

# NASA Technical Memorandum 86326

(NASA-TM-86326) A NEW IMPLEMENTATION OF THE  
PROGRAMMING SYSTEM FOR STRUCTURAL SYNTHESIS  
(PROSS-2) (NASA) 36 p CSCL 09B

N87-18330

Unclas  
G3/61 43375

A NEW IMPLEMENTATION OF THE PROGRAMMING SYSTEM FOR  
STRUCTURAL SYNTHESIS (PROSS-2)

IN-61  
58125  
DATE OVERRIDE  
P-36

James L. Rogers, Jr.



NOVEMBER 1984

Date for general release November 30, 1986

## NASA

National Aeronautics and  
Space Administration

Langley Research Center  
Hampton, Virginia 23665

A NEW IMPLEMENTATION OF THE PROGRAMMING SYSTEM  
FOR STRUCTURAL SYNTHESIS (PROSSS-2)

James L. Rogers, Jr.

NASA Langley Research Center

Hampton, Virginia

INTRODUCTION

The Programming System for Structural Synthesis (PROSSS), which combines analysis and optimization techniques with applications to structures, was first released to the public in December 1981 (refs. 1,2) as the first step in a series of implementations. This first implementation combined a finite element program for structural analysis, SPAR (ref. 3) with a large general purpose optimization program, CONMIN (ref. 4) and several small problem-dependent FORTRAN programs and subroutines which must be written by the user to interface the analysis and optimization programs. All of the programs were connected by a network of control cards in the standard Control Data Corporation CYBER Control Language (CCL, ref.5). Following implementations have included distributing PROSSS between a mainframe and a minicomputer (refs. 6,7) to take advantage of the best features of both computers, and incorporating PROSSS entirely within the Engineering Analysis Language (EAL) computer program to simplify the maintenance, control, and data management (ref. 8).

During training sessions with potential users of the first implementation of PROSSS, it became apparent that the many files and control cards involved in creating a PROSSS execution system, made the

system very complex and difficult for a new user to comprehend. A much simpler system, EAL-PROSSS, was available, but users would have to purchase a copy of the proprietary EAL computer program in order to use the system. Although SPAR lacks a number of the capabilities found in EAL, it was decided to try and duplicate EAL-PROSSS using SPAR. This would result in a much easier to use system where all of the components are in the public domain. In addition, this system would not be restricted to the CDC computers, but could be implemented on many computers since versions of SPAR exist that are written completely in FORTRAN.

This paper describes this new implementation of PROSSS, hereafter called PROSSS-2. The paper is divided into six major parts: (1) a step-by-step approach for creating and executing PROSSS-2, (2) changes made to SPAR to allow this implementation, (3) problem-independent code, (4) SPAR runstreams, (5) problem-dependent code, and (6) combining new processors into an absolute overlay. A final, less significant part discusses how a user would employ other analysis and optimization programs in place of SPAR and CONMIN.

The purpose of this paper is to describe a new method for implementing a flexible software system combining large, complex program with small, user-supplied problem dependent programs into a single system. An assumption is made that a potential user of PROSSS-2 has gained an understanding of PROSSS by reading references 1-4. Identification of commercial products in this report is used solely for describing the system. The identification of these commercial products does not constitute official endorsement, expressed or implied, of such products by the National Aeronautics and Space Administration.

## STEP-BY-STEP APPROACH FOR CREATING AND EXECUTING PROSS-2

1. Create a relocatable version of PROSS-2 by combining the relocatable code from SPAR, modifications to SPAR, and the problem-independent routines.
2. Test the model by coding the non-repeatable SPAR runstream and input it into the standard version of SPAR for execution.
3. Code the repeatable SPAR runstream. If Option 2.3 is to be used, execute the program for generating analytical runstreams using the non-repeatable and repeatable runstreams as input. Save the output runstreams for future use.
4. Code and compile the problem-dependent routines such as the front and end processors. If Option 2.3 is to be used, code a routine for chain differentiation.
5. Create an absolute version of PROSS-2 by combining the relocatable version of PROSS-2 with the relocatable code for CONMIN, the problem-dependent routines, and the SPAR COM code.
6. Execute the absolute version of PROSS-2 using the non-repeatable runstream with data initializations added to AUS as input. Save Library 1 for future use.
7. Execute the absolute version of PROSS-2 using the repeatable runstream and Library 1 from step 6 as input.
8. If changes are required to the repeatable runstream, repeat steps 3 (if Option 2.3 is used) and 7. If changes are required to the non-repeatable runstream, repeat steps 3 (if Option 2.3 is used), 6 and 7. If changes are required to the problem-dependent code, repeat steps 4, 5, and 7.

## MODIFICATIONS TO SPAR

To incorporate PROSSS-2 into SPAR it was necessary to alter three routines in SPAR. These routines include the main program and the drivers for the DCU and AUS processors. Changes made to the main program allow three new processors to be added to SPAR. These new processors, named OPTD, ENDP, and DKDV, were added to the SPAR overlay by increasing the array NP (NP is the number of processors) to 36 and adding the names of the new processors to the DATA statement for NP. The new processor OPTD calls the optimizer and the front processor for converting from optimization to analysis. ENDP calls the end processor for converting from analysis to optimization. DKDV computes the factors used in a chain differentiation for calculating the derivative of the stiffness matrix with respect to the design variable.

A new subprocessor was needed in DCU. This subprocessor called CLEAN erases all data on a library. The code is based on the method of handling scratch libraries in SPAR. Scratch libraries are created within a processor and released upon exiting the processor. The FREE command in EAL works similarly. The CLEAN command was needed because the optimization process is iterative and the libraries tend to grow to exceed the available disk space. Use of the COPY command in DCU should be kept to a minimal because of the I/O costs involved.

The new processor DKDV creates a data set called FACT DKDV 1 1 which contains the factors used in the chain differentiation calculating the derivative of the stiffness matrix with respect to the design variable. In the original version of PROSSS, a FORTRAN program created a SPAR runstream that was later executed to handle this calculation. In EAL/PROSSS the

factors were stored in registers after being retrieved from the data set. Since these factors change in each iteration and PROSSS-2 does not exit SPAR as the original PROSSS did; nor does SPAR have registers like EAL, modifications had to be made to AUS. These modifications affect the subprocessor SUM in AUS. If this derivative is needed, such as for analytical gradients, the SPAR runstream will contain a command for SUM in AUS. The factor before each data set in SUM will be between 9901. and 9929.. The new code in AUS will subtract 9900. from each factor to determine the location of the factor within the FACT DKDV data set. This factor will then be substituted for the 9900. factor in the runstream and the sum calculated. More details on this operation appear in a later section.

#### PROBLEM-INDEPENDENT CODE

There are several new pieces of code that have been added to SPAR to form PROSSS-2 which are problem-independent and therefore will not have to be modified by the user as the problem changes. These codes contain drivers for the three new processors, drivers for CONMIN, CONMIN, and a subroutine for computing finite difference gradients. Table 1 lists all of the problem-independent routines, their purpose, and the data sets they use. Even though users will not have to modify these codes, it is important that they have an understanding of how they work.

The driver for the new processor OPTD calls the CONMIN driver and the front processor. In PROSSS-2 there are only three options available as opposed to the five options in previous implementations. These options are Option 1.1 (flowchart in figure 1) which uses a nonlinear programming

method with the gradients computed in CONMIN by finite differences, Option 2.2 (flowchart in figure 2) which uses a piecewise linear programming method with the gradients computed external to CONMIN also by finite differences, and Option 2.3 (flowchart in figure 3) which also uses a piecewise linear programming method but the gradients are computed analytically external to CONMIN. Options 1.2 and 1.3 which both use a nonlinear programming method are not available in PROSS-2 because experience has shown that most users seldom use these two options. The option number is stored in a data set created by the user and called OPT NO 1 1. The OPTD driver reads this data set and then branches to the set of statements required to execute the desired option.

OPTD calls two subroutines for driving CONMIN. These subroutines are CNMDRV1 and CNMDRV2. CNMDRV1 is used with Option 1.1 and CNMDRV2 is used with Options 2.2 and 2.3. These subroutines are similar to CONMS1 and CONMS2 in PROSS. There is only one other subroutine of interest that is called by OPTD, that is the subroutine EVALNG. This subroutine is only called by Option 2.2 and is used for computing finite difference gradients. It is similar to the subroutine EVALS in PROSS. The primary difference between these subroutines in PROSS-2 and their counterparts in PROSS is that the subroutines in PROSS-2 pass data through SPAR libraries, while those in PROSS pass data through files.

The driver for the ENDP processor also depends on the option. If no analytical gradients are required (Option 1.1 or 2.2) then just the end processor is called. The end processor subroutine is problem-dependent and must be coded by the user. It is discussed in more detail in a later section. If analytical gradients are required (Option 2.3) then another

problem dependent subroutine, DRVS, may be called before the end processor. Whether or not DRVS is called depends on if the user needs to convert forces and moments and the derivatives of forces and moments to stresses and stress derivatives. The user must create a data set called DRV CHEK 1 1 which is read by DRVS. A one (1) is placed in the data set if DRVS is to be called, otherwise only the end processor is called.

The driver for the DKDV processor calls a subroutine to compute the factors needed for the chain differentiation in calculating the derivative of the stiffness matrix with respect to the design variable. This is required when an element has more than one contributing factor. An example of a factor computed by this subroutine is  $DA/DV$ , the derivative of the cross-sectional area with respect to the design variable. The subroutine called by DKDV is problem-dependent and must be supplied by the user.

#### CREATING SPAR RUNSTREAMS FOR PROSS-2

This section and the next are the most critical in terms of user understanding when solving a problem with PROSS-2. There are two areas of code that must be developed to set up PROSS-2 for solving a problem. The first area is the SPAR runstream discussed in this section and the second is the problem-dependent FORTRAN code to be added to SPAR discussed in the next section. The SPAR runstream is divided into three parts, the non-repeatable part, the repeatable part, and a part specifically for analytical gradients. For PROSS-2 the non-repeatable and repeatable parts are two separate and distinct steps.



### Non-repeatable SPAR Runstream

The non-repeatable SPAR runstream has two functions. The first is to set up the geometry of the model using the TAB and ELD processors. The user should do this part first and execute SPAR with just these two processors until satisfied that the model is correct. Processors PLTA and PLTB can also be used at this point to aid in verifying the model geometry.

Once the user is satisfied with the model geometry, processor AUS can be added to this runstream. This is different from the first implementation of PROSSS, but is a spinoff from a method used in EAL/PROSSS. The TABLE subprocessor in AUS is used to initialize variables for later use in PROSSS-2. One drawback to using the TABLE subprocessor is that it will not accept integer numbers, thus all values must be assigned using a decimal. The problem-independent code will handle most of the conversion to integer. The user must take care of this conversion if it is needed in any problem-dependent code. These initialized variables are stored in SPAR data sets on SPAR library 1. (Note: Library 1 should be saved after the non-repeatable runstream has been executed) In general the TABLE command is as follows:

```
TABLE(NI=1,NJ=xxx); n1 n2 n3 n4
```

```
J=1,xxx
```

values for data set separated by semi-colons

where

xxx is the number of values in the data set

n1 n2 n3 n4 is the data set name

(Note: the ,xxx in J=1,xxx can be omitted if NJ=1)

EXAMPLE (set up the initial design variables)

```
TABLE(NI=1,NJ=3); DESV CNMN 1 1
```

```
J=1,3
```

```
.24; 1.0; 10.
```

Table 2 lists all of the data set names, the number of values in each data set, and the purpose of each data set input in the non-repeatable runstream. TBD under the number of values for each data set means that this number is problem-dependent and is to be determined by the user. The data set names and NJ should not be changed for any data sets being used by problem-independent code. An asterisk (\*) before the data set name implies that this data set is optional, depending on the user-supplied problem-dependent code. The user can add or change any data sets required for problem-dependent code. Unless otherwise indicated by an option number in parentheses, each data set should be created for each option.

#### Repeatable SPAR Runstream

The repeatable SPAR runstream will be executed iteratively as in the first implementation of PROSSS. Library 1, saved from the execution of the non-repeatable runstream, is used as input. Typically, the SPAR repeatable runstream will contain processors such as TOPO, E, EKS, K, INV, AUS, SSOL, and GSF which are all discussed in detail in the SPAR reference manual. The user can set up the repeatable runstream using just these processors and append it to his non-repeatable runstream to make a complete run through SPAR for testing. After this initial test, the two runstreams should be kept separate.

The repeatable runstream must begin with an [XQT OPTD to call the option driver, which in turn calls CONMIN and the front processor. The repeatable runstream must end with an [XQT ENDP to call the end processor, followed by an [XQT EXIT. The end processor checks to see if the optimization process is complete or requires another iteration. If another iteration is required, then the file (unit 5, the INPUT file for SPAR) is rewound by the end processor and the repeatable runstream is executed again from [XQT OPTD. If the optimization process is complete this file is not rewound and the [XQT EXIT command is processed and the repeatable part is terminated.

The main difference between the repeatable part in PROSSS-2 and PROSSS is that in PROSSS-2 no exit is made from SPAR during the entire iterative process. In SPAR, when a data set is created with a name that is already in the library, then the already existing data set is disabled and the new data set is added to the library. Even though the data set is disabled the data is not removed from the disk. In an iterative process, data sets with identical names are created over and over again causing large amounts of data to be stored on a disk. A FREE command exists in EAL which allows the user to release and return disk space to the host operating system. No similar command exists in SPAR, so it was added in the form of the CLEAN subprocessor in DCU (described above in section about modifications to SPAR). Therefore DCU must be executed after GSF and before ENDP. In DCU, certain large data sets which are local to a particular iteration should be disabled. This will reduce the I/O costs when performing a COPY. The data sets which should be disabled include KMAP, AMAP, all EFIL's, DEM DIAG, K SPAR, INV K, APPL FORC, STAT DISP, STAT REAC, and any others that will be

recomputed in the next iteration. After the DISABLES, do a COPY 1,6 which will copy all enabled data sets from library 1 to library 6. Follow this with a CLEAN 1 which releases all space on the disk claimed by library 1. Next is a DUPLICATE 6,1 which is an inexpensive method for duplicating all data sets on library 6 (regardless of whether they are disabled or not) back on to library 1. Finally a CLEAN 6 command is executed to clean up the disk space for library 6. This command is followed by the [XQT ENDP.

#### Program for Generating Analytical Runstreams

In order to compute analytical gradients in Option 2.3, special additions must be made to both the non-repeatable and repeatable runstreams. These additions are made using a supplied program called GENRS. The user creates the non-repeatable and repeatable runstreams just as would be done for either of the other options. These runstreams are then input to GENRS along with a file containing certain information about the model. The non-repeatable runstream is input on unit 21 and the modified non-repeatable runstream is output on unit 20. The repeatable runstream is input on unit 30 and the modified repeatable runstream is output on unit 31. Unit 5 is the file containing information about the model. The execution statement for this program should have the files in the following order:

```
LGO(TAPE5,TAPE21,TAPE20,TAPE30,TAPE31)
```

There is only one slight change that has to be made to the non-repeatable runstream for this program. SPAR comment cards, \$START and \$END are used in the ELD processor to offset the different elements used as design variables. For example, a \$START would appear immediately before

a E21 card in ELD and a \$END would appear immediately after the last connection card for the E21 elements and immediately before a \$START or [XQT for a new processor. The output modified runstream contains the original non-repeatable runstream with modifications appended to the end. These modifications include the UPDATE mode of the TAB processor in which all design variables are set to unity for computing the derivatives. All of the data sets created to this point on library 1 are then copied over to library 2. A pass is made through ELD, E, EKS, TOPO, and K to compute the derivative of the mass matrix with respect to the design variable from data set DEM DIAG and the derivative of the stiffness matrix with respect to the design variable from data set K SPAR. Processor DCU is used to change the name from DEM DIAG to DMDV DIAG 0 n and from K SPAR to DKDV SPAR m n where n is the number of the design variable and m is degrees of freedom from the SPAR START card (not to be confused with the \$START card) squared. This data set is then copied to library 2.

If a design variable has more than one contributing factor then the program handles them in a special way. First the number of the design variable is multiplied by 100, then the number of the contributing factor is added to it to form a key for that design variable and its contributing factor. Thus for the mass matrix, the name is changed to DMDV DIAG 0 key and for the stiffness matrix the name is changed to TEMP SPAR m\_key . These data sets are used by AUS in the modified repeatable runstream in a chain differentiation to combine the contributing factors into a single derivative for the stiffness matrices.

EXAMPLE: Suppose a model has three elements contributing design variables and the second set of elements from the three

is the beam elements. Also suppose that the beams have four contributing factors of which the cross-sectional area is the third. Then the number of the beam design variable is 2 which is multiplied by 100 to give 200 and the number of the cross-sectional area contributing factor is 3 which is added to 200 to give 203. Thus 203 is the key for the cross-sectional area contributing factor to the beams.

The modified repeatable runstream which is output from the program contains the original repeatable runstream down through GSF with the modifications appended to it. If needed, the modifications contain calls to DKDV and AUS to perform a chain differentiation on design variables with more than one contributing factor. After that processors AUS, SSOL, and GSF are used to compute displacement derivatives and the derivatives of the stresses for each different design variable with respect to each other design variable. Thus, for example, if there are three elements with design variables, then there will be nine (9) data sets created. The DCU processor is used to again change the names from STRS Exy k n where Exy is the element name (eg. E23), k is a new load case number, and n is the number of the design variable. DCU is also used to store these data sets onto library 1 for use by the end processor. The final part of the original repeatable processor containing DCU with the DISABLEs, COPY 1,6, CLEAN 1, DUPLICATE 6,1, CLEAN 6, [XQT ENDP, and [XQT EXIT are then appended to the modifications.

The input file to the program contain the following information about the model input with a 6(I5) format:

NOLC - number of load cases

NODV - number of design variables

ISNOLC - starting number for derivative load cases (eg. NOLC+100)

JOINTS - number of joints in the model

NDF - number of degrees of freedom per joint (from START card)

NOEL - number of different elements

After this card there are two cards for each different element (2\*NOEL cards). The first card is input with a (1x,A3,1x,I3,1x,I3) format, while the second has a format (2I13) These cards contain the following information about the model:

CARD NUMBER 1

EL - names of elements containig design variables

NSECT - last section number used for each design variable

NODVPE - number of design variables per element

CARD NUMBER 2

NOOFDV - location of the design variables in the TAB cards

there are NODVPE of these per card

(eg. J0 is the 6th location on a DSY card)

#### CREATING PROBLEM-DEPENDENT CODE FOR PROSS-2

Two problem-dependent subroutines for the front and end processor must be created for PROSS-2. A third for chain differentiation may be necessary if Option 2.3 is to be used. For these subroutines, blank common

is typically used as a work area. This space is set up with the statement

```
COMMON KORE,KEVEN,A(1)
```

The first two words of blank common, KORE and KEVEN, are reserved for SPAR usage. The array, A(1) determines the address of the first word in blank common. This method uses A as a dynamic array because the size is determined by the field length.

```
LENGTH = FIELD LENGTH - FIRST WORD OF BLANK COMMON + 1
```

When using blank common in this fashion, the STATIC option must be used on the CDC FORTRAN compiler to keep blank common at the upper end of the memory space. If the user does not desire to use the STATIC option, but still wishes to use blank common as a working space, the array in blank common can be given a large dimension. Whichever method is selected, the user can then set up a series of indices into the array. These act as pseudo arrays for referencing the data. For example

```
IAX = 1  
IAG = IAX + NEL  
IRFL = IAG + NCON
```

This example sets an index, IAX, for a pseudo array of design variables and an index, IAG, for a pseudo array of constraints. IAX is the starting point of the design variables in the blank common array and NEL locations (number of elements) are reserved in the array for the design variables. IAG is the starting point of the constraints in the same array and NCON locations (number of constraints) are reserved in the array for the constraints.

It is not required for the user to use the blank common area as a working space. Regular arrays can be set up in the problem-dependent code



or a combination of blank common and arrays can be used. The drawback to using arrays is that they will probably have to be changed in size if the problem changes significantly, whereas the blank common indices changes are dependent on the input data.

These subroutines interface with the SPAR data libraries by using the SPAR data handling utilities (ref. 9). If only one block of data is being used at a time then only the DAL utility is required to input and output data to and from the library. If more than one block is used to store the data, then the RIO utility must be used in conjunction with DAL. Other utilities are available for the user and can be used if desired.

#### Front Processor

The front processor must have the name FPDRIV because that is the the name of the subroutine called by processor OPTD. If the user desires to have more than one front processor for use with multiple problems then FPDRIV can be used to call each individual front processor and the user can set up a SPAR data set to determine which front processor is to be used. This data set should be included in the non-repeatable runstream along with other initialization data sets described previously. The purpose of the front processor is to use the design variables output from CONMIN to update the section properties found in SPAR data sets. SPAR has different data sets containing the section properties for different elements. For example, BA BTAB 2 9 contains the section properties for the beam elements. To determine the location of a particular section property within a particular data set consult reference 10.

The front processor obtains the design variables from data set DESV

CNMN 1 1. If the front processor needs certain data that is problem dependent, then the user can create a data set similar to the FPRC CONS 1 1 data set in the discussion of the non-repeatable runstream. After the code has been developed to use the design variables to change the section properties, then call DAL to update the element BTAB data sets being change. It is easier to set up the code to change one set of elements before moving to the next set.

#### End Processor

The driver for the end processor, ENDP, calls subroutine EPDRVNG when no analytical gradients (Option 1.1, 2.2) are needed and calls subroutine EPDRVG when analytical gradients are needed (Option 2.3). In addition, ENDP may call subroutine DRVS to compute stress and stress derivatives from forces and moments, and derivatives of forces and moments if they are needed by Option 2.3. EPDRVNG and EPDRVG are used instead of only one call to facilitate going from one option to another. The purpose of the end processor is to convert data from the repeatable analysis to a form suitable for input into CONMIN. Each of the above subroutines will be discussed separately.

EPDRVNG is very simple to code. It requires the following data sets as input:

EPRC CONS 1 1 (optional)  
DESV CNMN 1 1  
OBJF AUS 1 1  
STRS Exy 1 1  
CHEK DATA 1 1

EPRC contains constants that may be used in the end processor. Examples of the constants are the allowable stresses for each different element and the number of elements for each different kind. This data set is optional and is only used by the end processor, therefore the name can be whatever the user desires when it is initialized in the non-repeatable runstream. DESV contains the design variables. OBJF contains the objective function as it is calculated in the repeatable runstream (therefore the data set name OBJF AUS 1 1 must be used in the runstream). STRS Exy 1 1 are data sets containing the stresses. The xy in Exy is the number of the element, for example E23 is for the rod elements. CHEK is used in conjunction with the following code in any end processor independently of whether analytical gradients are needed are not. This code tests the value in CHEK and determines if another iteration is needed are not. If another iteration is needed (CHEK=1) then the INPUT file is rewound, if not (CHEK=0) then a RETURN is executed.

```
CALL DAL(1,11,CHEK,0,3,KADR,IERR,NWDS,NE,LB,ITYPE,  
1 4HCHEK,4HDATA,1,1)  
IF(CHEK.EQ.0.) RETURN  
REWIND 5  
RETURN
```

The end processor computes the constraints from the stresses. Two data sets are used for output from the end processor and input to CONMIN. These data sets are

```
G    CNMN 1 1  
OBJ  CNMN 1 1
```

where G contains the constraints and OBJ CNMN contains the objective function.

EPDRVG is more complex than EPDRVNG because in addition to the constraints and objective function, their gradients must also be computed. The following input data sets used in EPDRVG are identical to those used in EPDRVNG

```
EPRC CONS 1 1 (optional)
DESV CNMN 1 1
OBJF AUS 1 1
STRS Exy 1 1
CHEK DATA 1 1
```

In addition, there are other data sets required for input into EPDRVG. These data sets are

```
INFO LOAD 0 0
BLKI CNMN 2 0
DSTR Exy m n
```

There are four pieces of data the user will probably need from the INFO data set. These are the number of load cases, the number of design variables, the number of different types of elements, and a number specifying where the numbering for load cases for the derivatives is to start (this number is typically the number of load cases plus 100). The only value needed from the BLKI data set is the number of constraints which will be the second value. DSTR Exy m n contains the stress derivatives. The value of m is dependent on the load case number for the derivatives, while the value of n is dependent on the number of the design variable. Both are stored in the INFO data set. The following output data

sets are almost identical (the 0 0 following OBJ CNMN is the only difference) to those found in EPDRVNG

G CNMN 1 1

OBJ CNMN 0 0

In addition, the following data sets are also output by EPDRVNG

OBJI CNMN 0 0

BLOK CNMN 1 1

OBJI contains an initial objective function used for iterating within CONMIN. BLOK contains the following block of data. The order of the data is important and should not be adjusted. It is easiest to store these values using the array in blank common.

1. The initial values of the design variables.
2. The initial value of the constraints.
3. The gradient of the objective function.
4. The gradients of the constraints.

A large portion of the DRVS subroutine should be adaptable to any problem. DRVS calls subroutine BMSTRS. DRVS uses the following data sets which are also used as input by EPDRVNG and FPDRIV

INFO LOAD 0 0

EPRC CONS 1 1

FPRC CONS 1 1

DRVS also uses the following data sets as input

ELTS NAME 0 0

FAMS Exy 1 1

DFAM Exy m n

ELTS contains the names of the elements, such as E23. FAMS and DFAM are

the similar to the STRS and DSTR data sets discussed previously. The only difference is that FAMS and DFAM contain forces and moments and their derivatives while STRS and DSTR contain stresses and their derivatives. The formats of the data sets are identical. DRVS is only used to convert the data sets for elements where forces and moments are computed in the repeatable analysis, for example E21 elements. Data sets for other elements are not used in this subroutine. Data is read in from the FAMS data set and subroutine BMSTRS is called to convert this data to stresses. BMSTRS is dependent upon the elements being used. In BMSTRS, certain data (such as the forces and moments, moments of inertia, and Y values) is stored from the FAMS data set. Reference 10 can be used to determine where this data is stored with respect to a particular data set. The stresses that are calculated are then stored in the STRS Exy 1 1 data set which is used by EPDRVG. The data sets from DFAM (one data set for each set of elements) are then read and BMSTRS is called to compute the stress derivatives in a similar fashion to the way the stresses were computed. The stress derivatives are then stored on data sets DSTR Exy m n which is then used in EPDRVG. Thus the output data sets from DRVS are

STRS Exy 1 1

DSTR Exy m n

#### Chain Differentiation

This subroutine, DKDVSUB, is called from the new processor DKDV. The name DKDVSUB must be used. The purpose of this subroutine is to compute the factors for a chain differentiation to find the derivatives of the stiffness and mass matrices with respect to a design variable when the

design variable has more than one contributing factor. The actual chain differentiation takes place in the AUS portion of the repeatable runstream created by the GENRS program. An example of a factor that might be computed in this subroutine is DA/DV, the derivative of the cross-sectional area of a beam with respect to the design variable. Data sets that are input to this subroutine depend upon the problem. Typically, the following two data sets are input

FPRC CONS 1 1

DESV CNMN 1 1

FPRC contains the constant used in the front processor and DESV contains the design variables. After the information in these data sets is used to compute the derivatives, the derivatives are written to data set FACT DKDV 1 1. This name must be used for the data set because the modified AUS accesses the data set by this name.

#### CREATING AN ABSOLUTE EXECUTABLE FOR PROSS-2

The following steps (figure 4) are used to create a new absolute executable for SPAR adding the modified SPAR code, the problem-independent code, and the problem-dependent code to the existing SPAR code. The example is for use on the CDC computer, but similar steps can be taken for any other computer. The file names in parentheses correspond to the listing below. To create the absolute executable the user will need six files containing the relocatable code for SPAR (NRL15EG), the SPAR COM decks (COM15EG), the modified SPAR routines (NPCHNGB), the problem-independent routines (NPNDPBB), the problem-dependent routines (NPDEPB), and CONMIN (CONMINB).

1. The COPYL command is used to replace existing SPAR routines with the modified routines with the results placed on a temporary file, NPTEMP.

2. NPTEMP contains 460 records (one for each routine). These 460 records are copied to file NEWPRS using the COPYBR command. The problem-independent routines are then written to the NEWPRS file from NPNDPBB using COPYBF. COPYBF is used here to place an EOF mark. Once this NEWPRS file has been created it can be saved on disk so that steps 1. and 2. will not have to be executed again.

3. Libraries are created for the COM decks, CONMIN, and the problem-dependent routines using the LIBGEN command. These libraries are used to satisfy any unsatisfied externals with the LDSET command. LDSET is also used to preset core to zero and store the map listing on file NPMAP.

4. The LOAD command with the NOGO command is used to create the absolute executable from file NEWPRS. The absolute code is stored on files named in the overlay cards, SPAR and DCU. The REPLACE command is used to store the files on a disk for future access. The stored files are called NPSPAR and NPDCU.

```
GET,CONMINB,NPNDPBB,NPDEPB,NPCHANGB.
```

```
GET,COM15EG,NRL15EG.
```

```
COPYL,NRL15EG,NPCHANGB,NPTEMP,,R.
```

```
REWIND,NPTEMPB.
```

```
COPYBR,NPTEMP,NEWPRS,460.
```

```
COPYBF,NPNDPBB,NEWPRS.
```

```
SAVE,NEWPRS.
```

```
REWIND,NEWPRS.
```

```
LIBGEN,F=COM15EG,P=COMLIB.
```



LIBGEN,F=CONMINB,P=CONLIB.  
LIBGEN,F=NPDEPB,P=DEPLIB.  
RFL,106000.  
REDUCE,-.  
LDSET,LIB=COMLIB/CONLIB/DEPLIB.  
LDSET,PRESET=ZERO,MAP=SBEX/NPMAP.  
LOAD,NEWPRS.  
NOGO.  
REPLACE,SPAR=NPPAR,DCU=NPDCU.

#### USE OF OTHER OPTIMIZATION AND ANALYSIS CODES

PROSSS was originally developed to be flexible so that any optimization and analysis codes could be combined. Although this particular implementation is limited to CONMIN for optimization and SPAR for analysis, the user can adapt the concepts to other codes. If an optimization code other than CONMIN is desired, The user can write a driver similar to the problem-independent subroutine CNMDRV1. This routine should initialize all variables by reading certain data sets and call the optimization program. The optimization program will probably appear as a "black box" to the user, thus it can be added to the system as a library to satisfy any unsatisfied externals in a manner similar to CONMIN for this system.

Because the entire system is dependent on the analysis code for this implementation, the use of a different analysis code poses a more challenging problem. To convert to a new analysis program, the user must first determine what drives the program. In the case of SPAR, it is driven

by runstreams calling different processors and is therefore very modularized. If the desired analysis code is also in this format, then it is only a matter of adding new processors to handle the optimization and front and end processors. Since all of PROSSS-2 is in FORTRAN the primary difference in using a new analysis program for these processors is in the way the processors interface with the data base supplied by the program. The majority of the remainder of the code should be used intact. Another primary task is for the user to determine how to initialize the variables for a problem. For this implementation, this was handled through the SPAR runstreams in the non-repeatable part. The final task is for the user to develop a program or programs that will generate whatever is needed in the of code or input to handle the analytical gradients for option 2.3. This is not a trivial task. Program GENRS and subroutines DKDVSUB and DRVS can be used as models.

## REFERENCES

1. Sobieszczanski-Sobieski, J.; and Bhat, R. B., "Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled by a Computer Operating System." A Collection of Technical Papers on Structures - AIAA/ASME/ASCE/AHS 20th SDM Conference , April 1979, pp. 20-71, AIAA Paper No. 79-0723.
2. Rogers, J. L., Jr., Sobieszczanski-Sobieski, J., Bhat, R. B., "An Implementation of the Programming Structural Synthesis System (PROSSS)", NASA TM 83180, December 1981.
3. Whetstone, W. D., SPAR Structural Analysis System Reference Manual - System Level 13A. Volume I: Program Execution. NASA CR-158970-1, 1978.
4. Vanderplaats, G. N., CONMIN - A Fortran Program for Constrained Function Minimization. User's Manual. NASA TM X-62282, 1973.
5. NOS Version 1 Reference Manual - Volume 1. Publ. No. 60435400, Control Data Corp., September 1979.
6. Rogers, J. L., Jr., Dovi, A. R., and Riley, K. M., "Distributing Structural Optimization Software Between a Mainframe and a Minicomputer", Presented at the Second International Conference and Exhibition on Engineering Software, London, England, March 24-26, 1981. Proceedings entitled Engineering Software II , Hobbs The Printers, Southampton, England, 1981, pp. 400-415.
7. Rogers, J. L., Jr., "An Implementation of the Distributed Programming Structural Synthesis System (PROSSS)", NASA TM 83253, December 1981.
8. Whetstone, W. D., "EISI-EAL: Engineering Analysis Language", Proceedings of the Second Conference on Computing in Civil Engineering ,

ASCE, 1980, pp. 276-285.

9. Giles, G. L.; and Haftka, R. H., SPAR Data Handling Utilities. NASA TM 78701, 1978.

10. Cunningham, S. W., "SPAR Data Set Contents", NASA TM 83181, October 1981.

-----  
Routine: OPTD

Purpose: Controls flow through option

Option: All

Data Sets		I/O	Purpose
OPT NO	1 1	I	Option number
CHEK EVAL	1 1	I/O	Flag for performing finite differences (Opt. 2.2)
PASS CNMN	1 1	I/O	Iteration number through system

-----

Routine: CNMNDRV1

Purpose: Driver program for CONMIN

Option: 1.1

Data Sets		I/O	Purpose
BLKR CNMN	1 1	I/O	Real data for CONMIN common block CNMN1
BLKI CNMN	1 1	I/O	Integer data for CONMIN common block CNMN1
PASS CNMN	1 1	I/O	Iteration number through system
VLB CNMN	1 1	I	Lower bounds for design variables
VUB CNMN	1 1	I	Upper bounds for design variables
ISC CNMN	1 0	I	Linear constraint identifiers
DESV CNMN	1 1	I/O	Design variables
*CSAV CNMN	1 1	I/O	Data for CONMIN common block CONSAV
*WORK CNMN	1 1	I/O	Data for blank common
*G CNMN	1 1	I	Constraint data from end processor
*A CNMN	1 1	I/O	Gradients of active or violated constraints
*IC CNMN	1 1	I/O	Active or violated constraints
*OBJ CNMN	1 1	I	Objective function from end processor
CHEK DATA	1 1	0	System terminator for convergence

-----

Routine: CNMDRV2

Purpose: Driver program for CONMIN

Option: 2.2, 2.3

Data Sets		I/O	Purpose
CNMN PARM	2 0	I	CONMIN parameters
BLKR CNMN	1 1	I	Real data for CONMIN common block CNMN1
BLKI CNMN	2 0	I	Integer data for CONMIN common block CNMN1
VLB CNMN	1 1	I	Lower bounds for design variables
VUB CNMN	1 1	I	Upper bounds for design variables
ISC CNMN	2 0	I	Linear constraint identifiers
*BLOK CNMN	1 1	I	Initial design variables and constraints with gradients of constraints and objective function from end processor
DESV CNMN	1 1	I/O	Design variables
*G CNMN	1 1	I	Constraints from end processor
*OBJ CNMN	0 0	I	Objective function from end processor
*OBJI CNMN	0 0	I	Initial objective function from end processor
TEST OBJ	1 1	I/O	Convergence criteria for CONMIN
CHEK DATA	1 1	0	System terminator for convergence

-----

Routine: EVALNG

Purpose: Compute finite difference gradients

Option: 2.2

Data Sets		I/O	Purpose
EVAL PARM	1 1	I	Step size change in design variables
BLKR CNMN	1 1	I	Real data for CONMIN common block CNMN1
BLKI CNMN	2 0	I	Integer data for CONMIN common block CNMN1
PASS CNMN	1 1	I/O	Iteration number through system
DESV CNMN	1 1	I/O	Design Variables
*BLOK CNMN	1 1	I/O	Initial design variables and constraints with gradients of constraints and objective function from end processor
*OBJ CNMN	0 0	I/O	Objective function
*OBJI CNMN	0 0	I/O	Initial objective function
*G CNMN	1 1	I/O	Constraints
*ICNT CNMN	1 1	I/O	Counter for finite difference iterations
CHEK EVAL	1 1	0	Flag for performing finite differences

Routine: ENDP

Purpose: Driver for calling end processor

Option: All

Data Sets		I/O	Purpose
OPT NO	1 1	I	Option number
DRV CHEK	1 1	I	Flag for converting forces and moments and their derivatives to stresses and their derivatives (Option 2.3)

Routine: DKDV

Purpose: Compute factors used in chain differentiation

Option: 2.3

Data Sets: None

Table 1. - Problem-independent routines

I = Input O = Output I/O = Input/Output

\* = data set generated within system, not by user

DATA SET NAME	NJ	PURPOSE
PASS CNMN 1 1	1	Initializes the pass through CONMIN to 1
CHEK DATA 1 1	1	Initializes system terminator to 1
CHEK EVAL 1 1	1	Initializes flag for using finite differences to 1 (Opt 2.2)
OPT NO 1 1	1	Sets option number (ex. 23. for option 2.3)
*FPRC CONS 1 1	TBD	Contains constants needed for front processor
*EPRC CONS 1 1	TBD	Contains constants needed for end processor
VLB CNMN 1 1	TBD	Sets lower bounds used in CONMIN
VUB CNMN 1 1	TBD	Sets upper bounds used in CONMIN
DESV CNMN 1 1	TBD	Sets starting values for design variables
BLKR CNMN 1 1	12	Contains real parameters for CONMIN (Note: DABFUN multiplied by .1 in CONMIN driver)
BLKI CNMN 1 1	15	Contains integer parameters for CONMIN (Opt 1.1)
BLKI CNMN 2 0	15	Contains integer parameters for CONMIN (Opt 2.2, 2.3)
TEST OBJ 1 1	4	Contains convergence criteria for CONMIN (Opt. 2.2, 2.3) (Fourth value is the percentage for change first three numbers are pseudo objective functions which are updated in each iteration, ex. 10.; 50.; 10.; .05)
CNMN PARM 2 0	4	Contains special values needed (Opt 2.2, 2.3) 1st-maximum number of active constraints 2nd-lower bound on change in design variable 3rd-upper bound on change in design variable 4th-passes through limited analysis
EVAL PARM 1 1	1	Sets step size for change in design variable (Opt 2.2)
DRV CHEK 1 1	1	Checks to see if forces and moments and their derivatives need to be converted to stresses and their derivatives 0-no, 1-yes (Opt 2.3)
*INFO LOAD 0 0	TBD	Contains information used in subroutine DRVS (Opt 2.3)
ISC CNMN 1 0	TBD	Contains linear constraint identifiers for CONMIN (Opt 1.1) Format for this data set is different from others as follows TABLE(NI=1,NJ=xxx); ISC CNMN 1 0 I=1 DDATA=0. J=1,xxx 0.
ISC CNMN 2 0	TBD	Contains linear constraint identifiers for CONMIN (Opt 2.2, 2.3) The format is the same as for ISC CNMN 1 0 except a 1. should appear after J=1,xxx instead of a 0. DDATA remains the same.

TABLE 2. - Data sets containing initialized variables to be used in non-repeatable part with the AUS TABLE subprocessor

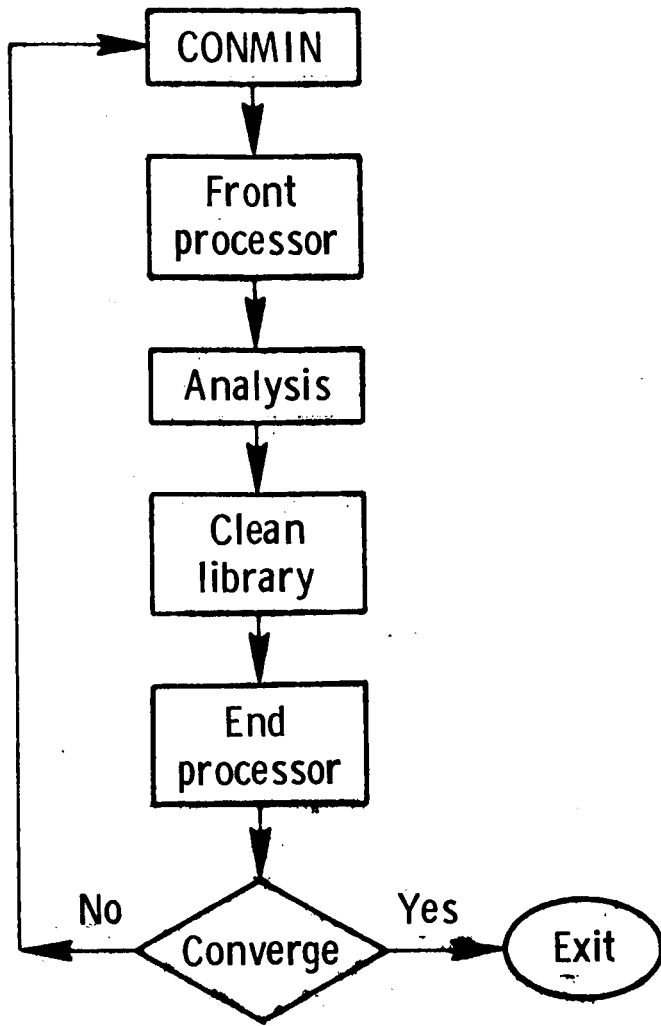


FIGURE 1. - FLOW CHART FOR OPTION 1.1



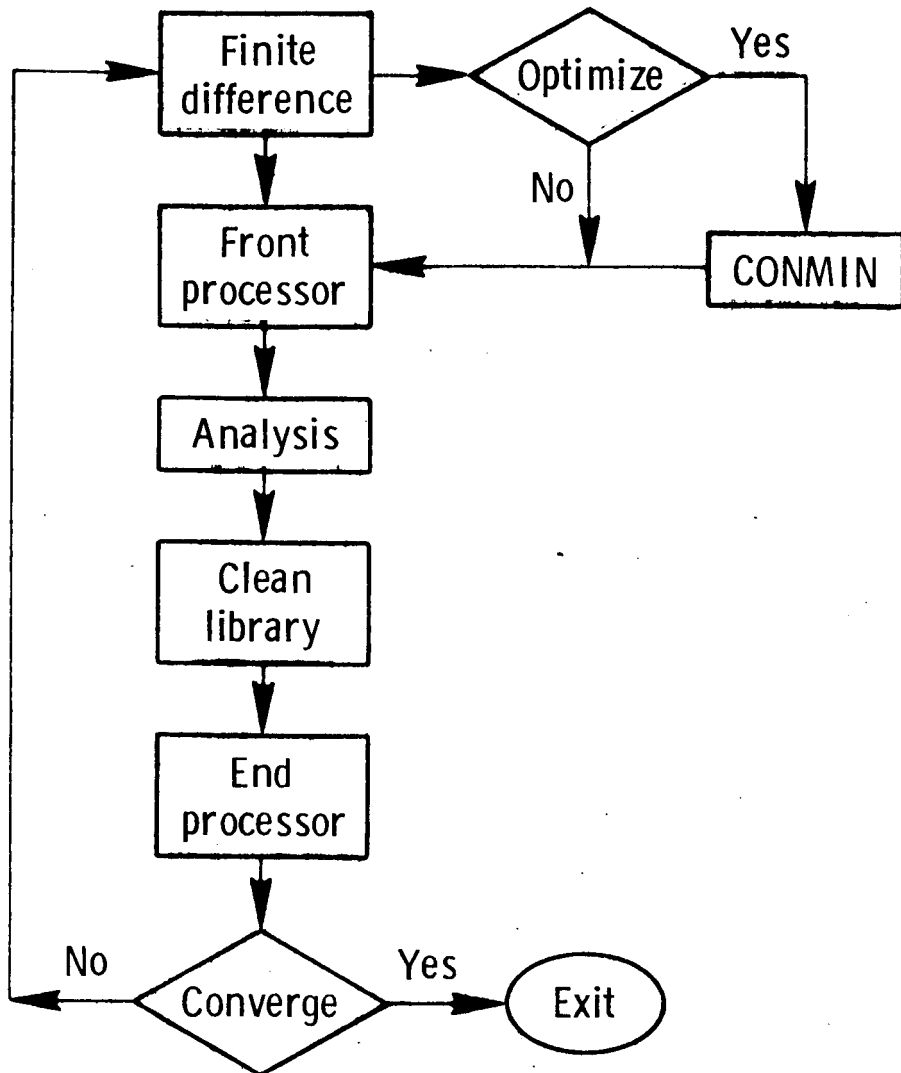


FIGURE 2. - FLOW CHART FOR OPTION 2.2

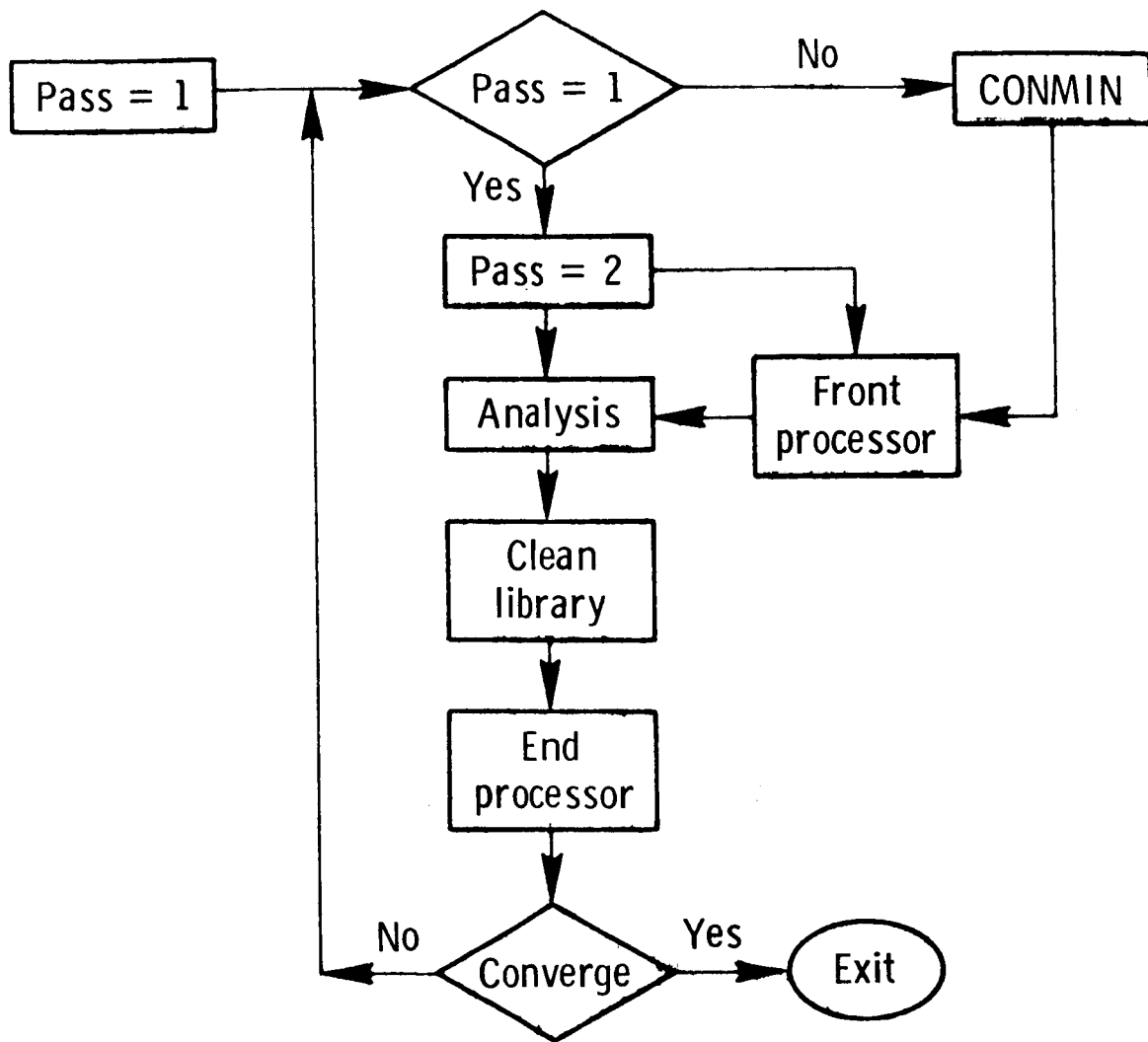


FIGURE 3. - FLOW CHART FOR OPTION 2.3

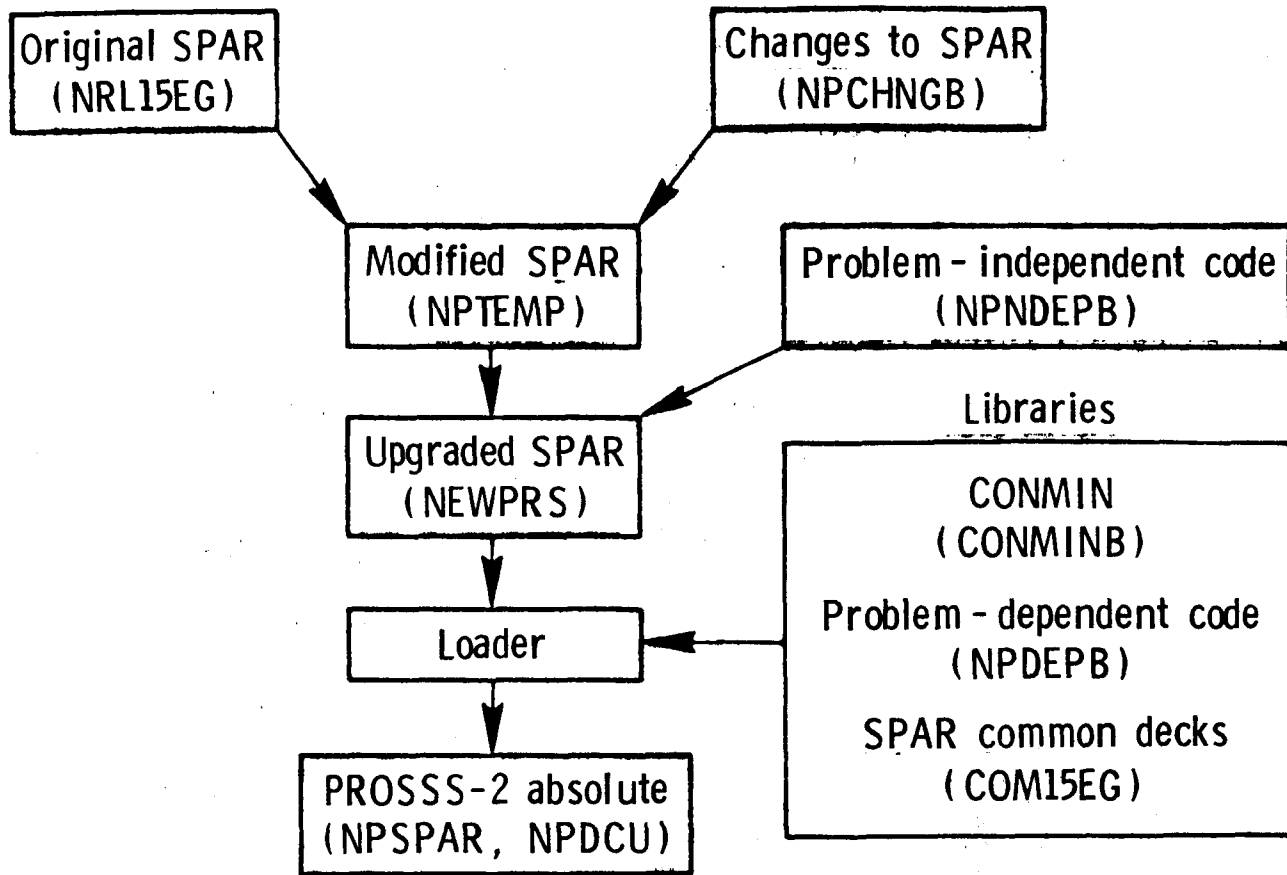



FIGURE 4. - CREATING PROSS-2

1. Report No. NASA TM-86326		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle A New Implementation of the Programming System for Structural Synthesis (PROSS-2)				5. Report Date November 1984	
				6. Performing Organization Code 505-33-53-12	
7. Author(s) James L. Rogers, Jr.				8. Performing Organization Report No.	
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665				10. Work Unit No.	
				11. Contract or Grant No.	
12. Sponsoring Agency Name and Address National Aeronautics & Space Administration Washington, DC 20546				13. Type of Report and Period Covered Technical Memorandum	
				14. Sponsoring Agency Code	
15. Supplementary Notes					
16. Abstract This new implementation of the PROgramming System for Structural Synthesis (PROSS-2) combines a general-purpose finite element computer program for structural analysis, a state-of-the-art optimization program, and several user-supplied, problem-dependent computer programs. The results are flexibility of the optimization procedure, organization, and versatility of the formulation of constraints and design variables. The analysis-optimization process results in a minimized objective function, typically the mass. The analysis and optimization programs are executed repeatedly by looping through the system until the process is stopped by a user-defined termination criterion. However, some of the analysis, such as model definition, need only be one time and the results are saved for future use. The user must write some small, simple FORTRAN programs to interface between the analysis and optimization programs. One of these programs, the front processor, converts the design variables output from the optimizer into a suitable format for input into the analyzer. Another, the end processor, retrieves the behavior variables and, optionally, their gradients from the analysis program and evaluates the objective function and constraints and optionally their gradients. These quantities are output in a format suitable for input into the optimizer. These user-supplied programs are problem-dependent because they depend primarily upon which finite elements are being used in the model. PROSS-2 differs from the original PROSS in that the optimizer and front and end processors have been integrated into the finite element computer program. This was done to reduce the complexity and increase portability of the system, and to take advantage of the data handling features found in the finite element program.					
17. Key Words (Suggested by Author(s)) Optimization, finite element analysis			18. Distribution Statement  Subject Category 61		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of Pages 35	22. Price