - TAN DER ZAHL !'
      READ  (*,*) IWAHL
      IF (IWAHL .EQ. 1) CALL SUB3 (SIN, DZAHL, DERG)
      IF (IWAHL .EQ. 2) THEN
         CALL SUB3 (COS, DZAHL, DERG)
      ELSE IF (IWAHL .EQ. 3) THEN
         CALL SUB3 (TAN, DZAHL, DERG)
      END IF
      WRITE (*,*) 'DAS ERGEBNIS LAUTET:', DERG
      CALL SUB0 (IWAHL)
      IF (REAL(IWAHL).GT.DWURZ(DERG)) CALL SUB1(QUAD,SUM,IZAHL(1),
     &QUAD,SUM,IZAHL)
      CALL SUB4('CALL SUB0(MINI,MAXI)', 'MINI')
      CALL SUBNOT
      CALL CALL
      CALL ENTRY
C
      WRITE (*,*) 'GEBEN SIE FUENF INTEGER-ZAHLEN EIN:'
      READ  (*,*) (IZAHL(IZ), IZ = 1, 5)
      WRITE (*,*) 'IWAHL EINGEBEN :'
      READ  (*,*) IWAHL
      IF (IWAHL .EQ. 1) THEN
         CALL SUB2E1 (IZAHL, IERG, QUAD)
      ELSE IF (IWAHL .EQ. 2) THEN
         CALL SUB2E1 (IZAHL, IERG, SUM)
      ELSE
         CALL SUB2E2 (IZAHL, IERG, SUM)
      END IF
      WRITE (*,*) 'DAS ERGEBNIS LAUTET:', IERG
      WRITE (*,*) AWURZ(2.), ' IST DIE ACHTE WURZEL VON 2 !'
      WRITE (*,*) VWURZ(FWURZ(2.)), ' IST DIE 20. WURZEL VON 2 !'
      END
C-----------------------------------------------------------------
      SUBROUTINE SUB0 (IWAHL)
      EXTERNAL ENORM
      EXTERNAL UNORM
      REAL ENORM1(2)
      WRITE (*,*) 'SUBROUTINE SUB0: ZWEI REAL-ZAHLEN EINGEBEN:'
      READ (*,*) ENORM1(1), ENORM1(2)
      IF (MINI(IWAHL,2).GT. 2)
     &ENORM0 = QUADNO (ENORM1(1),ENORM1(2),ENORM)
      IF (ENORM0 .GT. DWURZ(3.0))
```

```
      &ENORM0 = KUBNO (ENORM1(1),ENORM1(2),
      &UNORM)
       WRITE (*,*) 'ERGEBNIS VON SUB0: ENORM0 = ', ENORM0
       END
C-----------------------------------------------------------------
       SUBROUTINE SUB1 (SUBEX1,SUBEX2,IZ,SUBEX3,SUBEX4,IZAHL)
       INTEGER IZAHL(5)
       WRITE (*,*) 'EXTERNALS IN SUB1: SUBEX1,SUBEX2'
       CALL SUBEX3(IZAHL,IERG1)
       IERG2 = 5
       IF (IERG1.GT.10) CALL SUBEX4(IZAHL,IERG2)
       IERG1 = IERG1 * IZ
       WRITE (*,*) 'ERGEBNIS VON SUB1: IERG1 = ',MINI(IERG1,IERG2)
       WRITE (*,*) 'LOGER() WIRD HIER NICHT AUFGERUFEN'
       END
C-----------------------------------------------------------------
       BLOCKDATA TEST2
       INTEGER LFELD(5)
       COMMON /FELD2/ LFELD
       DATA LFELD / 3,3,3,3,3 /
       END
C-----------------------------------------------------------------
       SUBROUTINE SUB2 (IZAHL, IERG, SUBEXT)
       INTEGER IZAHL(5)
       ENTRY SUB2E1 (IZAHL, IERG, SUBEXT)
       WRITE (*,*) 'ERSTER ENTRY IN SUB2 !!'
       CALL SUBEXT (IZAHL, IERG)
       ENTRY SUB2E2 (IZAHL, IERG, SUBEXT)
       WRITE (*,*) 'ZWEITER ENTRY IN SUB2 !!'
       END
C-----------------------------------------------------------------
       SUBROUTINE SUB3 (FUNC, DZAHL, DERG)
       WRITE (*,*) 'SUBROUTINE SUB3 !!!!'
       DERG = FUNC(DZAHL)
       END
C-----------------------------------------------------------------
       SUBROUTINE SUB4 (CHELP1, CHELP2)
       PARAMETER (MAXP=10)
       INTEGER IPOS(MAXP)
       CHARACTER*(*) CHELP1, CHELP2
       CALL INDEX2(CHELP1,CHELP2,IANZ,IPOS,MAXP)
       WRITE (*,*) 'CHELP2 WURDE IN CHELP1 ', IANZ, 'GEFUNDEN !'
       END
C-----------------------------------------------------------------
       SUBROUTINE QUAD (IZAHL, IERG)
       INTEGER IZAHL(5)
       IERG = IZAHL(1) * IZAHL(1) + IZAHL(2) * IZAHL(2)+IZAHL(3)
      &       *IZAHL(3) + IZAHL(4) * IZAHL(4) + IZAHL(5)*IZAHL(5)
       END
C-----------------------------------------------------------------
       SUBROUTINE DUMMY (X, Y)
       IERG = INT (DWURZ (X) * VWURZ (X)) + MINI (INT(X),INT(Y))
       END
C-----------------------------------------------------------------
```

```
      SUBROUTINE SUM (IZAHL, IERG)
      INTEGER IZAHL(5)
      IERG = IZAHL(1) + IZAHL(2) + IZAHL(3) + IZAHL(4)+IZAHL(5)
      END
C-----------------------------------------------------------------
      FUNCTION MINI (I1,I2)
C
      IF (I1.LT.I2) MINI = I1
      IF (I2.LT.I1) MINI = I2
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION ENORM (IZAHL)
      INTEGER IZAHL(5)
C
      CALL SUM (IZAHL, IERG)
      ENORM = DWURZ(REAL(IERG))
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION QUADNO (X,Y,EFUNCQ)
C
      QUADNO = EFUNCQ(X,Y)*EFUNC(X,Y)
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION KUBNO (X,Y,EFUNCK)
C
      KUBNO = EFUNCK(X,Y)*EFUNCK(X,Y)*EFUNCK(X,Y)
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION DWURZ (X)
C
      DWURZ = X ** (1./3.)
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION VWURZ (X)
C
      VWURZ = SQRT(SQRT(X))
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION FWURZ (X)
C
      FWURZ = X ** (1./5.)
C
      END
C-----------------------------------------------------------------
      REAL FUNCTION AWURZ (X)
C
      AWURZ = SQRT(VWURZ(X))
C
```

```
      END
C-----------------------------------------------------------------
      LOGICAL FUNCTION LOGER (I)
C
      LOGER = MOD(I,2) .EQ. 0
C
      END
C-----------------------------------------------------------------
      SUBROUTINE INDEX2 (CSTR1, CSTR2, IANZ, IPOS, MAXANZ)
C
C=================================================================C
C                                                                 C
C            S U B R O U T I N E   I N D E X 2                    C
C                                                                 C
C=================================================================C
C                                                                 C
C     AUTOR ................ MICHAEL ZANGER / PHDR / KFK         C
C     ERSTELLUNGSDATUM ..... 05-12/1990                           C
C     LETZTE AENDERUNG ..... 21.05.1990                           C
C     PROGRAMMIERSPRACHE ... FORTRAN 77                           C
C     COMPILER-OPTIONEN .... OPTIMIZE                             C
C     RECHENANLAGE ......... VAX / IBM                            C
C     SACHGEBIET ........... FORTRAN-TOOLS                        C
C                                                                 C
C=================================================================C
C                                                                 C
C                     BESCHREIBUNG DER SUBROUTINE:                C
C                     ============================                C
C                                                                 C
C     ERWEITERUNG DER INTRINSIC-FUNCTION INDEX(). SAEMTLICHE      C
C     VORKOMMEN DER ZEICHENKETTE <CSTR2> IN <CSTR1> WERDE         C
C     GESUCHT.                                                    C
C                                                                 C
C=================================================================C
C                                                                 C
C                          UEBERGABE-ARGUMENTE:                   C
C                          ====================                   C
C                                                                 C
C CSTR1 .... ZU DURCHSUCHENDER CHARACTER-STRING                   C
C CSTR2 .... GESUCHTER CHARACTER-STRING                           C
C IANZ ..... HAEUFIGKEIT DES VORKOMMENS VON <CSTR2> IN <CSTR2>    C
C IPOS() ... POSITIONEN DES VORKOMMENS VON <CSTR1> IN <CSTR2>     C
C MAXANZ ... MAXIMAL ERLAUBTE ANZAHL VON POSITIONEN               C
C                                                                 C
C=================================================================C
C
      CHARACTER*(*)  CSTR1, CSTR2
C
      INTEGER  IPOS(MAXANZ)
C
      IPOSH1 = 1
      DO 1000 IANZ = 1, MAXANZ
         IPOSH2 = INDEX(CSTR1(IPOSH1: ), CSTR2)
         IF (IPOSH2 .EQ. 0) GOTO 999
```

```
         IPOSH1 = IPOSH1 + IPOSH2
         IPOS(IANZ) = IPOSH1 - 1
 1000 CONTINUE
C
  111 IPOSH2 = INDEX(CSTR1(IPOSH1: ), CSTR2)
      IANZ = IANZ + 1
      IF (IPOSH2 .GT. 0) THEN
         IPOSH1 = IPOSH1 + IPOSH2
         GOTO 111
      END IF
C
  999 IANZ = IANZ - 1
C
      E N D
```

****

PROTOCOL DATASET FROM AIT496.TESTBSP1.FORT
==

This data set includes also an EXTERNAL cross reference table which displays all of the references to the externals in the program (including SUBROUTINEs, ENTRY points and FUNCTIONs) and the modules in which they are referenced. From a first inspection of the table it is evident that no attempt was made in ARBUS to track the EXTERNAL reference history beyond the first calling level since its declaration. EXTERNALs declared but not used within the program are pointed out by the symbol '----' in the third and fourth column of the table, corresponding to the lower hierarchical module level.

At the end of the report is a list of external subprograms (typically operating system service and/or environment library subroutines) that are referenced but not defined in the source code. Additionally the SUBROUTINEs and FUNCTIONs which are present but not referenced within the source code are also listed.

```
*****************************************
***   LIST OF CALLS FOR EVERY MODULE   ***
*****************************************
```

THE MAIN-PROGRAM MAIN CALLS:
============================

SUB3, SUB0, SUB1, DWURZ, SUB4, SUBNOT, CALL, ENTRY, SUB2E1, SUB2E2, AWURZ, VWURZ, FWURZ

THE SUBROUTINE SUB0 CALLS:
==========================

MINI, QUADNO, KUBNO, DWURZ

THE SUBROUTINE SUB1 CALLS:
==========================

SUBEX3, SUBEX4, MINI

THE SUBROUTINE SUB2 CALLS:
==========================

THE SUBROUTINE ENTRY SUB2E1 CALLS:
==================================

SUBEXT

THE SUBROUTINE ENTRY SUB2E2 CALLS:
==========

THE SUBROUTINE SUB3 CALLS:
==========

FUNC

THE SUBROUTINE SUB4 CALLS:
==========

INDEX2

THE SUBROUTINE QUAD CALLS:
==========

THE SUBROUTINE DUMMY CALLS:
==========

MINI, DWURZ, VWURZ

THE SUBROUTINE SUM CALLS:
==========

THE SUBROUTINE INDEX2 CALLS:
==========

THE SUBROUTINE SUBNOT CALLS:
==========

THE SUBROUTINE CALL CALLS:
==========

THE SUBROUTINE ENTRY CALLS:
==========

THE FUNCTION MINI CALLS:
==========

THE FUNCTION ENORM CALLS:
==========

SUM, DWURZ

THE FUNCTION QUADNO CALLS:
==========

EFUNCQ

THE FUNCTION KUBNO CALLS:
==========

EFUNCK

THE FUNCTION DWURZ CALLS:
==========

THE FUNCTION VWURZ CALLS:
==========

THE FUNCTION FWURZ CALLS:
==========

THE FUNCTION AWURZ CALLS:
==========

VWURZ

THE FUNCTION LOGER CALLS:
==========

****
*** LIST OF CALLING MODULES FOR EVERY MODULE ***
****

THE SUBROUTINE SUB0 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE SUB1 IS CALLED FROM:

==========

MAIN

THE SUBROUTINE SUB2 IS CALLED FROM:
==========

THE SUBROUTINE ENTRY SUB2E1 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE ENTRY SUB2E2 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE SUB3 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE SUB4 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE QUAD IS CALLED FROM:
==========

THE SUBROUTINE DUMMY IS CALLED FROM:
==========

THE SUBROUTINE SUM IS CALLED FROM:
==========

ENORM

THE SUBROUTINE INDEX2 IS CALLED FROM:
==========

SUB4

THE SUBROUTINE SUBNOT IS CALLED FROM:
==========

MAIN

THE SUBROUTINE CALL IS CALLED FROM:
==========

MAIN

THE SUBROUTINE ENTRY IS CALLED FROM:
==========

MAIN

THE FUNCTION MINI IS CALLED FROM:
==========

SUB0, SUB1, DUMMY

THE FUNCTION ENORM IS CALLED FROM:
==========

THE FUNCTION QUADNO IS CALLED FROM:
==========

SUB0

THE FUNCTION KUBNO IS CALLED FROM:
==========

SUB0

THE FUNCTION DWURZ IS CALLED FROM:
==========

MAIN, SUB0, DUMMY, ENORM

THE FUNCTION VWURZ IS CALLED FROM:
==========

MAIN, DUMMY, AWURZ

THE FUNCTION FWURZ IS CALLED FROM:
==========

MAIN

THE FUNCTION AWURZ IS CALLED FROM:
==========

MAIN

THE FUNCTION LOGER IS CALLED FROM:
==========

```
**********************************
***    BLOCK-DATA-SUBROUTINES    ***
**********************************

-NONAME-, TEST, TEST2

*******************************
***    ENTRYS IN ROUTINES    ***
*******************************

ENTRYS OF SUB2:
===============

SUB2E1, SUB2E2

************************************************
***    EXTERNAL FUNCTIONS AND SUBROUTINES:    ***
************************************************
```

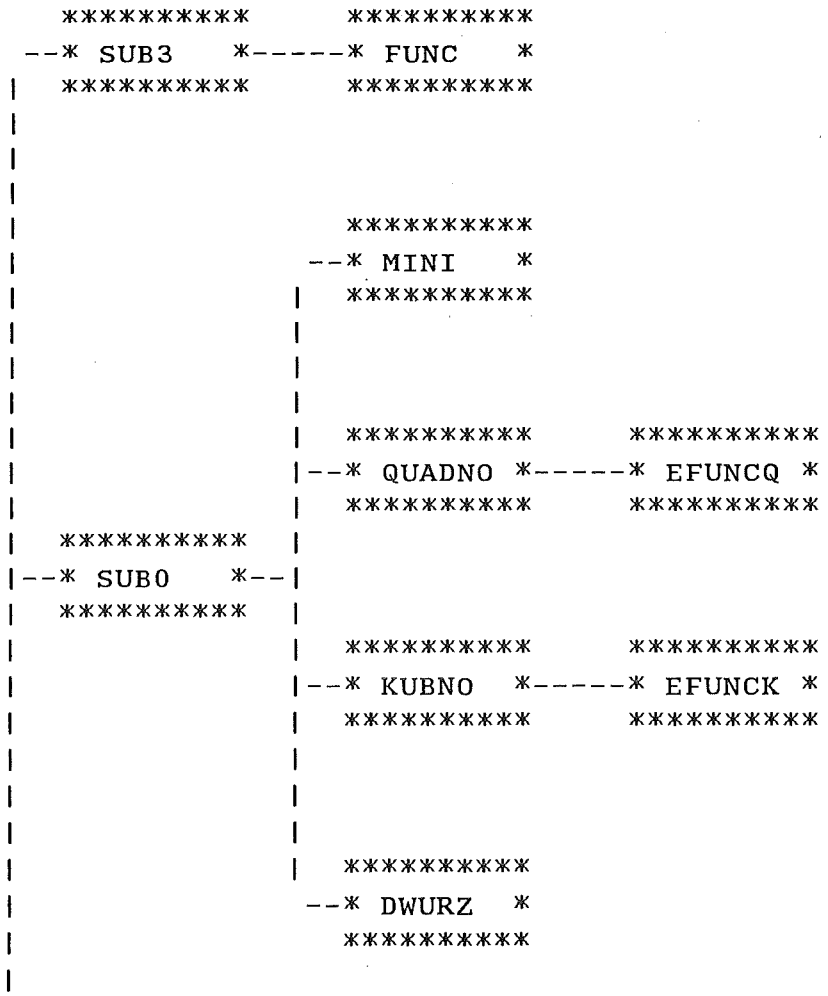
EXTERNAL	FROM PROCEDURE	TO PROCEDURE	WITH NAME
QUAD	MAIN	SUB1	SUBEX1
QUAD	MAIN	SUB1	SUBEX3
QUAD	MAIN	SUB2E1	SUBEXT
SUM	MAIN	SUB1	SUBEX2
SUM	MAIN	SUB1	SUBEX4
SUM	MAIN	SUB2E1	SUBEXT
SUM	MAIN	SUB2E2	SUBEXT
SIN	MAIN	SUB3	FUNC
COS	MAIN	SUB3	FUNC
TAN	MAIN	SUB3	FUNC
WURZEL	MAIN	----	----
SINHYP	MAIN	----	----
ENORM	SUB0	QUADNO	EFUNCQ
UNORM	SUB0	KUBNO	EFUNCK

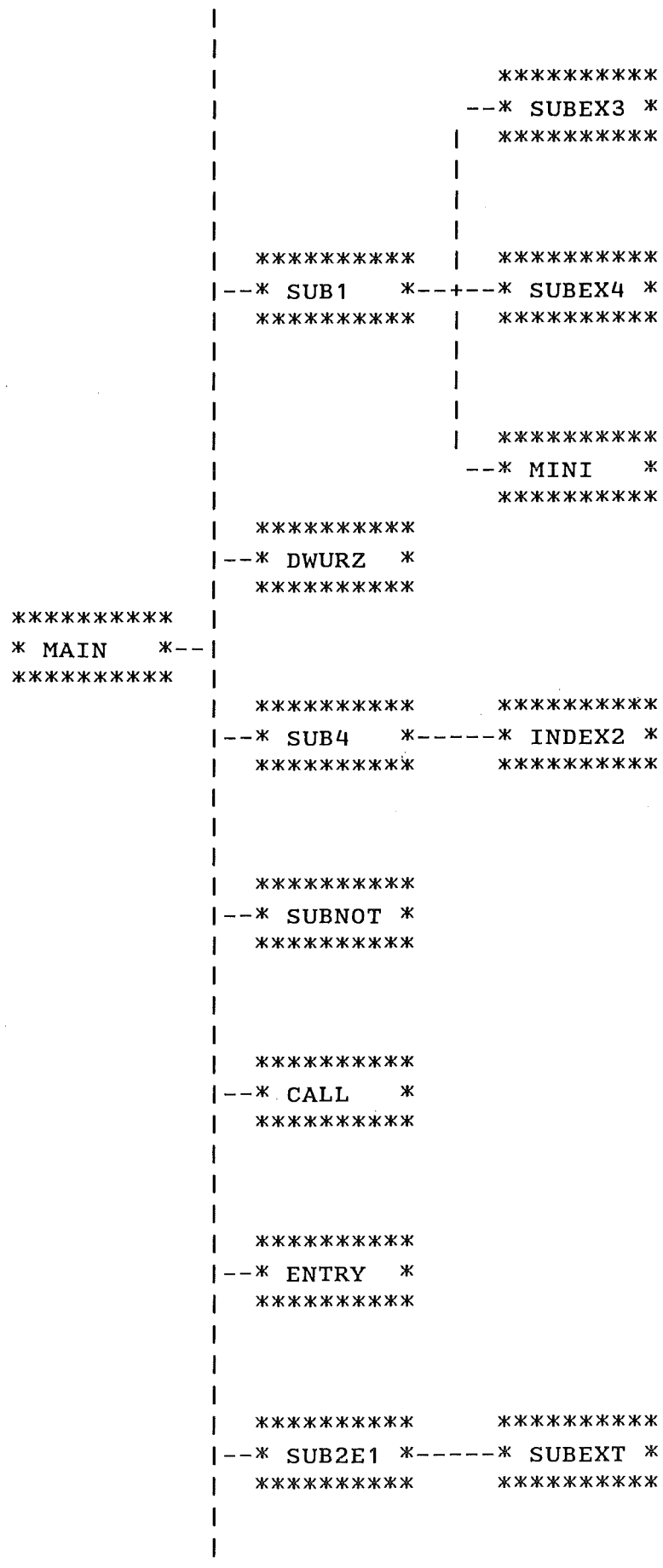
```
*****************************************************
***    SUBROUTINES WHICH WERE NOT IN SOURCE CODE    ***
*****************************************************

SUBNOT, CALL, ENTRY

*********************************************************
***    NOT REFERENCED SUBROUTINES IN SOURCE CODE    ***
```

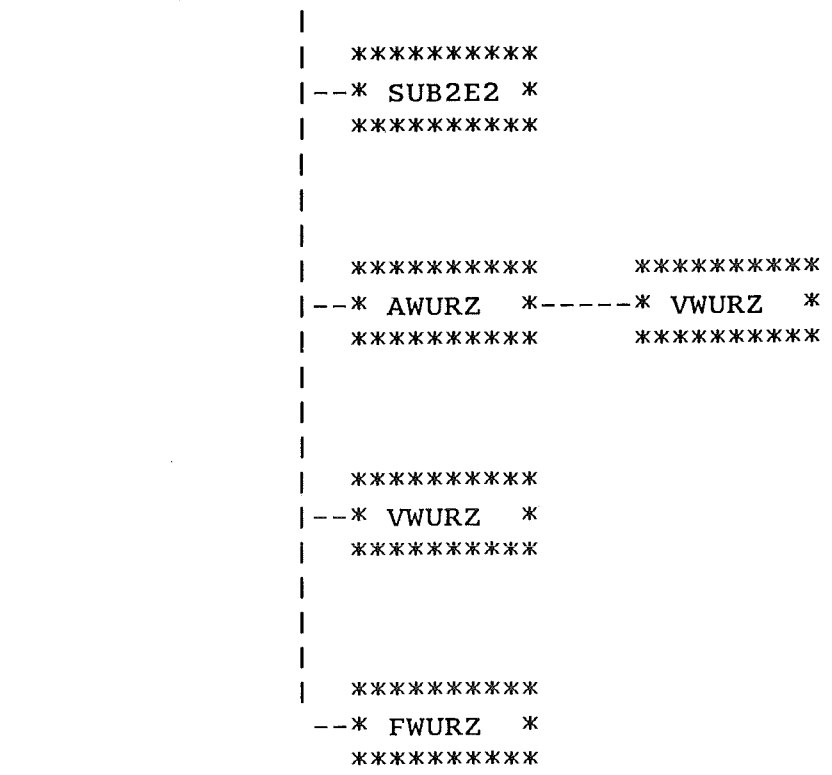
```
**********************************************************

SUB2, QUAD, DUMMY, SUBEX3, SUBEX4, SUBEXT

**********************************************************
***    NOT REFERENCED FUNCTIONS IN SOURCE CODE    ***
**********************************************************

ENORM, LOGER, FUNC, EFUNCQ, EFUNCK

**********************************************************************
```

```
**************************************************
***  TREE FROM PROGRAM AIT496.TESTBSP1.FORT  ***
**************************************************
```

In this file the name of the selected module is printed in the leftmost part of the page. From it a branching structure expands including the successive hierarchical module levels up to the seventh. This constitutes the upper bound of the band structure and is imposed by the maximum paper sheet width of usual printer devices. If the user's program has more than 7 levels of module invocations it is possible to obtain a complete calling tree of its source by attaching the invocation bands reports for several source code procedures together. However this is a very laborious solution, thus it is suggested to make instead use of the plot option which allows the graphic representation of up to twenty hierarchical levels. Should also this measure not be enough, then the user shall repeatedly run ARBUS by declaring one by one all the names of the modules in the last plotted level as new starting points till the related branchings have been completed and continue in the same way till the very last level of every subsequent starting point has been eventually reached. Yet, in most of the practical cases, this cumbersome series of operations is not needed.




MAIN
SUB1
SUBEX3
SUBEX4
MINI
DWURZ
SUB4
INDEX2
SUBNOT
CALL
ENTRY
SUB2E1
SUBEXT


SUB2E2
AWURZ
VWURZ
VWURZ
FWURZ

****

```
STATISTICS FROM PROGRAM AIT496.TESTBSP1.FORT
============================================

NUMBER OF SUBROUTINES ..................   9

NUMBER OF FUNCTIONS ....................   9

NUMBER OF BLOCK DATA ROUTINES ..........   3

NUMBER OF ENTRIES ......................   2

NUMBER OF EXTERNALS ....................   9

NUMBER OF NOT PRESENT SUBROUTINES ......   3

NUMBER OF LINES ........................ 241

NUMBER OF PROGRAM STATEMENTS ........... 148

NUMBER OF CONTINUATION LINES ...........   5

NUMBER OF COMMENT LINES ................  88

MAXIMUM CALL DEPTH (START PROCEDURE: 0) .....  3

LENGTH OF TREE DIAGRAM PLOT .....   0.200 M =   7.87 INCHES
!

WIDTH OF TREE DIAGRAM PLOT ......   0.130 M =   5.12 INCHES
!

LENGTH OF TREE DIAGRAM DATASET ...   112 LINES !
```

Example No. 2

This example reflects essentially the same structure of the previous one but VERSATEC call tree plots were added relating to JCL's where the branching depth and/or the starting point were changed (s. Figs. 1 to 4). In the diagram where the max. depth has not been reached a dash mark is attached straight to the right side of the cases enclosing the names of the procedures which have some CALLees. This mark is to make the user aware that "something" follows on there but does not appear on the sheet. An analogous pattern was implemented also in the call tree report by making use of appropriate printing characters. In Fig. 5 the tree diagram of ARBUS itself is presented.

```
//AIT496Z JOB (0496,935,P0000),FERRERO,NOTIFY=AIT496,MSGCLASS=H,
//   REGION=4000K,TIME=(1,0)
// EXEC F7CLG,PARM.C='LANGLVL(77),ASTER,SOURCE,NOMAP,NOLIST'
//C.SYSIN DD DSN=AIT496.BAUM.FORT(ARBUS),DISP=SHR
//G.SYSIN DD *
1
AIT496.TESTBSP2.FORT
SUB2
1
AIT496.CALLTAB.DATA
AIT496.TREE.DATA
AIT496.STATIS.DATA
```

```
**********************************************************************
```

```
C=====================================================================C
C                                                                     C
C     T E S T B E I S P I E L  2                                      C
C                                                                     C
C=====================================================================C
      CALL WURZEL (A,B,C)
      R = FUNW (C)
      CALL SUB2 (D)
      WRITE (*,*) 'WIE GROSS SOLL N SEIN ?'
      READ  (*,*) N
      IF (N .GT. 10) THEN
         CALL SUB4E2 (M,N)
      ELSE IF (N .GE. 0) THEN
         CALL SUB4E4 (M,N)
      ELSE IF (N .GE. -10) THEN
         CALL SUB4E1 (M,N)
      ELSE
         CALL SUB4E3 (M,N)
      END IF
      IF (M .LT. 10000) CALL SUB4E2 (I,M)
      IF (I .LT. 100000) CALL SUB4E2 (J,I)
```

```
      WRITE (*,*) 'SPECIALCHECK !!! 3. WURZEL VON I = ',
     &DWURZ(REAL(I))
      E N D
C-----------------------------------------------------------------
      SUBROUTINE WURZEL (A,B,C)
      WRITE (*,*) 'A,B EINGEBEN:'
      READ  (*,*) A,B
      C = SQRT(A+B)
      END
C-----------------------------------------------------------------
      REAL FUNCTION FUNW (C)
      EXTERNAL EXT1, EXT2, EXT3
      EXTERNAL EXT4
      EXTERNAL EXT5
      EXTERNAL EXT6
      INTEGER  IFELD(2,2)
      DATA IFELD / 0,0,0,0 /
C
      IF (C.LT.0.0) CALL SUB1(C,D,IFELD(1,1),EXT1,EXT2,EXT3,
     &    IFELD(1,2))
      IF (C.EQ.0.0) THEN
         CALL SUB1(C,D,IFELD(1,1),EXT5,EXT4,EXT5,IFELD(2,1))
      ELSE
         CALL SUB1(C,D,IFELD(1,1),EXT6,EXT2,EXT3,IFELD(2,1))
      END IF
      WRITE (*,*) 'D = ',D,' 3. WURZEL VON D = ', DWURZ(D)
      FUNW = D
      END
C-----------------------------------------------------------------
      SUBROUTINE SUB1 (C,D,I1,EXFUN1,EXFUN2,EXFUN3,I2)
C
      D = EXFUN1(C) * EXFUN2(C) * EXFUN3(C)
      I1 = INT(D)
      I2 = - I1
      END
C-----------------------------------------------------------------
      REAL FUNCTION DWURZ*4(X)
      DWURZ=X**(1./3.)
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT1*4(X)
      EXT1=SQRT(-X)
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT2(X)
      EXT2=1. / X
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT3(X)
      IF (X.GT.0.) THEN
         EXT3 = 1.
      ELSE
         EXT3 = -1.
      END IF
```

```
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT4(X)
      EXT4=1./SQRT(X)
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT5(X)
      EXT5=X + 1.0
      END
C-----------------------------------------------------------------
      REAL FUNCTION EXT6(X)
      EXT6=SQRT(X)
      END
C-----------------------------------------------------------------
      SUBROUTINE SUB2 (D)
      EXTERNAL EXT7, EXT8
      IF (D .LT. DWURZ(0.5)) THEN
         CALL SUB3 (*10, D, EXT7, E, *20)
      ELSE
         CALL SUB3 (*10, D, EXT8, E, *20)
      ENDIF
   10 WRITE (*,*) 'SUBROUTINE AUSGEFUEHRT (:) E = ', E
      GOTO 999
   20 WRITE (*,*) 'SUBROUTINE NICHT AUSGEFUEHRT (!)'
  999 E N D
C-----------------------------------------------------------------
      SUBROUTINE SUB3 (*, D, EXTSUB, E, *)
      IF (D .LT. -1.0) THEN
         RETURN2
      ELSE
         IF (D .LT. DWURZ(1.0)) CALL EXTSUB (D,E)
      END IF
      RETURN 1
      END
C-----------------------------------------------------------------
      SUBROUTINE EXT7 (D, E)
      E=SIN(D)
      END
C-----------------------------------------------------------------
      SUBROUTINE EXT8 (D, E)
      E=COS(D)
      END
C-----------------------------------------------------------------
      SUBROUTINE SUB4 (M, N)
      LOGICAL LOGER
      ENTRY SUB4E1 (M,N)
      LOGER = .FALSE.
      CALL LOSUB (LOGER, N)
      ENTRY SUB4E2 (M,N)
      CALL SUM (M, N)
      IF (LOGER) M = -M
      RETURN
      ENTRY SUB4E3 (M,N)
      LOGER = .FALSE.
```

```
      CALL LOSUB (LOGER, N)
      ENTRY SUB4E4 (M,N)
      M = IFAKUL (M, N)
      IF (LOGER) M = -M
      RETURN
      E  N  D
C-----------------------------------------------------------------
      SUBROUTINE SUM (M,N)
      M = N * (N+1) / 2
      END
C-----------------------------------------------------------------
      SUBROUTINE LOSUB (LO, N)
      LOGICAL LO
      IF (MODULO(N,2) .NE. 0) LO = .NOT. LO
      END
C-----------------------------------------------------------------
      FUNCTION IFAKUL (N)
      LOGICAL LOHELP
      LOHELP = .FALSE.
      CALL LOSUB (LOHELP,I)
      M = 1
      DO 1000 I = 2, N
         M = M * I
 1000 CONTINUE
      IF (LOHELP) THEN
         IFAKUL = -M
      ELSE
         IFAKUL = M
      END IF
      END
C-----------------------------------------------------------------
      FUNCTION MODULO (M,N)
      MODULO = M - M/N * N
      END
```

****

```
PROTOCOL DATASET FROM AIT496.TESTBSP2.FORT
==========================================


******************************************
***   LIST OF CALLS FOR EVERY MODULE   ***
******************************************


THE MAIN-PROGRAM MAIN CALLS:
============================

WURZEL, FUNW, SUB2, SUB4E2, SUB4E4, SUB4E1, SUB4E3, DWURZ

THE SUBROUTINE WURZEL CALLS:
============================

     -----

THE SUBROUTINE SUB1 CALLS:
==========================

EXFUN1, EXFUN2, EXFUN3

THE SUBROUTINE SUB2 CALLS:
==========================

DWURZ, SUB3

THE SUBROUTINE SUB3 CALLS:
==========================

EXTSUB, DWURZ

THE SUBROUTINE EXT7 CALLS:
==========================

     -----

THE SUBROUTINE EXT8 CALLS:
==========================

     -----

THE SUBROUTINE SUB4 CALLS:
==========================

     -----

THE SUBROUTINE ENTRY SUB4E1 CALLS:
==================================
```

LOSUB

THE SUBROUTINE ENTRY SUB4E2 CALLS:
==========

SUM

THE SUBROUTINE ENTRY SUB4E3 CALLS:
==========

LOSUB

THE SUBROUTINE ENTRY SUB4E4 CALLS:
==========

IFAKUL

THE SUBROUTINE SUM CALLS:
==========

THE SUBROUTINE LOSUB CALLS:
==========

MODULO

THE FUNCTION FUNW CALLS:
==========

SUB1, DWURZ

THE FUNCTION DWURZ CALLS:
==========

THE FUNCTION EXT1 CALLS:
==========

THE FUNCTION EXT2 CALLS:
==========

THE FUNCTION EXT3 CALLS:
==========

THE FUNCTION EXT4 CALLS:

```
=========================

     -----

THE FUNCTION EXT5 CALLS:
========================

     -----

THE FUNCTION EXT6 CALLS:
========================

     -----

THE FUNCTION IFAKUL CALLS:
==========================

LOSUB

THE FUNCTION MODULO CALLS:
==========================

     -----


**************************************************
***   LIST OF CALLING MODULES FOR EVERY MODULE   ***
**************************************************


THE SUBROUTINE WURZEL IS CALLED FROM:
=====================================

MAIN

THE SUBROUTINE SUB1 IS CALLED FROM:
===================================

FUNW

THE SUBROUTINE SUB2 IS CALLED FROM:
===================================

MAIN

THE SUBROUTINE SUB3 IS CALLED FROM:
===================================

SUB2

THE SUBROUTINE EXT7 IS CALLED FROM:
===================================

     -----
```

THE SUBROUTINE EXT8 IS CALLED FROM:
==========

THE SUBROUTINE SUB4 IS CALLED FROM:
==========

THE SUBROUTINE ENTRY SUB4E1 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE ENTRY SUB4E2 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE ENTRY SUB4E3 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE ENTRY SUB4E4 IS CALLED FROM:
==========

MAIN

THE SUBROUTINE SUM IS CALLED FROM:
==========

SUB4E2

THE SUBROUTINE LOSUB IS CALLED FROM:
==========

SUB4E1, SUB4E3, IFAKUL

THE FUNCTION FUNW IS CALLED FROM:
==========

MAIN

THE FUNCTION DWURZ IS CALLED FROM:
==========

MAIN, SUB2, SUB3, FUNW

THE FUNCTION EXT1 IS CALLED FROM:
==========

THE FUNCTION EXT2 IS CALLED FROM:
==========

THE FUNCTION EXT3 IS CALLED FROM:
==========

THE FUNCTION EXT4 IS CALLED FROM:
==========

THE FUNCTION EXT5 IS CALLED FROM:
==========

THE FUNCTION EXT6 IS CALLED FROM:
==========

THE FUNCTION IFAKUL IS CALLED FROM:
==========

SUB4E4

THE FUNCTION MODULO IS CALLED FROM:
==========

LOSUB

****
*** BLOCK-DATA-SUBROUTINES ***
****

*** THERE ARE NO BLOCK-DATA-ROUTINES IN THE SOURCE CODE ***

****
*** ENTRYS IN ROUTINES ***
****

ENTRYS OF SUB4:
==========

SUB4E1, SUB4E2, SUB4E3, SUB4E4

****
*** EXTERNAL FUNCTIONS AND SUBROUTINES: ***
****

EXTERNAL	FROM PROCEDURE	TO PROCEDURE	WITH NAME
EXT1	FUNW	SUB1	EXFUN1
EXT2	FUNW	SUB1	EXFUN2
EXT3	FUNW	SUB1	EXFUN3
EXT4	FUNW	SUB1	EXFUN2
EXT5	FUNW	SUB1	EXFUN1
EXT5	FUNW	SUB1	EXFUN3
EXT6	FUNW	SUB1	EXFUN1
EXT7	SUB2	SUB3	EXTSUB
EXT8	SUB2	SUB3	EXTSUB

****
*** SUBROUTINES WHICH WERE NOT IN SOURCE CODE ***
****

*** ALL THE CALLED SUBROUTINES ARE PRESENT IN THE SOURCE CODE ***

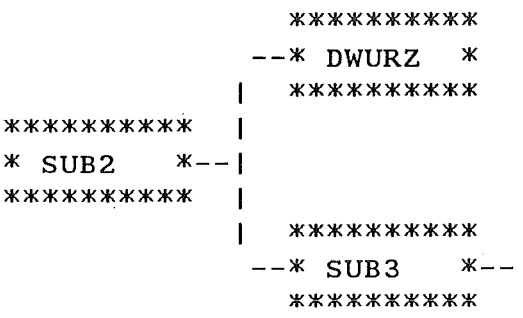
****
*** NOT REFERENCED FUNCTIONS IN SOURCE CODE ***
****

EXT1, EXT2, EXT3, EXT4, EXT5, EXT6, EXFUN1, EXFUN2, EXFUN3

****

```
STATISTICS FROM PROGRAM AIT496.TESTBSP2.FORT
============================================

NUMBER OF SUBROUTINES ..................... 9

NUMBER OF FUNCTIONS ....................... 10

NUMBER OF BLOCK DATA ROUTINES ............. 0

NUMBER OF ENTRIES ......................... 4

NUMBER OF EXTERNALS ....................... 8

NUMBER OF NOT PRESENT SUBROUTINES ......... 0

NUMBER OF LINES ....................... 162

NUMBER OF PROGRAM STATEMENTS ........... 135

NUMBER OF CONTINUATION LINES ........... 1

NUMBER OF COMMENT LINES ................ 26

MAXIMUM CALL DEPTH (START PROCEDURE: 0) ..... 1

LENGTH OF TREE DIAGRAM PLOT .....   0.045 M =  1.77 INCHES !

WIDTH OF TREE DIAGRAM PLOT ......   0.070 M =  2.76 INCHES !

LENGTH OF TREE DIAGRAM DATASET ...    19 LINES !

*****************************************************************
```

```
*******************************
***  AIT496.TESTBSP2.FORT  ***
*******************************
```



2.4 PARTICULARITIES AND LIMITATIONS

In order to keep the memory size needed by a program run below 5,000 Kbytes, it was decided to fix up the maximum dimensions of some crucial arrays in the program corresponding to as many user relevant quantities like the number of subroutines and/or functions, the number of entries, that of block data subprograms etc. which can be processed by ARBUS.

Table 1 summarizes the max. allowed values of the code parameters of major interest for the user as provided by ARBUS.

Parameter Description	Max. Value
no. of SUBROUTINEs	1600 *)
no. of FUNCTIONs	300
no. of ENTRY points	200
no. of BLOCK DATA subprograms	100
no. of SUBROUTINEs called within the program yet not available in the program itself	200
no. of CALL statements within each SUBROUTINE	300
no. of EXTERNAL declarations	200
max. length (in records) of the output data set	70,000
max. width (in columns) of the output data set	132 **)
max. length (in m) of the VERSATEC plot	100
max. width (in m) of the VERSATEC plot	0.584 ***)

Table 1. Maximum values of the major code parameters

*) includes ENTRY points and SUBROUTINEs called but not present in source code
**) corresponding to a 7-fold nested CALL-structure
***) corresponding to a 20-fold nested CALL-structure

If the program being analysed contains any couple of procedures invoking one another (recurrent call schema), then ARBUS recovers the error and continues generating the tree giving a warning message in the job output.

As it was possible to create a correct tree diagram of large and highly structured codes such as CONTAIN /6/ and TWODANT /7/, it was felt that the limit values given in Tab. 1 may be a reasonable compromise between memory saving requirements and

the user's wish of analysing as much automatically as possible large computer codes' structures with a reliable (possibly fault-free) software tool. However all the ARBUS parameters can be easily reset to higher values, provided the resulting core memory requirement doesn't exceed 10,000 Kbytes viz the max. available memory size on the IBM 3090 computer of Kernforschungszentrum.

The execution time depends on the program size and on the ARBUS options in use. A rough estimate for a run with call tree print and plot output request would be approximately 4 seconds per 10,000 lines on the above mentioned IBM 3090 system.

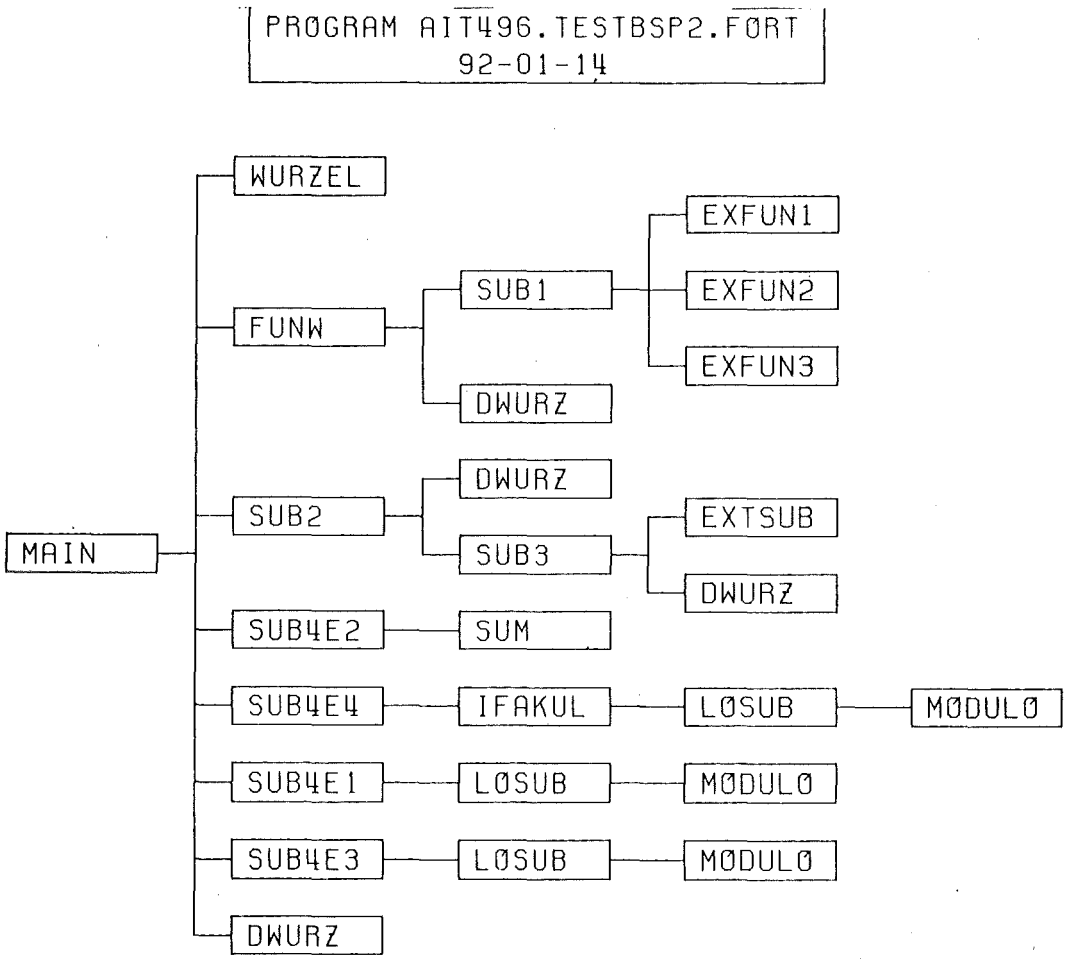


Figure 1. Complete call tree diagram of the test program TESTBSP2.FORT

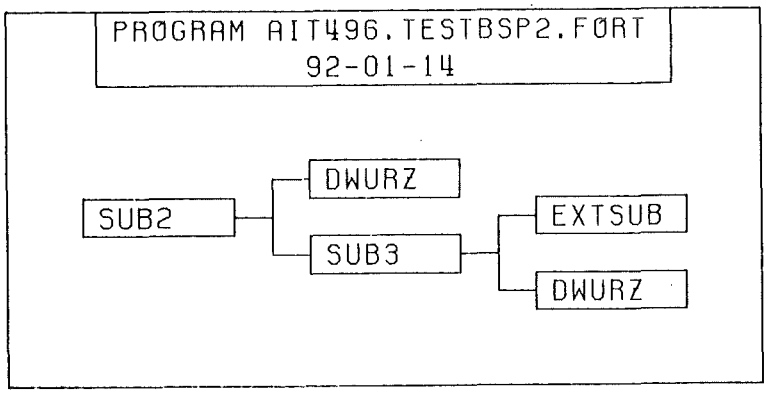


Figure 2. Partial call tree diagram of the test program TESTBSP2.FORT with the subroutine SUB2 as a starting point

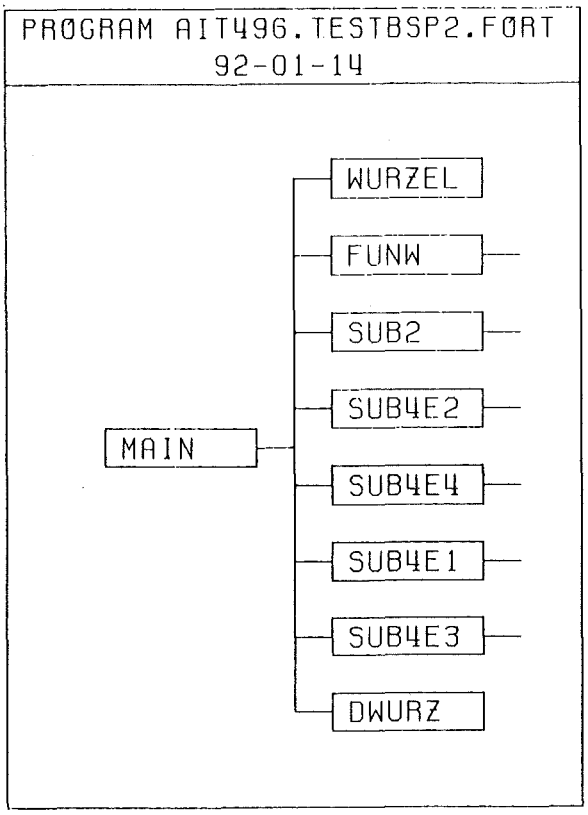


Figure 3. Partial call tree diagram of the test program TESTBSP2.FORT with the MAIN program as a starting point and selected branching depth equal "1"

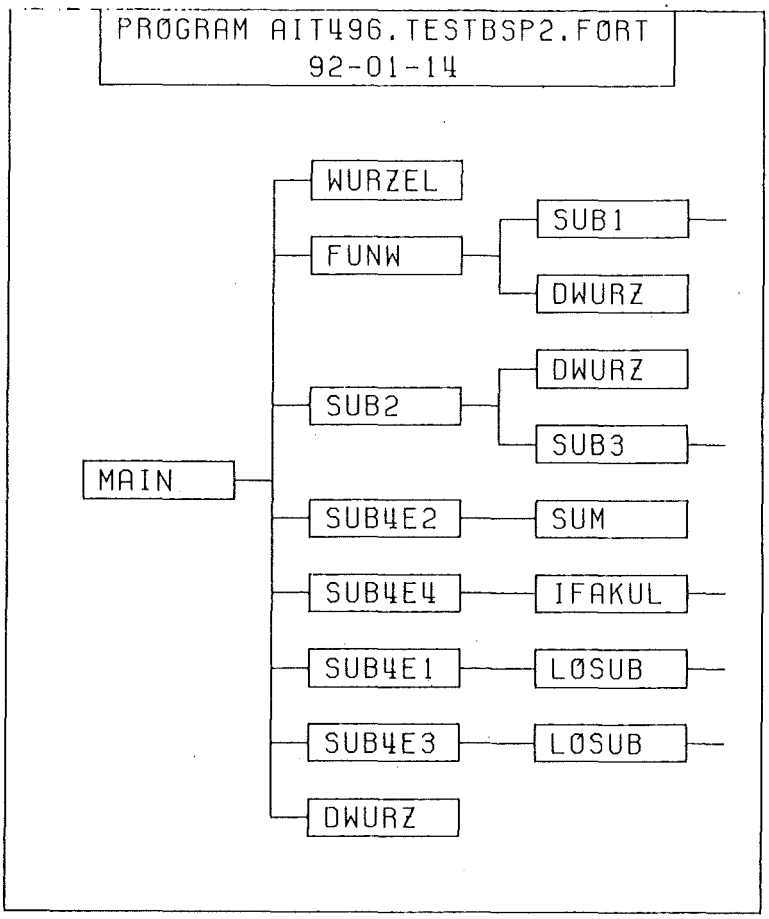


Figure 4. Partial call tree diagram of the test program TESTBSP2.FORT with the MAIN program as a starting point and branching depth equal "2"

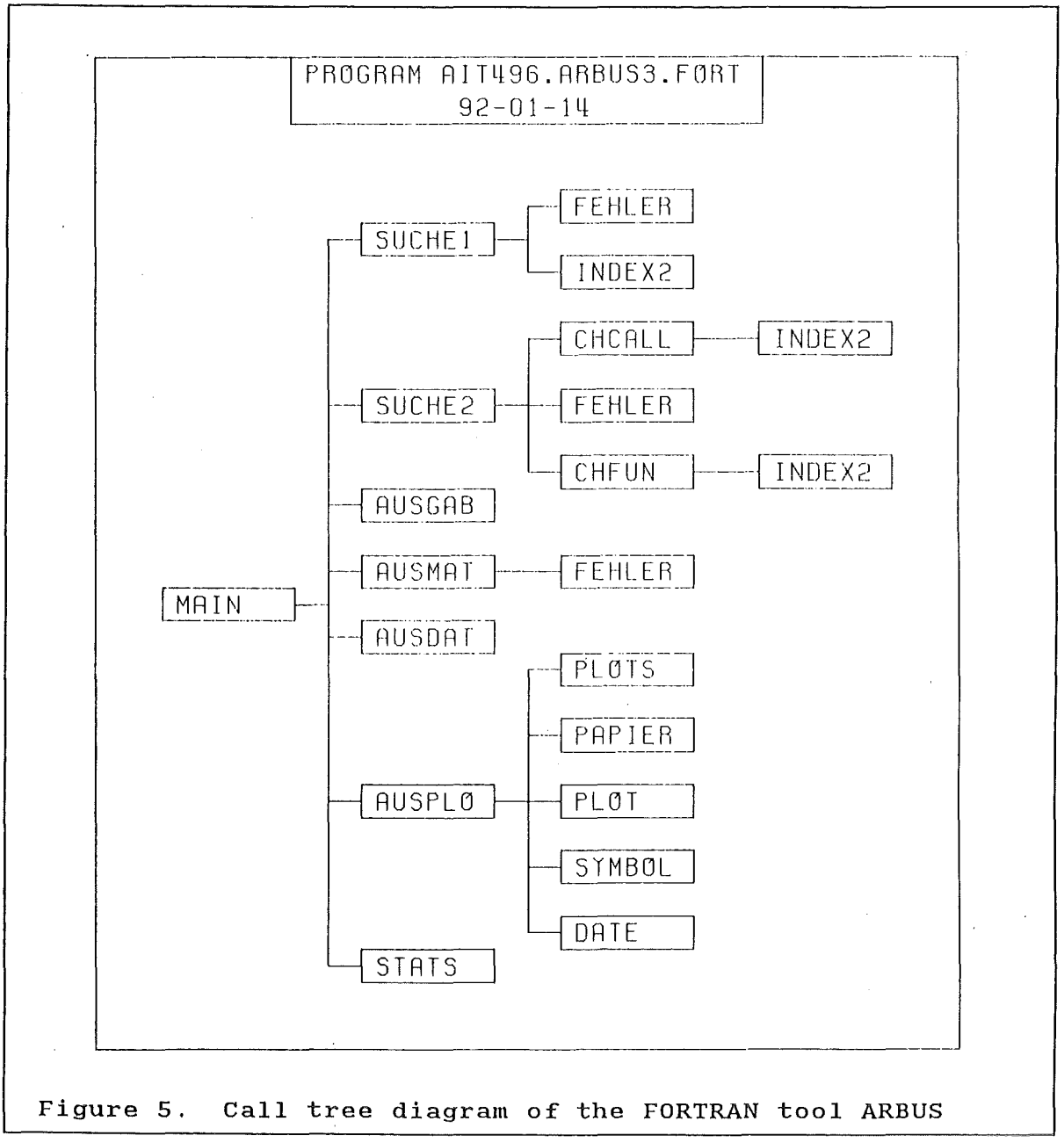


Figure 5. Call tree diagram of the FORTRAN tool ARBUS

3. REFERENCES

/1/ RXVP80 - The Verification and Validation System for FORTRAN Version 4.0, User's Manual, RM-2419, 1985

/2/ STREAM77 Siemens System 7.800 Handbook, April 1987

/3/ P. Fette,
PLOT-Handbuch, Anleitung für die Benutzung der Plotter-Software, Programmbeschr. Nr. 268, Kernforschungszentrum Karlsruhe, October 1982

/4/ B. Manes,
Private Communication

/5/ Siemens System 7.800 FORTRAN77 Reference Manual, October 1985

/6/ K.D. Bergeron et al.,
User's Manual for CONTAIN 1.0, A Computer Code for Severe Nuclear Reactor Accident Containment Analysis, Sandia National Laboratories, SAND84-1204, NUR-EG/CR-4085, 1985

/7/ R.E. Alcouffe, F.W. Brinkley,Jr.; D.R. Marr and R.D. O'Dell,
User's Manual for TWODANT: A Code Package for Two-Dimensional, Diffusion-Accelerated, Neutral-Particle Transport, Los Alamos National Laboratory Report LA-10049-M-Rev.1, October 1984