LPSPEC: A Language for Representing Linear Programs

Pai-chun Ma
Graduate School of Business Administration
New York University
New York, New York

Frederic H. Murphy
School of Business
Temple University
Philadelphia, Pennsylvania

and

Edward A. Stohr
Graduate School of Business Administration
New York University
New York, New York

October 1986

Center for Research on Information Systems
Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #139

GBA #86-104

This work was carried out as part of a jointly-defined research study on expert systems with the IBM Corporation.

Table of Contents

List of Figures

1. Introduction

Linear programming (LP) has had many successful applications in production planning, logistics, finance and marketing [13]. An LP model involves the maximization (or minimization) of a linear objective function subject to the satisfaction of linear equations and inequalities [2]:

$$\begin{array}{ll} \text{Max} & cx \\ \text{Subject to} & \\ & Ax \leq b \\ & x \geq 0 \end{array}$$

Figure 1-1: Linear Program in matrix notation

Significant progress has been made in the development of algorithms and software for solving large LPs where the matrix, A, may have thousands of rows and columns. However, building such large scale models for real world problems is a time-consuming and error-prone task. The objective of the research described in this paper and in [6], [11] and [7], is to develop an intelligent software system that will help expert and non-expert users formulate LPs.

The process of formulating and solving an LP has five stages: problem investigation, model formulation, data binding, algorithmic solution and analysis of the solution. Various types of knowledge and reasoning processes are used at each stage. The problem investigation and model formulation stages are currently accomplished by human experts because of the high degree of ambiguity and complexity involved. By data binding we mean the assignment of data values to the symbols in the model statement. Modeling languages such as OMNI, [12] and GAMS, [8] are helpful, but this stage is still quite difficult and error-prone. Advances in computer technology and optimization theory have made the fourth stage, solving the LP, relatively routine even for very large problems. Finally, a number of systems now exist to help users analyze the results of LP models [4].

Our research is directed towards the relatively neglected formulation stage of LP modeling. The LPFORM system is based on the following design principles:

1. Change the representation used for stating an LP away from the traditional mathematical or tableau-oriented approaches and towards a more visual form, [6].

2. Support a number of problem-solving strategies to reduce the cognitive complexity of the task. Examples include hierarchical decomposition of the problem, inheritance of properties from more general objects, and a non-procedural approach that allows users to define their problems piece-by-piece in an arbitrary order.

3. Allow large models to be built from combinations of smaller models. Provide a model library with standard models and allow users to add their own model templates.

4. Provide many different problem representations, each of which is suitable for a different user task.

5. Check for consistency at all stages in the development of the model. In current systems, one has to run the model to discover errors and then undertake a difficult investigation to determine the cause. This is done by examining the structure of the LP tableau either manually, or preferably, using sophisticated software [4].

The examples in Section 3 and 4 illustrate some of these ideas.

The purpose of this paper is to describe the current interface to the LPFORM system which consists of a command language, LPSPEC, rather than the graphics interface mentioned above and discussed in detail in [6]. Section 2 describes the role of LPSPEC in the LPFORM system. Section 3 shows a complete example of the use of LPFORM to formulate and execute a small LP. Section 4 provides more examples of the use of LPSPEC to define LP models. Section 5 discusses some technical features of LPSPEC, together with the online "Help" facility. Finally, the three Appendices constitute a reference manual for the system. These describe respectively, the LPSPEC language, the LPFORM interactive commands and the standard model templates provided by the system.

2. The Role of LPSPEC in the LPFORM System

Figure 2-1 shows the architecture of the LPFORM system. The design resembles the stages for formulating and solving LP problems given above.

```
    ______________________________________________________________________________
    | Integrated LP                                                              |
    | System                          ANALYZER                                   |
    |    .----------------------------->[ ]<-------------------------------.     |
    |    |                                                                 |     |
    |    |                                                                 |     |
    |    |                                                                 |     |
    |    |      LP FORMULATOR          TABLEAU                LP           |     |
U   |    v        (LPFORM)             GENERATOR   LP         SOLVER       v     |   U
S <--> Graphic ---->[ ]------> Algebraic -->[ ]----> Matrix ----->[ ]----> Solution --> S
E   |  Dialogue       ^        Statements   ^       Statements                   |   E
R   |                 |                     |       ("Case")                     |   R
    |               Data                   Data                                  |
    |               Knowledge              Values                                |
    |                 |                     |                                    |
    |                 '--------->{ }<--------'                                   |
    |                    DATA BASE SYSTEM                                        |
    |                                              [ ] : Process                 |
    |                                              { } : Data or Knowledge Base  |
    |____________________________________________________________________________|
```

Figure 2-1: Integrated LP System Diagram

Currently, the LP Generator is composed of three sub-systems loosely coupled via communicating files:

1. The LPFORM system, ([10], [5]) which translates from a graphics representation to an algebraic representation.

2. A Tableau Generator, MPGEN [15], which is similar in function to the GAMS system [8] (both take an algebraic approach).

3. IBM's MPSX system for solving linear and integer mathematical programs [9] (or the LINDO system [13]).

Another two systems will be added later:

1. IBM's SQL database management system (DBMS) [1]. This will provide meta information on the structure and contents of the database to the LPFORM system which will generate the SQL queries [1] to obtain the data needed by the tableau generator.

2. A tableau solution analyzer (ANALYZE [4]) which will analyze the solution and provide useful management reports.

The user first defines the problem using the graphics interface. This interaction is then translated into statements in the LPSPEC language described in this paper. LPSPEC is the first of five internal representations that can be viewed by the user (see Figure 2-2).

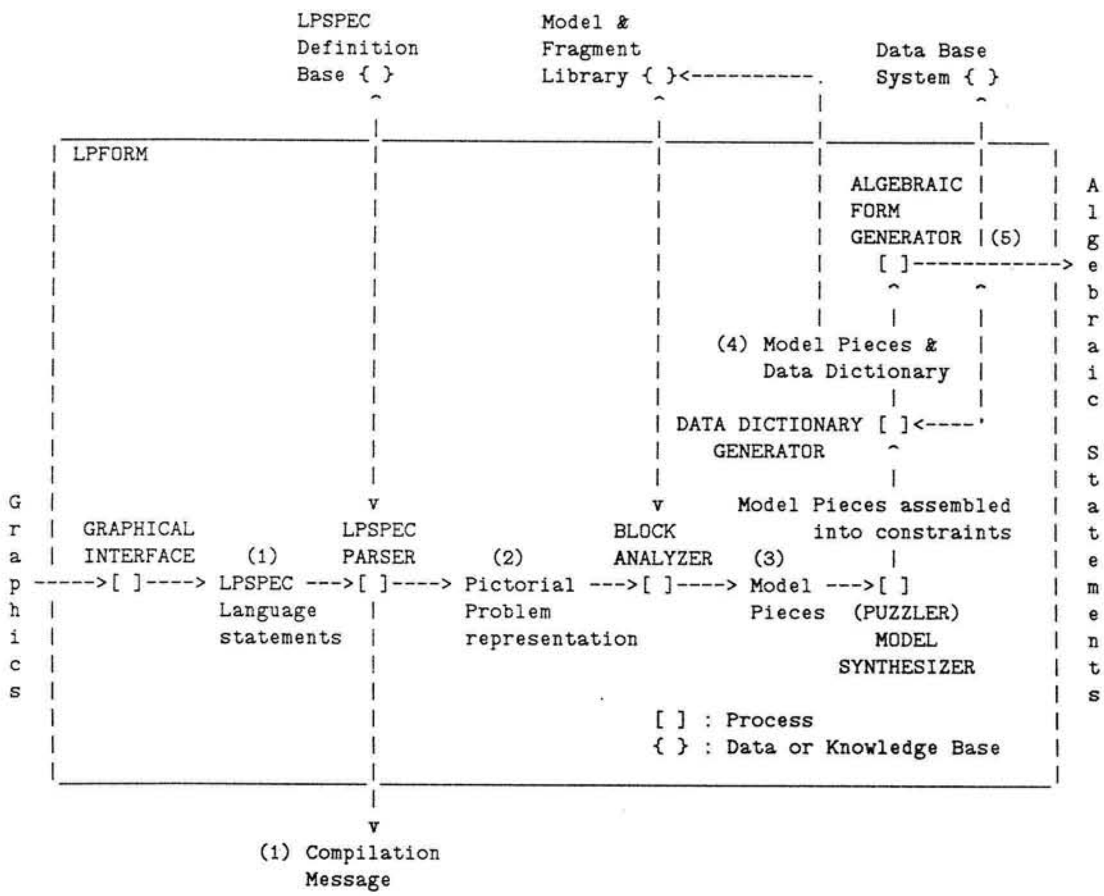

Figure 2-2: LPFORM System Diagram

The second representation involves the internal data structures that record the network structure of the problem. The third consists of the "model-pieces" that are generated by the inference rules in LPFORM. These will be needed in the algebraic statement of the problem. Each model-piece represents a term in a constraint or objective function expressed in summation notation. For example:

$$\sum_{j} a_{i,j} X_{i,j}$$

In the fourth representation, the model-pieces are assembled into a 'picture' of the tableau obtained by arranging the algebraic terms in rows and columns. This is accompanied by a model data dictionary defining all variables, coefficients and indices. This representation is useful both for on-line checking and as permanent documentation of the model. The fifth representation is written in the input language of whatever tableau generator is being used. It is obtained simply by reformatting the internal tableau.

The development work so far has involved the construction of an intelligent system to translate between representations (2) and (4). The LPSPEC language to be described in this paper is a formal, command-oriented language intended ultimately to be an internal representation only. At the present stage of development of the prototype however it is the only user interface. After the graphics interface has been developed, LPSPEC will remain: (1) as an intermediate language generated by the interface (see Figure 2-2) and (2) as an alternate input medium.

The current version of LPSPEC is preliminary in nature. It is capable of representing only a subset of LP models relating to production planning and logistics. Much work will be required to determine its completeness as a language for defining LPs.

LPSPEC is a collection of declarative statements rather than a programming language. Each statement is a collection of data items that result from an interaction with the user in the graphics interface. The screen shown in Figure 2-3 shows the relationship. Each of the menu items on the right of the screen (REL though OPT) has a corresponding LPSPEC statement. There are two classes of commands. The "Data" commands in the upper right of the screen declare the relations, tables, parameters and sets that will be required. The "Structure" commands in the lower right of the screen, define the structure of the LP as illustrated below.

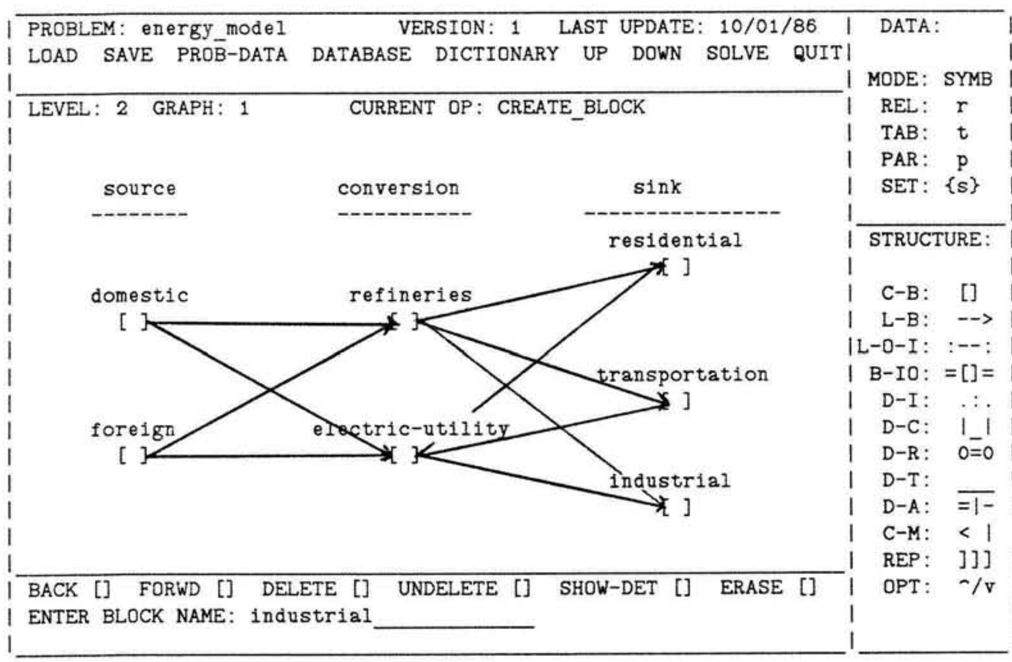

Figure 2-3: Create-Block Screen

Users will interact with the graphics system to build network representations of their models by pointing to one of these commands and then to the position in the center of the screen upon which the command is to act. In LPFORM, a "block" represents an arbitrary collection of activities at a point in space or time. The user will be able to place an icon representing a "block" in the center of the screen by selecting "C-B" on the right and then indicating the desired location in the center of the screen. The user will then be asked to supply relevant descriptive information and the corresponding CREATE-BLOCK LPSPEC command will be generated by the graphics subsystem.

There are two formulation modes in LPSPEC. In "Symbolic Mode", no attempt is made by LPFORM to link the symbols in the problem statement to their values. In other words, the Data Commands in Figure 2.2 are not used (except for SET). In "Data Mode", LPFORM links the algebraic symbols to their values in database relations, tables and parameter lists. Only Symbolic Mode has been implemented so far.

The design proposed for the graphics interface is presented in [6].

3. Using LPSPEC: A Complete Example

In this section we show how to define and run a simple LP problem in full detail. We define the problem to be solved, explain the LPSPEC formulation and then give an annotated example showing all steps in the use of LPFORM. There are two ways to run LPFORM. In "file mode", the user enters the LPSPEC statements in an external file and specifies the file name when prompted by the system. In "interactive mode", LPSPEC statements are entered one-at-a-time in response to prompts from the system. The example problem is explained in Section 3.1, solved using the file mode in Section 3.2 and solved interactively in Section 3.3.

3.1. The Transportation Problem

Assume that the user wishes to develop an optimal pattern of distribution between vendors located at New York and Boston and warehouses at Buffalo, Houston, and Denver. There is only a single commodity to be transported. In mathematical notation this transportation problem is:

$$\text{MIN} \sum_{i \in vendor, j \in warehouse} tc_{i,j} X_{i,j}$$

SUBJECT TO

$$\sum_{j \in warehouse} X_{i,j} \leq s_i, \ \forall i \in vendor$$

$$\sum_{i \in vendor} X_{i,j} \geq d_j, \ \forall j \in warehouse$$

Here we have two indices, i and j, representing *vendor* and *warehouse*, a decision variable, $X_{i,j}$, representing the transportation activity between vendors and warehouses, and three coefficients, $tc_{i,j}$, s_i, and d_j, representing the unit transportation cost between i and j, the amount of the commodity supplied by vendor i, and the amount of the commodity demanded by warehouse j. Figure 3-1 provides a graphical view of the problem.

```
Top Level:
                    Vendors --------------------> Warehouses

Second Level:
```

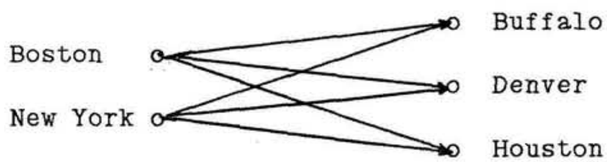

Figure 3-1: Graphical View of the Transportation Problem

This graph could be constructed on a graphics screen as discussed above and in more detail in [6]. The LPSPEC statements that would be generated as a result of this interaction are shown in Figure 3-2.

```
create_block(example1,[vendor,warehouse]).
link_block(space,explicit,[vendor,warehouse],x).
create_block(vendor,[newyork,boston]).
create_block(warehouse,[buffalo,houston,denver]).
optimize(min,example1,cost,symbolic).
```

Figure 3-2: LPSPEC Statements for Transportation Problem

The first CREATE-BLOCK statement defines a 'block' (in this case, "example1") and associates with it its children blocks ("vendor" and "warehouse"). The highest block in the hierarchy automatically becomes the problem name. The LINK-BLOCK statement defines a flow from vendors to warehouses. This completes the representation at the top level in Figure 3-1. The next two CREATE-BLOCK statements define the second level diagram in the figure. Finally, the OPTIMIZE statement records the direction of optimization, the kind of objective function type as "cost" and the mode of the solution as "symbolic". The type of objective function is irrelevant here but is useful when many different templates are to be combined. As explained above, no attempt is made to link the problem statement to data values when the mode is symbolic.

Figure 3-1 is an example of a "generalization" hierarchy, [14]. It is designed to simplify the definition process for the user. Properties defined at higher levels are automatically inherited by lower levels. In this case, the fact that vendors are linked to warehouses in the top level causes links to be generated from all instances of Vendor to all instances of Warehouse in the second level. This is the default mode; arbitrary networks can be defined at any level in the hierarchy by entering the arcs individually, or by specifying an external list where they can be found.

3.2. Running LPFORM in File Mode

We now show how this problem can be formulated using the file-mode and then run through the MPGEN and LINDO systems to obtain a solution. The same problem will be used in the following section to illustrate the interactive use of LPFORM.

The following steps are to be performed:

1. Enter the LPSPEC problem statement using the system Editor.
2. Execute LPFORM system to parse the LPSPEC statement and to write the

formulation for the MPGEN system on an external file.

3. Execute MPGEN, which will allow the user to interactively define the data for the problem (necessary since the mode is "symbolic") and write the MPS form of the problem on another external file for input to LINDO.

4. Execute LINDO to obtain the solution to the problem.

Figure 3-3 illustrates the communication between these systems and gives the file names that are used in the example.

```
LPSPEC                    MPGEN                    MPS
Definition                Definition               Format

transf.x05 --> LPFORM --> exampl.apl --> MPGEN --> lindo.aas --> LINDO --> solution
```

Figure 3-3: Communication between Systems

This process is illustrated below from an actual run of the system. The statements in Figure 3-2 have already been entered into the transf.xo5 file. User responses to prompts by the system are underlined. Explanatory comments are shown in italics.

To execute LPFORM the user types "do lpform" and enters the terminal type in response to the prompt. A menu is displayed.

```
@do lpform
yes
| ?- lpform.

Terminal Type (vt100, heath):  heath

    ***********************************************************************
    *                                                                     *
    *            L P F O R M   V E R S I O N   4 . 1                      *
    *                                                                     *
    *   September 1986                                     Pai-chun Ma    *
    *                                                    (212)505-9392    *
    *---------------------------------------------------------------------*
    *   Type:                                                             *
    *                                                                     *
    *       help.           To obtain an explanation of LPFORM.           *
    *                                                                     *
    *       run_problem.    To formulate an LP problem.                   *
    *                                                                     *
    *       template.       To maintain library of problem templates.     *
    *                                                                     *
    *       lpspec.         To modify Lpspec definitions.                 *
    *                                                                     *
    *       ctrl-z          To exit LPFORM.                               *
    *                                                                     *
```

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

The user types "run_problem." in order to generate the LP formulation and is then asked to give the names of two input files and two output files.

```
| ?- run problem.

------------------ LPFORMulator  version 4.1 -----------------------

Enter the names of files to be used:

   1. File names must be enclosed by two single quotes.
   2. The input of a list (multiple items) must be terminated by a period.
   3. To use system default file names type "d"

        Template file(s)          : .
        LPSPEC file(s)            : 'transf.x05'.
        Template save file        : d
        MPGEN specification file  : d
```

"Template files" are used to load previously defined models that may form part of the specification of the new model. The user types "." to indicate that none are to be used in this run of LPFORM. The LPSPEC file, transf.x05, contains the LPSPEC definition in Figure 3-2.

The user decides to use the system default file names for the "template save file", where the formulated model will be stored for possible reuse and the "MPGEN specification file", which will contain the final algebraic formulation.

```
Define identifiers for indices, variables and coefficients? y
```

The "identifiers" are used in the algebraic formulation. The user will be asked to input them later after the system has determined what is required.

```
Choose problem representations to be displayed:

        (1) LPSPEC compilation, valid and error statements.
        (2) Block reasoning, relevant blocks, flows, activities.
        (3) Model piece construction, a tentative LP formulation.
        (4) Internal tableau, final LP formulation.
        (5) MPGEN specification.

        Checking points? (1,2,3,4,5): 4,5.
```

The user has asked to see the final two representations of the problem. The system now interprets the LPSPEC statements in the input file. If checking point 1. had been selected the system would have displayed each statement as it was parsed.

```
>>>>>> Compiling LPSPEC File: transf.x05 <<<<<<

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

```
*                                                                 *
*          Compilation Statistics for File: transf.x05            *
*                                                                 *
*******************************************************************

Number of lines read: 5
Number of Valid lines:      5
VALID statements saved in file: transf.ok
Number of Error lines:      0
Detail compilation messages in file: transf.msg
Compilation run time:   1.479 seconds

>>>>>> Analyzing Blocks and Flows <<<<<<

>>>>>> Scanning Internal Objects <<<<<<

>>>>>> Consolidating Model: example1 <<<<<<
```

At this point LPFORM has a correct formulation and asks the user to name the variables, indices and coefficients that will appear in the formulation. If this run had been in "data mode" the system would now try to match the symbols it needs with the data items in the declared tables and relations. Identifiers would automatically be generated from the data item names in the tables and the next step would display the automatically generated names and allow the user to override them.

```
Enter the identifiers for INDICES:
    vendor, warehouse.

        vendor: i
        warehouse: j

Enter the identifiers for VARIABLE(S):
  x.

        x: x

Enter the identifiers for COEF:
  obj?*^vendor^warehouse, rhs?*^demand^warehouse,
  rhs?*^supply^vendor.
        obj?*^vendor^warehouse: tc
        rhs?*^demand^warehouse: d
        rhs?*^supply^vendor: s
```

Note that the system displays all of the symbols in each class before requesting the names of individual identifiers. Data coefficients are distinguished internally by their position in the tableau and their associated index sets. In the above interaction, the user has, for example, given the identifier "tc" to an objective function coefficient which will be doubly indexed by vendor and warehouse.

As requested earlier by the user, LPFORM next displays its internal tableau representation immediately followed by the model data dictionary that it has generated.

```
PROBLEM/MODEL/FRAGMENT = example1.

ROW\COL    X(i,j)           RHS
OBJ=       +S{i;j}tc[i;j]  MIN
Use[i]     +S{j}1[i;j]      < +s[i]
Supply[j] +S{i}1[i;j]      > +d[j]
```

In the above representation, the "S" stands for the mathematical "sigma" (Σ) symbol. The indices for the summation are enclosed in the braces and the indices for the coefficients in square brackets. Note that the constraint types, "Use" and "Supply", have been inferred from the problem context and are now displayed for verification by the user.

```
                  * Symbol convention for example1 *

Set Reference:
SYMBOL:   SET NAME:
---------------------------------------------------------------
i         : Vendor
            Meaning: from_block.
j         : Warehouse
            Meaning: to_block.

Activity Reference:
SYMBOL:   ACTIVITY (VARIABLE):
---------------------------------------------------------------
X(i,j)    : X(Vendor,Warehouse)

Coefficient Reference:
SYMBOL:   COEFFICIENT (DATA):
---------------------------------------------------------------
1[i;j]    : 1[Vendor,Warehouse]
s[i]      : Rhs?*^supply^vendor[Vendor]
d[j]      : Rhs?*^demand^warehouse[Warehouse]
tc[i;j]   : Obj?*^vendor^warehouse[Vendor,Warehouse]
```

The model data dictionary displays each index, variable and data coefficient together with information about its role in the model.

```
Generate Output to MPGEN system? (y or n): y
```

Since the above formulation appears to be correct, the user decides to generate the statement for the mathematical programming generator.

```
>>>>>> Generating Output to MPGEN for example1 <<<<<<

* NAME example1
*
* VENDOR = 'newyork,boston'
E VENDOR _ 1 THRU NI _ 2
*
* WAREHOUSE = 'buffalo,houston,denver'
E WAREHOUSE _ 1 THRU NJ _ 3
*
DATA= NI,NJ,S(NI),D(NJ),TC(NIxNJ)
*
```

```
VAR=X(i,j), i in Vendor, j in Warehouse
*
MIN
  S S tc[i;j]X(i,j)
i in Vendor, j in Warehouse
*
FOR i in Vendor
  S X(i,j) < s[i]
j in Warehouse
*
FOR j in Warehouse
  S X(i,j) > d[j]
i in Vendor
*
```

The conventions for representing linear programs in MPGEN are explained in [15]. Lines starting with an asterisk are comments. Lines beginning with an "E" contain executable APL statements. In the above, these are used to generate the Vendor and Warehouse sets. The "DATA=" line declares the data that will be used together with the dimensions of arrays (for example, TC is an NI by NJ array). The "VAR=" line declares variable, x, and assigns it to columns in the tableau. The remainder of the problem statement approximates normal mathematical conventions. Note that "S" has again been used to stand for the "sigma" sign.

```
Save the model template and write the MPGEN problem statement? y

>>>>>> Saving the Model <<<<<<<

            ******************************************************
            *                                                    *
            *  The LP Formulation of Model: example1 is finished!  *
            *                                                    *
            ******************************************************
            Model template saved in file : exampl.mod
            MPGEN statement saved in file: exampl.apl
            Run time = 9.695 seconds

yes
| ?-^z
```

The user now quits LPFORM and executes the MPGEN program.

```
@do mpgen
```

The following statements are executed automatically; the MPGEN system is loaded, the file output by LPFORM is read and the first few statements of the algebraic problem statement are interpreted.

```
@apl

        Loading APLSF.............

terminal..tty

      STARTUP
```

```
---- LATEST VERSION OF MPGEN   7/12/85  ----
1 RECORDS READ FROM FILENM
32 RECORDS READ FROM EXAMPL.APL
-------- PARSING PROBLEM STATEMENT = EXAMPL
*NAME EXAMPLE1
*
* VENDOR = 'NEWYORK,BOSTON'
E VENDOR _ 1 THRU NI _ 2
*
* WAREHOUSE = 'BUFFALO,HOUSTON,DENVER'
E WAREHOUSE _ 1 THRU NJ _ 3
*
DATA= NI,NJ
```

The MPGEN system now prompts the user to interactively input the remaining data for the problem.

```
DATA= S(NI)
  VARIABLE NOT DEFINED.  ENTER VALUES ?  Y
  LENGTH 2
  :      2 3

DATA= D(NJ)
  VARIABLE NOT DEFINED.  ENTER VALUES ?  Y
  LENGTH 3
  :      1 2 2

DATA= TC(NIxNJ)
  VARIABLE NOT DEFINED.  ENTER VALUES ?  Y
  DIMENSIONS 2 3
  ROW 1
  :      2.1 2.5 4.1
  ROW 2
  :      1.8 2.4 0.9
```

MPGEN now interprets the problem statement.

```
*
VAR=X(I,J),
: I IN VENDOR,
: J IN WAREHOUSE
*
MIN
  S S TC[I;J]X(I,J)
I IN VENDOR,
: J IN WAREHOUSE
*
FOR I IN VENDOR
  S X(I,J)
: < S[I]
J IN WAREHOUSE
*
FOR J IN WAREHOUSE
  S X(I,J)
: > D[J]
I IN VENDOR
*

LP TRIPLES (ROW,COL,VALUE) HAVE BEEN FORMED.
TABLEAU SIZE: ROWS = 6  COLS = 7  TRIPLES = 23

STORED VALUES IN FILE FOR:
```

```
    NI NJ S D TC
ERASED VALUES FROM WS FOR:
    NI NJ S D TC
```

The problem statement has been interpreted without error. MPGEN generates the output file to LINDO and returns to the system monitor.

```
--------- START OF GENLINDO ---------------
NAME   EXAMPL.APL   9/23/1986   18:30:32

--------- END OF GENLINDO -------------
```

The user now types "do lindo" to execute the LINDO system and read the file containing the problem statement in the MPS format output by MPGEN. The user then responds to the prompt asking for the direction of optimization and types "go" to solve the linear programming problem.

```
@do lindo
 LINDO (UC 30 APRIL 82)
:rmps lindo.aas
NAME   EXAMPL.APL   9/23/1986   18:30:32

OBJECTIVE ROW FOUND:        1
MAX OR MIN ?
?min
ROWS=       6 VARS=       6 NO. INTEGER VARS=       0
NONZEROES=      23 CONSTRAINT NONZ=      12(     12 ARE +- 1) DENSITY= .548
SMALLEST AND LARGEST ELEMENTS IN ABSOLUTE VALUE=    0.900000        4.10000
NO. < :    2 NO. =:    0 NO. > :    3, OBJ=MIN, GUBS <=    3
SINGLE COLS=     0
:go
    LP OPTIMUM FOUND  AT STEP      4

         OBJECTIVE FUNCTION VALUE

 1)          8.60000000

VARIABLE          VALUE           REDUCED COST
      X11          0.000000           0.200000
      X12          2.000000           0.000000
      X13          0.000000           3.100000
      X21          1.000000           0.000000
      X22          0.000000           0.000000
      X23          2.000000           0.000000


ROW          SLACK OR SURPLUS      DUAL PRICES
   2)              0.000000           0.000000
   3)              0.000000           0.100000
   4)              0.000000          -1.900000
   5)              0.000000          -2.500000
   6)              0.000000          -1.000000

NO. ITERATIONS=        4

 DO RANGE(SENSITIVITY) ANALYSIS?
?n
:quit
STOP
```

```
END OF EXECUTION
CPU TIME: 1.48  ELAPSED TIME: 32.49
```

3.3. Running LPFORM in Interactive Mode

This section describes the features that have been implemented in LPFORM to help users define their problems in an interactive mode.

As before, the user accesses LPFORM and types "run-problem" to start the definition process. In this case, however, there is no external file containing the LPSPEC definition of the problem so the user types "user" in response to the LPSPEC file prompt. This automatically places the system in interactive mode.

```
| ?- run problem.

------------------ LPFORMulator  version 4.1 ------------------------

Enter the names of files to be used:

   1. File names must be enclosed by two single quotes.
   2. The input of a list (multiple items) must be terminated by a period.
   3. To use system default file names type "d"

        Template file(s)          : .
        LPSPEC file(s)            : user
        Template save file        : d
        MPGEN specification file  : d

Define identifiers for indices, variables and coefficients? y

Choose problem representations to be displayed:

        (1) LPSPEC compilation, valid and error statements.
        (2) Block reasoning, relevant blocks, flows, activities.
        (3) Model piece construction, a tentative LP formulation.
        (4) Puzzler reasoning, a complete LP formulation.
        (5) MPGEN specification.

        Checking points? (1,2,3,4,5): 4.

* Interactive Mode = On
```

The system now prompts the user to input LPSPEC statements one-by-one. As each statement is entered it is interpreted immediately and error diagnostics are displayed if a mistake is made.

```
Enter LPSPEC statement (h.=help, a.=assistance, ^Z=end): a.
LPSPEC Name: create block
```

The user can access an on-line help utility by typing "h." in response to the first *prompt (see Section 5.2). Statements are entered in "assistance" mode if "a" is entered followed by the name of an LPSPEC command. Finally, to end the definition phase, the user types ctrl-z (εz) in response to this prompt.*

The values of the arguments of the LPSPEC statement are entered one-at-a-time in response to prompts:

```
block_name(atom): example1

block_list(list): vendor,warehouse.

Statement No: 1
create_block(example1,[vendor,warehouse])
```

At this point, the call_model command has been entered successfully. Note that LPFORM indicates the kind of data that should be entered (atom or list) and, as illustrated in the following example, the permissible values where these are appropriate.

```
Enter LPSPEC statement (h.=help, a.=assistance, ^Z=end): a.
LPSPEC Name: link block

Permissible $link_type values are: {all, partial}.
link_type(atom): all

Permissible $link_mode values are: {file, explicit}.
link_mode(atom): explicit

directed_arcs(list): vendor,warehouse.

flow_var(atom): x

Statement No: 2
link_block(all,explicit,[vendor,warehouse],x)
```

The remaining LPSPEC statements are entered at this point. Finally, the user types "ctrl-z" in response to the prompt and obtains a complete listing of the problem statement:

```
Enter LPSPEC statement (h.=help, a.=assistance, ^Z=end): ^z

The following LPSPEC statements have been entered successfully:

create_block(example1,[vendor,warehouse]).
link_block(all,explicit,[vendor,warehouse],x).
create_block(vendor,[newyork,boston]).
create_block(warehouse,[buffalo,houston,denver]).
optimize(min,example1,cost,symbolic).
```

The problem can now be run to its conclusion as in the preceding section.

4. Further Examples of LPSPEC

This section contains three more examples of LPSPEC models. They illustrate some new LPSPEC statements and some of the different strategies for building models that are available in LPFORM. Only very general explanations are provided here. The reader is referred to Appendix I for detailed definitions of the LPSPEC statements.

When a model or sub-model is constructed from scratch using LPSPEC commands such as Create_block, Link_block and Def_Activity, we say that we are using a "first-principles approach". When sub-models are retrieved from the LPFORM model library using the Call_model command, we are using a "model-mapping approach". Mixtures of these two basic approaches are also possible.

The set of templates currently in the LPFORM model library are listed in Appendix III. User-defined models can be added to the library very easily.

The first example illustrates the use of the Call_Model statement to access the LPFORM model library. A transportation model template is used to formulate the transportation model of the previous session. The second example uses a simple product-mix problem to illustrate the use of the Def_Activity command. The final example shows how a mixture of Call_model and other commands can be used to build a complex model.

4.1. Model Mapping Approach: Using Call_model

In the example in Section 3, Create_block and Link_block commands were used to describe the graph associated with a transportation problem. This is a first-principles approach to model-building. The same model can be constructed using the model-mapping approach as shown in the following LPSPEC model:

```
Call_model(transportation,          example1,
           [from_block,to_block],   [vendor,warehouse],
           [flow],                  [x],
           [trans_cost,gain_or_loss,supply,demand], [tc,1,s,d]).
def_set(vendor,[newyork,boston]).
def_set(warehouse,[buffalo,houston,denver]).
optimize(min,example1,cost,symbolic).
```

The Call_model statement "maps" a stored template into the model being formulated by replacing template parameter names by the names that are to be used in the model. There are eight arguments arranged in pairs. The first pair of arguments replaces the template name (e.g. transportation) by the model name (e.g. example1). The last three pairs of arguments are lists which map the index sets, variables and data coefficients of the template into those for the model. The elements in each list must be in 1:1 correspondence. In the example, "from_block" in the template becomes "vendor" in the model and so on.

The stored templates are generally the most complicated of their type. They can be simplified by omitting index sets and/or replacing data coefficient names by constants. The REPLICATE command (see Appendix) can be used to add index sets to sub-models derived from templates.

4.2. First Principles Approach: Using Def_Activity

The *product_mix* model determines the levels of production activities, X_j, that maximize profits subject to constraints on the availability of raw materials:

$$\text{Maximize:} \quad \sum_{j \in output} p_j X_j$$

$$\text{Subject to:} \quad \sum_{j \in output} a_{i,j} X_j \leq b_i, \quad \forall \in input$$

The corresponding LPSPEC problem statement is:

```
Def_Activity(prodmix,[output],x,[input],[tech_coef],[output],
             profit,profit,#,#,#,linear,product_mix).
optimize(max,prodmix,profit,symbolic).
```

Briefly, the Def_Activity command defines a set of decision variables. The idea is illustrated in Figure 4-1 which is adapted from [3].

```
                    Activity X

Input 1 ----------->|
                    |------------> Output 1
Input 2 ----------->|
  ...              ...             ...
                    |
                    |------------> Output n
Input m ----------->|
```

Figure 4-1: Picture of an Activity Set

In the example, Def_Activity is used to define the set of production variables, X. The names of the input, output and cost coefficients are specified together with the type of the activity. The "#" symbols, specify that the slots for upper- and lower-bounds and unit should be ignored in the formulation.

4.3. A More Elaborate Example

The following example shows how a complex model can be constructed by specifying its component sub-models. LPFORM combines the sub-models automatically using the information provided by the user-supplied names of model parameters. The user must obviously adhere to a common set of names throughout the formulation.

The problem involves the determination of an optimal pattern of production and distribution of energy in a hypothetical national economy. Foreign and domestic sources of raw energy (oil,gas and coal) are used by conversion centers (electric-utilities and refineries) to produce processed energy (gasoline and electricity) to be consumed at final markets (residential, transportation and industrial). The latter also consume raw-energy. A pictorial view of the problem is shown in Figure 4-2 and the algebraic statement in Figure 4-3.

```
                                                            Sink
Source                  Conversion                          {residential,
{domestic,              {refineries,                        transportation,
foreign}                electric-utilities}                 industrial}
        TSC                              TCS
           so,co,re                         co,si,pe
  [ ] ----------------> [=|-] ----------------------> [ ]
   |   Raw_energy        X        Processed_energy     ^
   |  (oil,gas,coal)      co,pe  (gasoline,electricity) |
   |                                                    |
   v                    TSS                             |
                           so,si,re
    ---------------------------------------------------->
                         Raw_energy
                         (oil,gas,coal)
```

Figure 4-2: Graphic View of Energy Problem

From Figure 4-2, it seems intuitively obvious that the model is a combination of three transportation models (Sources to Conversions, Conversions to Sinks and Sources to Sinks) together with the product-mix problems at the conversion centers. This simple view-point is adopted in the LPSPEC formulation shown in Figure 4-4

Note that the product-mix template is used twice in the example - once for the conversion of the raw energy inputs and once for the capacity resource usage constraints. LPFORM is able to "collapse" the objective functions from the two models into one. The internal tableau and model data dictionary for this problem are listed in Figure 4-5 and 4-6.

$$\textbf{Min:} \quad \sum_{co \in Conversion} \sum_{pe \in Processed\_energy} cc_{co,pe} X_{co,pe}$$
$$+ \sum_{so \in Source} \sum_{co \in Conversion} \sum_{re \in Raw\_energy} tcscr_{so,co,re} TSC_{so,co,re}$$
$$+ \sum_{so \in Source} \sum_{si \in Sink} \sum_{re \in Raw\_energy} tcssr_{so,si,re} TSS_{so,si,re}$$
$$+ \sum_{co \in Conversion} \sum_{si \in Sink} \sum_{pe \in Processed\_energy} tccsp_{co,si,pe} TCS_{co,si,pe}$$

Subject To:

Raw Energy Supply Constraint:

$$\sum_{co \in Conversion} TSC_{so,co,re} + \sum_{si \in Sink} TSS_{so,si,re} \le ssr_{so,re}, \ \forall \begin{cases} re \in Raw\_energy \\ so \in Source \end{cases}$$

Material Balance - Production of Processed Energy using Raw Energy:

$$- \sum_{pe \in Processed\_energy} tc_{co,re,pe} X_{co,pe} + \sum_{so \in Source} TSC_{so,co,re} \ge 0, \ \forall \begin{cases} re \in Raw\_energy \\ co \in Conversion \end{cases}$$

Capacity Limits for Conversion Processes:

$$\sum_{pe \in Processed\_energy} cu_{co,cp,pe} X_{co,pe} \le clc_{co,cp}, \ \forall \begin{cases} cp \in Capacity \\ co \in Conversion \end{cases}$$

Balance Constraint - Processed Energy:

$$X_{co,pe} - \sum_{si \in Sink} TCS_{co,si,pe} \ge 0, \ \forall \begin{cases} pe \in Processed\_energy \\ co \in Conversion \end{cases}$$

Demand Constraint for Raw Energy at Sink:

$$\sum_{so \in Source} TSS_{so,si,re} \ge dsr_{si,re}, \ \forall \begin{cases} re \in Raw\_energy \\ si \in Sink \end{cases}$$

Demand Constraint for Processed Energy at Sink:

$$\sum_{co \in Conversion} TCS_{co,si,pe} \ge dsp_{si,pe}, \ \forall \begin{cases} pe \in Processed\_energy \\ si \in Sink \end{cases}$$

Figure 4-3: Mathematical Formulation of Energy Problem

```
Call_model(transportation,                              energy_model,
           [from_block,      to_block,       commodity],
           [source,          conversion,     raw_energy],
           [flow],                                   [tsc],
           [gain_or_loss], [1]).

Call_model(transportation,                              energy_model,
           [from_block,      to_block,       commodity],
           [source,          sink,           raw_energy],
           [flow],                                   [tss],
           [gain_or_loss], [1]).

Call_model(transportation,                              energy_model,
           [from_block,      to_block,       commodity],
           [conversion,      sink,           processed_energy],
           [flow],                                   [tcs],
           [gain_or_loss], [1]).

Call_model(product_mix,                            energy_model,
           [block,           input,          output],
           [conversion,      raw_energy,     processed_energy],
           [volume],                            [x],
           [tech_coef],     [tech_coef]).

Call_model(product_mix,                            energy_model,
           [block,           input,          output],
           [conversion,      capacity,       processed_energy],
           [volume],                            [x],
           [tech_coef,available_input],
           [capacity_usage,capacity_limit]).

def_set(source,[foreign,domestic]).
def_set(conversion,[refineries,electric_utilities]).
def_set(sink,[residential,transportation,industrial]).
def_set(raw_energy,[oil,gas,coal]).
def_set(processed_energy,[gasoline,electricity]).
def_set(capacity,[labor,machine_hours]).

optimize(min,energy_model,cost,symbolic).
```

Figure 4-4: LPSPEC Statements for Model-Mapping Approach

5. LPFORM Utilities

In this section we provide a brief overview of some of the features of LPFORM that are designed to provide a good system development and prototyping environment. LPSPEC features are covered first followed by a brief description of the "Help" command.

```
PROBLEM/MODEL/FRAGMENT = energy_model.

ROW\COL        X(co,pe)             TSC(so,co,re)
OBJ=           +S{co;pe}cc[co;pe]   +S{so;co;re}tcscr[so;co;re]
Use[so;re]                          +S{co}1[so;co;re]
Supply[co;re]  -S{pe}tc[co;re;pe]   +S{so}1[so;co;re]
Supply[si;re]
Use[co;pe]     +1[co;pe]
Supply[si;pe]
Use[co;cp]     +S{pe}cu[co;cp;pe]

................................... continued ...................................

ROW\COL        TSS(so,si,re)              TCS(co,si,pe)                 RHS
OBJ=           +S{so;si;re}tcssr[so;si;re] +S{co;si;pe}tccsp[co;si;pe] MIN
Use[so;re]     +S{si}1[so;si;re]                                        < +ssr[so;re]
Supply[co;re]                                                           > +0[co;re]
Supply[si;re]  +S{so}1[so;si;re]                                        > +dsr[si;re]
Use[co;pe]                                 -S{si}1[co;si;pe]            > +0[co;pe]
Supply[si;pe]                              +S{co}1[co;si;pe]            > +dsp[si;pe]
Use[co;cp]                                                              < +clc[co;cp]
```

Figure 4-5: Internal tableau Representation of Energy Model

5.1. LPSPEC Support Facilities

LPSPEC statements are somewhat independent, because each individual statement has its own meaning and own effects in the LPFORM system. Definitions of all LPSPEC statements are listed in Appendix I. Each statement in LPSPEC is defined to the system by a "schema" or representative example. Thus the LINK_BLOCK statement has the schema:

link_block(link_type,link_mode,directed_arcs,flow-var).

The schema contains information to allow validity checking. When an LPSPEC model is parsed, each statement is checked against its schema. Each argument is checked to see if its data type is valid (atom, list or numeric), and if its value is within a permissible range. If a "key" has been defined for the statement the system also checks to ensure that no two statements have the same value for the key. For example, if we require uniqueness in the directed linkage between any two blocks, the attributes, from-block and to-block can be defined as a key in the schema for the *Link-Block* state-

```
* Symbol convention of energy_model *

Set Reference:
SYMBOL:            SET NAME:
------------------------------------------------------------------------------------------
cp               : Capacity
                   Meaning: input.
co               : Conversion
                   Meaning: block, transhipment_node.
pe               : Processed_energy
                   Meaning: commodity, output.
re               : Raw_energy
                   Meaning: commodity, input.
si               : Sink
                   Meaning: to_block.
so               : Source
                   Meaning: from_block.

Activity Reference:
SYMBOL:            ACTIVITY (VARIABLE):
------------------------------------------------------------------------------------------
TSC(so,co,re)    : TSC(Source,Conversion,Raw_energy)
TSS(so,si,re)    : TSS(Source,Sink,Raw_energy)
TCS(co,si,pe)    : TCS(Conversion,Sink,Processed_energy)
X(co,pe)         : X(Conversion,Processed_energy)

Coefficient Reference:
SYMBOL:            COEFFICIENT (DATA):
------------------------------------------------------------------------------------------
1[so;co;re]      : 1[Source,Conversion,Raw_energy]
1[so;si;re]      : 1[Source,Sink,Raw_energy]
1[co;si;pe]      : 1[Conversion,Sink,Processed_energy]
cu[co;cp;pe]     : Capacity_usage[Conversion,Capacity,Processed_energy]
clc[co;cp]       : Capacity_limit[Conversion,Capacity]
1[co;pe]         : 1[Conversion,Processed_energy]
0[co;pe]         : 0[Conversion,Processed_energy]
tc[co;re;pe]     : Tech_coef[Conversion,Raw_energy,Processed_energy]
0[co;re]         : 0[Conversion,Raw_energy]
ssr[so;re]       : Rhs?*^supply^source^raw_energy[Source,Raw_energy]
dsr[si;re]       : Rhs?*^demand^sink^raw_energy[Sink,Raw_energy]
dsp[si;pe]       : Rhs?*^demand^sink^processed_energy[Sink,Processed_energy]
tcscr[so;co;re]  : Obj?*^trans_cost^source^conversion^raw_energy[Source,Conversion,
                   Raw_energy]
tcssr[so;si;re]  : Obj?*^trans_cost^source^sink^raw_energy[Source,Sink,Raw_energy]
tccsp[co;si;pe]  : Obj?*^trans_cost^conversion^sink^processed_energy[Conversion,Sink,
                   Processed_energy]
cc[co;pe]        : Obj?*^profit^conversion^processed_energy[Conversion,Processed_energy]
```

Figure 4-6: Data Dictionary for Energy Model

ment. The schemas also contain special "explanations" that can be used to prompt the user for information if an argument value is missing or invalid.

A useful set of PROLOG utilities have been developed in order to handle the syntactical definition of the LPSPEC language. In particular, changes of LPSPEC statements during the prototyping stage can be accommodated by editing the corresponding schemas using these utilities. An automatic cross-referencing procedure ("allied

```
* NAME energy_model
* CAPACITY = 'labor,machine_hours'
E CAPACITY _ 1 THRU NCP _ 2
* CONVERSION = 'refineries,electric_utilities'
E CONVERSION _ 1 THRU NCO _ 2
* PROCESSED_ENERGY = 'gasoline,electricity'
E PROCESSEDENERGY _ 1 THRU NPE _ 2
* RAW_ENERGY = 'oil,gas,coal'
E RAWENERGY _ 1 THRU NRE _ 3
* SINK = 'residential,transportation,industrial'
E SINK _ 1 THRU NSI _ 3
* SOURCE = 'foreign,domestic'
E SOURCE _ 1 THRU NSO _ 2
*
DATA= NCP,NCO,NPE,NRE,NSI,NSO,CU(NCO#NCP#NPE),CLC(NCO#NCP),TC(NCO#NRE#NPE)
DATA= SSR(NSO#NRE),DSR(NSI#NRE),DSP(NSI#NPE),TCSCR(NSO#NCO#NRE)
DATA= TCSSR(NSO#NSI#NRE),TCCSP(NCO#NSI#NPE),CC(NCO#NPE)
*
VAR=TSC(so,co,re), so in Source, co in Conversion, re in Rawenergy
VAR=TSS(so,si,re), so in Source, si in Sink, re in Rawenergy
VAR=TCS(co,si,pe), co in Conversion, si in Sink, pe in Processedenergy
VAR=X(co,pe), co in Conversion, pe in Processedenergy
*
MIN  S S cc[co;pe]X(co,pe) +S S S tcscr[so;co;re]TSC(so,co,re)
: +S S S tcssr[so;si;re]TSS(so,si,re) +S S S tccsp[co;si;pe]TCS(co,si,pe)
co in Conversion, pe in Processedenergy, so in Source, co in Conversion,
: re in Rawenergy, so in Source, si in Sink, re in Rawenergy,
: co in Conversion, si in Sink, pe in Processedenergy
*
FOR so in Source FOR re in Rawenergy
  S TSC(so,co,re) +S TSS(so,si,re) < ssr[so;re]
co in Conversion, si in Sink
*
FOR co in Conversion FOR re in Rawenergy
  - S tc[co;re;pe]X(co,pe) +S TSC(so,co,re) > 0
pe in Processedenergy, so in Source
*
FOR si in Sink FOR re in Rawenergy
  S TSS(so,si,re) > dsr[si;re]
so in Source
*
FOR co in Conversion FOR pe in Processedenergy
  X(co,pe) -S TCS(co,si,pe) > 0
si in Sink
*
FOR si in Sink FOR pe in Processedenergy
  S TCS(co,si,pe) > dsp[si;pe]
co in Conversion
*
FOR co in Conversion FOR cp in Capacity
  S cu[co;cp;pe]X(co,pe) < clc[co;cp]
pe in Processedenergy
```

Figure 4-7: Formulation Statements for APL Tableau generator

schemas") ensures, for example, that if an "attribute" name is changed, it is changed in every schema where it is used.

5.2. LPFORM Help Facility

LPFORM provides substantial on-line documentation and help. Only a brief explanation will be provided here. Figure 5-1 shows the main help screen and Figure 5-2 the screen obtained in response to a request by the user to "View Model Templates".

```
yes
| ?- help.


*********************************************************************
*                                                                   *
*                    LPFORM On-line Help Utility                    *
*                                                                   *
*-------------------------------------------------------------------*
*   Type:                                                           *
*                                                                   *
*       1.          List LPFORM System Commands.                    *
*                                                                   *
*       2.          Display Definitions of LPSPEC Statements.       *
*                                                                   *
*       3.          Review Model Templates.                         *
*                                                                   *
*       0.          Exit Help.                                      *
*                                                                   *
*********************************************************************
Enter (0,1,2,3): 3.
```

Figure 5-1: Main Help Screen

This menu supports interactive users who wish to define LPSPEC models. The menu choices are self-explanatory. The displays generated by the first two choices are shown in the Appendix 2 and Appendix 1, respectively. Menu choice 3 results in the display shown in Figure 5-2.

This menu allows users to access some of the model management functions of LPFORM. Since users can add new templates to the system library, it is necessary to provide the on-line documentation, dictionary and query facilities associated with this menu.

6. Conclusion

This paper has reviewed the main features of the LPSPEC language and provided examples of its use in defining LP models. Together with the Appendices, the paper constitutes a fairly complete guide to the use of the current version of LPFORM. The system is in an early stage of development. The main infrastructure has been com-

```
-------------------- TEMPLATE INFORMATION MENU --------------------

     Enter:

     1.                List the names of all Templates in library.

     2.                Display an existing Model Template.

     3.                Obtain the Template symbol convention.

     4.                Output a Template in MPGEN form to a file.

     5.                Search for Templates using key words.

     0.                Quit.

     (0,1,2,3,4,5): 1


===================== MODEL TEMPLATE LIBRARY =====================

exog_demand, exog_supply, general_lp_max, general_lp_min,
input_cons, inventory, process_selection, product_mix, purchase,
transportation.
```

Figure 5-2: Template Maintenance Facility

pleted and a subset of problems can be successfully formulated. The definition of the LPSPEC language will have to expand and adapt to new requirements over time. A number of features have been built-in to LPFORM to facilitate this process.

The major directions for future research and development are:

1. Add rules to enable the system to handle models outside the production planning domain.
2. Complete work on the "data mode" of formulation in which the symbols in the problem statement are bound to values on external files and databases.
3. Develop the graphics interface to the system.
4. Investigate and develop methods to take advantage of specialized knowledge in different application domains.

Finally, after the interface has been built, we will need to carry-out extensive tests with real users to see if the new scheme for representing LPs is successful.

References

1. Astrahan, M. M., and Chamberlin, D. D. "Implementation of a Structured English Query Language". *Communications of the ACM 18*, 10 (October 1985), 580-588.

2. Dano, Sven. *Linear Programming in Industry - Theory and Applications.* Springer-Verlag, New York, 1974.

3. Dantzig, George B.. *Linear Programming and Extensions.* Princeton University Press, Princeton, N.J., 1963.

4. Greenberg, Harvey J. "A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models". *ACM Transactions on Mathematical Software 9*, 1 (March 1983), 18-56.

5. Ma, Paichun. *An Intelligent Approach to Formulating Linear Programs.* Ph.D. Th., New York University, 1986. Ph.D. Dissertation Proposal.

6. Ma, P., F. H. Murphy and E. A. Stohr. Design of a Graphics Interface for Linear Programming. Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986. Working Paper.

7. Ma, P., F. H. Murphy and E. A. Stohr. A Representation Scheme for an Intelligent LP System. Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986. Working Paper.

8. Meeraus, A. *General Algebraic Modeling System (GAMS): User's Guide, Version 1.* Development Research Ceter, World Bank, 1984.

9. *IBM Mathematical Programming Language Extended/370 (MPSX/370), Program Reference Manual, SH19-1095.* IBM Corporation, Paris, France, 1975.

10. Murphy, F. H. and E. A. Stohr. "An Intelligent System for Formulating Linear Programs". *International Journal of Decision Support Systems 2*, 2 (March 1986).

11. Murphy, F. H., E. A. Stohr and P. Ma. The Science and Art of Formulating Linear Programs. Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986. Working Paper.

12. *OMNI Linear Programming System: User Manual and Operating Manual.* Haverly Systems Inc., Denville, N.J., 1977.

13. Schrage, Linus. *Linear, Integer, and Quadratic Programming with LINDO.* The Scientific Press, Palo Alto, 1984.

14. Smith, J.M. and D.C.P. Smith. "Database Abstractions: Aggregation". *Communications of the ACM 20*, 6 (1977), 405-413.

15. Stohr, Edward A. A Mathematical Programming Generator System. Working Paper 96, New York University, New York, 1985.

I. Appendix: LPSPEC Language Statements

Section I.1 of this appendix contains a complete listing of LPSPEC commands. This listing can also be obtained on the screen using the on-line help feature. Section I.2 contains detailed definitions for each command.

I.1. Listing of LPSPEC Commands

```
 1: block_input_output(block_name, input, output).

 2: call_model(model_name, problem_name, model_index, problem_index,
              model_var, problem_var, model_coef, problem_coef).

 3: create_block(block_name, block_list).

 4: def_activity(block_name, activity_set, activity_var,
                activity_input, i_o_coef, activity_output, obj_coef,
                obj_type, upper_bound, lower_bound, unit,
                math_property, activity_type).

 5: def_inventory(block_name, commodity_set, inventory_var,
                 incoming_inv, gain_or_loss, outgoing_inv, obj_coef,
                 obj_type, upper_bound, lower_bound, unit,
                 inventory_property, inventory_type).

 6: def_rhs(cons_type, rhs_coef, row_set).

 7: def_set(set_name, set_member).

 8: def_transport(transported_commodity, directed_arcs, gain_or_loss,
                 trans_mode).

 9: link_block(link_type, link_mode, directed_arcs, flow_var).

10: link_output_input(block_set, common_commodity).

11: optimize(optimize_direction, problem_name, obj_type,
            formulation_mode).

12: short_name(model_name, name_type, long_name, short_name).

13: table(table_name, table_index, table_type, content_mode,
         content_spec, unit).
```

I.2. Detailed Definitions of LPSPEC Statements

In the following pages the LPSPEC statements are listed in alphabetical order. The description for each statement consists of a statement format, a screen design, the data types and domain sets of each argument, the statement's key (if any), an example and a brief explanation.

A "domain set" defines the permissible set of values for an argument. The domain set members are listed as part of the schema definition. When an argument is declared to have a data type of "atom" its value must be a single character string (no quotes, imbedded commas or other characters except for underline). A "list" data type consists of the empty string or one or more strings separated by commas (no imbedded blanks) and enclosed in square brackets. A "file-spec" data-type must contain a valid file name complete with file extension (e.g. myfile.new).

The "allied schemas" for an argument are the other LPSPEC statements that use the argument. The "key" of an LPSPEC schema is the set of one or more arguments that must have unique values in every instance of that statement in the model.

LPSPEC: **block_input_output**(block_name, input, output)

Screen Interface:

```
 ___________________________________________________________________________________________
| PROBLEM: energy_model        VERSION: 1   LAST UPDATE: 10/01/86  | DATA:        |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|              |
|___________________________________________________________________| MODE: DATA   |
| LEVEL: 1  GRAPH: 1         CURRENT OP: BLOCK_INPUT_OUTPUT         |  REL:  r     |
|        BLOCK: conversion                                          |  TAB:  t     |
|        ____________________________________________               |  PAR:  p     |
|       | INPUTS:                 OUTPUTS:             |             |  SET: {s}    |
|       | raw_energy______   |    processed_energy     |             |______________|
|       | ________________   |    ________________     |             | STRUCTURE:   |
|       | ________________   |    ________________     |             |              |
|       |______________________________________________|             |  C-B:   []   |
|                                                                   |  L-B:   -->  |
|                                                                   |L-O-I:  :--:  |
|                                                                   | B-IO:  =[]=  |
|                                                                   |  D-I:  .:.   |
|                                                                   |  D-C:  |_|   |
|                                                                   |  D-R:  0=0   |
|                                                                   |  D-T:  ___   |
|                                                                   |  D-A:  =|-   |
|                                                                   |  C-M:  < >   |
|___________________________________________________________________|  REP:  ]]]   |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |  OPT   ^/v   |
|                                                                   |              |
|___________________________________________________________________|______________|
```

Figure I-1: Defining Inputs and Outputs of a Block

	Argument	Data Type
1:	block_name	atom
	Allied Schema: def_inventory, def_activity, create_block.	
2:	input	list
3:	output	list

Key: [1]

Example:

```
block_input_output(conversion,[raw_energy],[processed_energy]).
```

Explanation:

Block-Inputs-and-Outputs (B-IO): Specifies the inputs to, and outputs from, a block one block at a time. A block usually represents activities at a location and the inputs and outputs are commodities. The information captured by this statement is generated (see Figure I-1) by prompting for information using slot titles, such as "inputs", and "outputs".

B-IO is usually used as part of the specification of a multi-commodity network. B-IO is used first to specify the inputs and outputs of several blocks; the blocks are then connected automatically using the LINK_OUTPUTS_INPUTS Statement which will establish links from the blocks that output each commodity to the blocks that input it.

LPSPEC: **call_model**(model_name, problem_name, model_index, problem_index, model_var, problem_var, model_coef, problem_coef)

Screen Interface:

```
 ____________________________________________________________________________________________
| PROBLEM: energy_model          VERSION: 1   LAST UPDATE: 10/01/86  |  DATA:        |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|               |
|___________________________________________________________________| MODE: DATA    |
| LEVEL: 1  GRAPH: 1          CURRENT OP: CALL_MODEL                 |  REL:  r      |
|        TEMPLATE MODEL: transportation                              |  TAB:  t      |
|        ____________________________________________                |  PAR:  p      |
|       | TEMPLATE INDICES:                          |                |  SET:  {s}    |
|       | FROM BLOCK     : source___________         |                |_______________|
|       | TO BLOCK       : conversion_______         |                | STRUCTURE:    |
|       | COMMODITY      : raw_energy_______         |                |               |
|       | TEMPLATE MODEL VARIABLE:                   |                |  C-B:   []    |
|       | FLOW           : tsc______________         |                |  L-B:   -->   |
|       | TEMPLATE MODEL COEFFICIENTS:               |                |L-O-I:   :--:  |
|       | TRANS_COST     : ?________________         |                | B-IO:   =[]=  |
|       | GAIN_OR_LOSS   : 1________________         |                |  D-I:   .:.   |
|       | SUPPLY         : ?________________         |                |  D-C:   |_|   |
|       | DEMAND         : ?________________         |                |  D-R:   0=0   |
|       |____________________________________________|                |  D-T:   ___   |
|                                                                    |  D-A:   =|-   |
|                                                                    |  C-M:   < >   |
|____________________________________________________________________|  REP:   ]]]   |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []   |  OPT    ^/v   |
|                                                                    |               |
|____________________________________________________________________|_______________|
```

Figure I-2: Mapping a Transportation Model on to the Sources - Conversions Link

	Argument	Data Type
1:	model_name Allied Schema: short_name.	atom
2:	problem_name Allied Schema: optimize.	atom
3:	model_index	list
4:	problem_index	list
5:	model_var	list
6:	problem_var	list
7:	model_coef	list
8:	problem_coef	list

Example:

```
call_model(transportation,                      energy_model,
           [from_block,to_block,commodity],     [source,conversion,raw_energy],
           [flow],                              [t],
           [gain_or_loss], [1]).
```

Explanation:

Call-Model (C-M): Specifies that a pre-existing template from the LPFORM system library is to be incorporated as a sub-model of the new formulation. The names of parameters in the pre-existing model must be matched with the names for these parameters in the new model. The statement has four slots to be filled, model name, index set names, variable names, and data coefficient names or values. These must be specified in pairs and in order. Each pair has the parameter name of the pre-existing model followed by a user-supplied name that is appropriate for the current context. If the parameter of the pre-existing model is a set (such as index set names) the corresponding new name must be a set and the included members of both sets must be in one-to-one correspondence. In the example below, the template index set, "from_block", acquires the name "source" in the formulated model.

An index from the template can be suppressed simply by omitting it from the list (this reduces the size of a model). Alternatively, the unknown value "?" can be used as a variable name or coefficient name. Later, LPFORM will generate proper names for the unknown values when they are needed.

LPSPEC: **create_block**(block_name, block_list)

Screen Interface:

```
 ____________________________________________________________________________________
| PROBLEM: energy_model          VERSION: 1   LAST UPDATE: 10/01/86 |  DATA:        |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|               |
|___________________________________________________________________| MODE: SYMB    |
| LEVEL: 2  GRAPH: 1         CURRENT OP: CREATE_BLOCK               |  REL:  r      |
|                                                                   |  TAB:  t      |
|                                                                   |  PAR:  p      |
|      source            conversion             sink                |  SET: {s}     |
|     --------          -----------       ----------------          |_______________|
|                                           residential             | STRUCTURE:    |
|                                              [ ]                  |               |
|     domestic           refineries                                 |  C-B:   []    |
|       [ ]                 [ ]                                     |  L-B:   -->   |
|                                                                   |L-O-I:  :--:   |
|                                          transportation           | B-IO:  =[]=   |
|                                              [ ]                  |  D-I:   .:.   |
|     foreign          electric-utility                             |  D-C:   |_|   |
|       [ ]                 [ ]                                     |  D-R:   0=0   |
|                                           industrial              |  D-T:   ___   |
|                                              [ ]                  |  D-A:   =|-   |
|                                                                   |  C-M:   < >   |
|___________________________________________________________________|  REP:   ]]]   |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |  OPT:   ^/v   |
| ENTER BLOCK NAME: industrial____________                          |               |
|___________________________________________________________________|_______________|
```

Figure I-3: Create Blocks at subsequent levels

	Argument	Data Type
1:	block_name	atom
	Allied Schema: def_inventory, def_activity, block_input_output.	
2:	block_list	list

Example:

```
create_block(source,[foreign,domestic]).
create_block(conversion,[refinery,electric_utility]).
create_block(sink,[residential,transportation,industrial]).
```

Explanation:

Create-Block (C-B): Specifies the set of children of a block in a hierarchical structure. In the graphics interface a small square indicates the position of each block and the user is prompted for the names of the block and its children. If block_name does not exist it is created. If block-list is empty, a block with no children is created.

A block is a user-defined grouping of activities in space or time. Create-Block statements can be used to create a multiple level hierarchy. Blocks at the lowest level in the hierarchy represent real world entities such as vendors, warehouses, factories, etc.

LPSPEC: **def_activity**(block_name, activity_set, activity_var, activity_input, i_o_coef, activity_output, obj_coef, obj_type, upper_bound, lower_bound, unit, math_property, activity_type)

Screen Interface:

```
 ________________________________________________________________________________________
| PROBLEM: energy_model         VERSION: 1   LAST UPDATE: 10/01/86  |  DATA:           |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|                  |
|___________________________________________________________________| MODE: SYMB       |
| LEVEL: 2  GRAPH: 1        CURRENT OP: DEF_ACTIVITY                |  REL:  r         |
|        BLOCK: conversion                                          |  TAB:  t         |
|        ____________________________________________               |  PAR:  p         |
|       | ACTIVITY SET: processed_energy             |             |  SET: {s}        |
|       | ACTIVITY VAR: x______________              |             |__________________|
|       | INPUTS: NAME                               |             | STRUCTURE:       |
|       | raw_energy______      ________________     |             |                  |
|       | OUTPUTS: NAME                              |             |  C-B:   []       |
|       | processed_energy      _______________      |             |  L-B:   -->      |
|       | OBJ. COEFFT : conversion_cost___           |             |L-O-I:  :--:      |
|       | OBJ. TYPE   : cost_____________            |             | B-IO:  =[]=      |
|       | ACT. COEFFTS: tech_coef_________           |             |  D-I:   .:.      |
|       | UPPER BOUNDS: __________________           |             |  D-C:   |_|      |
|       | LOWER BOUNDS: __________________           |             |  D-R:   0=0      |
|       | UNITS       : __________________           |             |  D-T:   ___      |
|       | MATH PROP   : linear___________            |             |  D-A:   =|-      |
|       | ACT. TYPE   : product_mix______            |             |  C-M:   < >      |
|       |____________________________________________|             |  REP:   ]]]      |
|___________________________________________________________________|  OPT:   ^/v      |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |                  |
|                                                                   |                  |
|___________________________________________________________________|__________________|
```

Figure I-4: Define Production Activity

	Argument	Data Type	Domain Set
1:	block_name	atom	
	Allied Schema: def_inventory, block_input_output, create_block.		
2:	activity_set	list	
3:	activity_var	atom	
4:	activity_input	list	
5:	i_o_coef	list	
6:	activity_output	list	
7:	obj_coef	atom	
	Allied Schema: def_inventory.		
8:	obj_type	atom	*objective_function_type*
	Allied Schema: def_inventory, optimize.		
9:	upper_bound	atom	
	Allied Schema: def_inventory.		
10:	lower_bound	atom	
	Allied Schema: def_inventory.		

Argument	Data Type	Domain Set
11: unit	atom	
Allied Schema: def_inventory, table.		
12: math_property	atom	*math_property*
13: activity_type	atom	*activity_type*

Key: [2, 3]

Domain Set Members:

objective_function_type = {profit, cost}
math_property = {linear, non_linear}
activity_type = {product_mix, transportation, blending, purchasing}

Example:

```
def_activity(conversion,processed_energy,x,[raw_energy],[tech_coef],
    [processed_energy],conversion_cost,cost,#,#,#,linear,product_mix).
```

Explanation:

Def-Activity (D-A): Defines a set of activities in a block. These usually involve a form transformation between inputs and outputs. For example, the activity might be of the product-mix or blending type. Each activity translates to a set of columns in the tableau and is associated with a variable name. The screen for D-A prompts for complete information about the activity, including the associated activity set, the input and output sets, the variable name, objective coefficient name, etc. Many of these slots have default values and need not be filled-in by the user.

The "activity set" specifies all the instances of an activity. For example, if we have a decision variable, X_j, $j \in$ *Products then Products is the activity set. The def_set command would be used to specify the elements of products.*

The "activity_input" list contains the set names for the different kinds of input to the activity. For example, if raw_materials and resources are used in the activity, the activity_input list would be: [raw_mats,resources] where each of the items in the list would be sets specified by the DEF_SET command.

The "i_o_coefficient" list contains the names of the data coefficients that perform the transformation from inputs to outputs for the activity. These must be stated in the same order as their associated inputs in the activity_input_list. If there is more than one output, the TAB (table) statement must be used to provide the information necessary to associate the technical coefficients with the correct outputs.

The "activity_output" list contains the set names for the different kinds of output from the activity.

The "obj_coef" is the name for the array of coefficients that are to be associated with the activity in the objective function. "Obj_type" is used to determine the sign (+ or -) of obj_coef in the objective function.

The "upper_bound" and "lower_bound" arguments provide names for the arrays of data coefficients that define the upper and lower bounds for the levels of the activity.

The following three arguments are not used by the current version of LPFORM but are useful for documenting the model:

The "units" argument is used to specify the units in which the activity is expressed (e.g. barrels of oil, units of product).

The "math_prop" slot is used to specify whether the activity is linear or should be approximated by its piece-wise linear approximation.

The "activity_type" is used to specify the type of activity (e.g. product_mix or blending, etc.)

LPSPEC: **def_inventory**(block_name, commodity_set, inventory_var, incoming_inv,gain_or_loss, outgoing_inv, obj_coef, obj_type,upper_bound, lower_bound, unit, inventory_property,inventory_type)

	Argument	Data Type	Domain Set
1:	block_name	atom	
	Allied Schema: def_activity, block_input_output, create_block.		
2:	commodity_set	list	
3:	inventory_var	atom	
4:	incoming_inv	list	
5:	gain_or_loss	atom	
	Allied Schema: def_transport.		
6:	outgoing_inv	list	
7:	obj_coef	atom	
	Allied Schema: def_activity.		
8:	obj_type	atom	*objective_function_type*
	Allied Schema: def_activity, optimize.		
9:	upper_bound	atom	
	Allied Schema: def_activity.		
10:	lower_bound	atom	
	Allied Schema: def_activity.		
11:	unit	atom	
	Allied Schema: def_activity, table.		
12:	inventory_property	atom	
13:	inventory_type	atom	*inventory_type*

Key: [2, 3]

Domain Set Members:

objective_function_type = {profit, cost}

inventory_type = {input, output, work_in_process}

Explanation:

Def-Inventory (D-I): Specifies the inventory to be accounted for in a block one commodity at a time. Some relevant properties about the inventory, such as the type of inventory (input, work-in-process or output) and special restrictions are specified by this statement. This command is similar to Def-Activity, but has not been implemented yet.

LPSPEC: **def_set**(set_name, set_member)

Screen Interface:

```
 ____________________________________________________________________________________
| PROBLEM: energy_model        VERSION: 1   LAST UPDATE: 10/01/86 | DATA:           |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|                 |
|                                                                 | MODE: DATA      |
|_________________________________________________________________|                 |
| LEVEL: 1  GRAPH: 0          CURRENT OP: DEF_SET                 |  REL:  r        |
|                                                                 |  TAB:  t        |
|          ______________________________________________          |  PAR:  p        |
|         | SET NAME    : source_______________________ |         |  SET: {s}       |
|         | SET MEMBERS : domestic,foreign_____________ |         |_________________|
|         |               _____________________________ |         | STRUCTURE:      |
|         |               _____________________________ |         |                 |
|         |               _____________________________ |         |  C-B:  []       |
|         |               _____________________________ |         |  L-B:  -->      |
|         |               _____________________________ |         |L-O-I: :-->      |
|         |_____________________________________________|         | B-IO: =[]=      |
|                                                                 |  D-I:  .:.      |
|                                                                 |  D-C:  |_|      |
|                                                                 |  D-R:  0=0      |
|                                                                 |  D-T:  ___      |
|                                                                 |  D-A:  =|-      |
|                                                                 |  C-M:  < >      |
|_________________________________________________________________|  REP:  ]]]      |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE [] |  OPT:  ^/v      |
|                                                                 |                 |
|_________________________________________________________________|_________________|
```

Figure I-5: Defining a source set

	Argument	Data Type
1:	set_name	atom
2:	set_member	list

Key: [1]

Example:

```
def_set(source,[domestic,foreign]).
```

Explanation:

Def-Set (Set): Specifies a set name and its members. The sets determine the dimensions of the model. For example, a single commodity transportation problem between "vendor" and "warehouse" is not really significant unless vendor and warehouse are sets. The screen interface prompts for the set name and its members.

LPSPEC: **def_transport**(transported_commodity, directed_arcs, gain_or_loss, trans_mode)

Screen Interface:

```
 ____________________________________________________________________________________
| PROBLEM: energy_model         VERSION: 1   LAST UPDATE: 10/01/86  |  DATA:        |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|               |
|___________________________________________________________________|  MODE: SYMB   |
| LEVEL: 2   GRAPH: 1        CURRENT OP: DEF_TRANSPORT               |  REL:  r      |
|                                                                   |  TAB:  t      |
|         ____________________________________________              |  PAR:  p      |
|        |  COMMODITY SET: processed_energy           |             |  SET:  {s}    |
|        |  DIRECTED ARCS: conversion,sink            |             |_______________|
|        |                 ______________             |             | structure:    |
|        |  GAIN OR LOSS  : 1_____________             |             |               |
|        |  MODE          : ______________             |             |  C-B:   []    |
|        |____________________________________________|             |  L-B:   -->   |
|                                                                   |L-O-I:  :--:   |
|                                                                   | B-IO:  =[]=   |
|                                                                   |  D-I:   .:.   |
|                                                                   |  D-C:   |_|   |
|                                                                   |  D-R:   0=0   |
|                                                                   |  D-T:   ___   |
|                                                                   |  D-A:   =|-   |
|                                                                   |  C-M:   < >   |
|___________________________________________________________________|  REP:   ]]]   |
| BACK []   FORWD []   DELETE []   UNDELETE []   SHOW-DET []   ERASE [] |  OPT    ^/v   |
|                                                                   |               |
|___________________________________________________________________|_______________|
```

Figure I-6: Defining the Commodities to be Transported

	<u>**Argument**</u>	<u>**Data Type**</u>
1:	transported_commodity	atom
2:	directed_arcs	list
	Allied Schema: link_block.	
3:	gain_or_loss	atom
	Allied Schema: def_inventory.	
4:	trans_mode	atom

Key: [1, 2]

Example:

```
def_transport(processed_energy,[conversion,sink],1,#).
```

Explanation:

Def-Transport (D-T): Specifies the commodity set to be transported on a directed arc or a set of directed arcs. The maximum and minimum capacity requirements of the flow, and the names of the arrays that contain the coefficients specifying gains or losses associated with the flow, are also specified. "Mode" is the set name for the different transportation modes that might be available (e.g. by rail, truck or air).

LPSPEC: **link_block**(link_type, link_mode, directed_arcs, flow_var)

Screen Interface:

```
 ________________________________________________________________________________________
| PROBLEM: energy_model          VERSION: 1   LAST UPDATE: 10/01/86  |  DATA:          |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|                  |
|____________________________________________________________________| MODE: SYMB      |
| LEVEL: 1  GRAPH: 1         CURRENT OP: LINK_BLOCK                  |  REL:  r        |
|                                                                    |  TAB:  t        |
|        | LINK TYPE :  SPACE: X_  |  TIME: ______  |                |  PAR:  p        |
|        | LINK MODE :  FILE: ___  |  EXPLICIT: X__ |                |  SET: {s}       |
|        | DIR ARCS  : source,conversion,conversion  |               |_________________|
|        |             sink,source,sink___________   |               | STRUCTURE:      |
|        | FLOW VAR  : t___________________________  |               |                 |
|        |___________________________________________|               |  C-B:  []       |
|                                                                    |  L-B:  -->      |
|                        energy_model                                |L-O-I: :-->      |
|                       ---------------                              | B-IO: =[]=      |
|       source              conversion              sink             |  D-I:  .:.      |
|         [ ]----------------->[ ]------------------>[ ]             |  D-C:  |_|      |
|          |                                          ^              |  D-R:  0=0      |
|          v                                          |              |  D-T:  ___      |
|          ------------------------------------------>               |  D-A:  =|-      |
|                                                                    |  C-M:  < >      |
|____________________________________________________________________|  REP:  ]]]      |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |  OPT:  ^/v      |
| ENTER LINK TYPE: space__________________                          |                  |
|____________________________________________________________________|_________________|
```

Figure I-7: Link Blocks at Set Level

	Argument	Data Type	Domain Set
1:	link_type	atom	*link_type*
2:	link_mode	atom	*link_mode*
3:	directed_arcs	list	
	Allied Schema: def_transport.		
4:	flow_var	atom	

Key: [3]

Domain Set Members:

link_type = {space, time}

link_mode = {file, explicit}

Example:

```
link_block(space,explicit,[source,conversion,conversion,sink,source,sink],t).
```

Explanation:

Link-Block (L-B): Specifies a directed linkage between two or more blocks. The type of the linkage can be either in space or time. If the linkage is in space (between blocks at different locations), LPFORM generates an associated flow-variable. If the linkage represents a time transformation, a transition variable is generated.

Because the block definition hierarchy can have multiple levels, only the relevant linkages need be specified by Link-Block statements. All linked blocks must be at the same level. In the Figure, three directed arrows are drawn between three blocks. The example shows the corresponding link_block statement.

If the default "link_mode" is explicit, the "directed_arcs" argument must contain a list of from-to node pairs as in the example. This corresponds to the user explicitly constructing the graph on the screen using a pointing device or simply typing in the pairs of blocks that are to be linked in response to a prompt. If the "link_mode" is "file", "directed_arcs" contains the name of a file containing the list of from_to pairs.

LPSPEC: **link_output_input**(block_set, common_commodity)

Screen Interface:

```
 ___________________________________________________________________________________________
| PROBLEM: energy_model        VERSION: 1   LAST UPDATE: 10/01/86  |  DATA:       |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|              |
|__________________________________________________________________| MODE: DATA   |
| LEVEL: 1  GRAPH: 1          CURRENT OP: LINK_OUTPUT_INPUT         |  REL:  r     |
|                                                                   |  TAB:  t     |
|                 ______________________________________________    |  PAR:  p     |
|                | BLOCK SETS:            COMMON COMMODITY:      |   |  SET: {s}    |
|                | source_________   |    raw_energy______       |   |______________|
|                | conversion______  |    processed_energy       |   | STRUCTURE:   |
|                | sink____________  |    ________________       |   |              |
|                |______________________________________________|   |  C-B:  []    |
|                                                                   |  L-B:  -->   |
|                                                                   |L-O-I: :--:   |
|                                                                   | B-IO: =[]=   |
|                                                                   |  D-I:  .:.   |
|                                                                   |  D-C:  |_|   |
|                                                                   |  D-R:  0=0   |
|                                                                   |  D-T:  ___   |
|                                                                   |  D-A:  =|-   |
|                                                                   |  C-M:  < >   |
|___________________________________________________________________|  REP:  ]]]   |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |  OPT   ^/v   |
|                                                                   |              |
|___________________________________________________________________|______________|
```

Figure I-8: Linking common commodities among Blocks

	Argument	Data Type
1:	block_set	list
2:	common_commodity	list

Example:

```
link_output_input([source,conversion,sink],[raw_energy,processed_energy]).
```

Explanation:

Link-Outputs-to-Inputs (L-O-I): Generates linkages within a given set of blocks to form a multi-commodity network. These linkages are made on the basis of common commodity flows. For example, a block with 'coal' as output commodity will be automatically linked to every block having coal as an input. Usually, the Block-Inputs-and-Outputs statement will have been used prior to this statement.

LPSPEC: **optimize**(optimize_direction, problem_name, obj_type, formulation_mode)

Screen Interface:

```
 ______________________________________________________________________________________
| PROBLEM: energy_model          VERSION: 1   LAST UPDATE:10/01/86  |  DATA:           |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|                  |
|___________________________________________________________________|  MODE:  DATA     |
| LEVEL: 1  GRAPH: 1          CURRENT OP: OPTIMIZE                  |  REL:   r        |
|         BLOCK: energy_model                                        |  TAB:   t        |
|         ____________________________________________               |  PAR:   p        |
|        | OPTIMIZATION                                |            |  SET:  {s}       |
|        | DIRECTION  :  MAXIMIZE    : ______          |            |__________________|
|        |            |  MINIMIZE    : X_____          |            | STRUCTURE:       |
|        | OBJ. TYPE  :  PROFIT      : ______          |            |                  |
|        |            |  COST        : X_____          |            |  C-B:   []       |
|        | MODE       :  SYMBOLIC    : X_____          |            |  L-B:   -->      |
|        |            |  DATA        : ______          |            |L-O-I:  :--:      |
|        |_____________________________________________|            | B-IO:  =[]=      |
|                                                                    |  D-I:   .:.      |
|                                                                    |  D-C:   |_|      |
|                                                                    |  D-R:   0=0      |
|                                                                    |  D-T:   ___      |
|                                                                    |  D-A:   =|-      |
|                                                                    |  C-M:   < >      |
|____________________________________________________________________|  REP:   ]]]      |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE []  |  OPT:   ^/v      |
|                                                                    |                  |
|____________________________________________________________________|__________________|
```

Figure I-9: Define Optimization Direction

	Argument	Data Type	Domain Set
1:	optimize_direction	atom	*optimization*
2:	problem_name	atom	
	Allied Schema: call_model.		
3:	obj_type	atom	*objective_function_type*
	Allied Schema: def_inventory, def_activity.		
4:	formulation_mode	atom	*formulation_mode*

Key: [2]

Domain Set Members:

optimization = {max, min}
objective_function_type = {profit, cost}
formulation_mode = {data, symbolic}

Example:

```
optimize(min,energy_model,cost,symbolic).
```

Explanation:

Optimize (OPT): Specifies the optimization direction (maximize or minimize) the type of the objective function (cost or profit) and formulation mode (symbolic or data). An Optimize statement is required in every formulation.

When the formulation mode is 'symbolic', the final formulation is in algebraic form and no data binding is involved. The data binding process is initiated by a value of 'data' for the formulation mode.

LPSPEC: **short_name**(model_name, name_type, long_name, short_name)

	Argument	Data Type	Domain Set
1:	model_name Allied Schema: call_model.	atom	
2:	name_type	atom	*name_type*
3:	long_name	atom	
4:	short_name	atom	

Key: [1, 2, 4]

Domain Set Members:
name_type = {index, var, coef}

Example:

```
short_name(energy_model,index,source,so).
```

Explanation:

Short-Name (SN): Specifies that an explicitly defined symbol dictionary is to be used in this formulation. The user specifies a problem (model) name, the type of the symbol (index, var, or coef), its full name, and short name. If this statement is not part of the LPSPEC problem specification, the short name will be generated either by asking the user in the formulation stage or by internally generating it according to some simple algorithm.

LPSPEC: **table**(table_name, table_index, table_type, content_mode, content_spec, unit)

Screen Interface:

```
 ____________________________________________________________________________________________
| PROBLEM: energy_model        VERSION: 1   LAST UPDATE:10/01/86  |  DATA:         |
| LOAD  SAVE  PROB-DATA  DATABASE  DICTIONARY  UP  DOWN  SOLVE  QUIT|               |
|_________________________________________________________________|  MODE: DATA   |
| LEVEL: 1  GRAPH: 0        CURRENT OP: TABLE                      |   REL:  r     |
|                                                                  |   TAB:  t     |
|                                                                  |   PAR:  p     |
|          | TABLE NAME   :  trans_cost_so_co               |      |   SET: {s}    |
|          | INDICES      :  source,conversion___           |      |_______________|
|          | TABLE TYPE   :  PROFIT        : ______         |      |  STRUCTURE:   |
|          |              |  COST          : X_____         |      |               |
|          |              |  UPPER BOUND   : ______         |      |   C-B:  []    |
|          |              |  LOWER BOUND   : ______         |      |   L-B:  -->   |
|          |              |  EXACT AMOUNT  : ______         |      |  L-O-I: :--:  |
|          | CONTENT MODE :  FILE          : X_____         |      |  B-IO: =[]=   |
|          |              |  EXPLICIT      : ______         |      |   D-I:  .:.   |
|          | CONTENT SPEC :  tcsc.dat______________         |      |   D-C:  |_|   |
|          | UNIT         :  ______________________         |      |   D-R:  0=0   |
|          |________________________________________________|      |   D-T:  ___   |
|                                                                  |   D-A:  =|-   |
|                                                                  |   C-M:  < >   |
|__________________________________________________________________|   REP:  ]]]   |
| BACK []  FORWD []  DELETE []  UNDELETE []  SHOW-DET []  ERASE [] |   OPT:  ^/v   |
|                                                                  |               |
|__________________________________________________________________|_______________|
```

Figure I-10: Defining transportation cost data Table

	Argument	Data Type	Domain Set
1:	table_name	atom	
2:	table_index	list	
3:	table_type	atom	*valid_coef_type*
4:	content_mode	atom	*table_content_mode*
5:	content_spec	file	
6:	unit	atom	

Allied Schema: def_inventory, def_activity.

Key: [1]

Domain Set Members:

valid_coef_type = {profit, cost, upper_bound, lower_bound, exact_amount}

table_content_mode = {file, explicit}

Example:

```
table(trans_cost_so_co,[source,conversion],cost,file,'tcsc.dat',#).
```

Explanation:

Table (TAB): Specifies that an explicitly defined table is to be used in the problem. Tables are needed when the optimization mode is "data" but can also be specified in "symbolic" mode. A "table" is a multi-dimensional array of numerical values. The "Table_index" argument defines its dimensions. Thus, in the example, the "trans_cost_so_co" table is a 2-dimensional array of cost coefficients with rows corresponding to the set of sources and columns corresponding to the set of conversions.

Two "content modes" are used in a Table statement to indicate the storage media. "File mode" indicates that the numerical values are stored in a file and the file specification is specified in the next slot (see Figure I-10). "Explicit mode" indicates that the numerical values are explicitly specified as a list at the next slot. If the unknown value "?" is specified together with explicit mode, the numerical values will be acquired interactively from the user during the consolidation stage of LPFORM.

The units in which the data is expressed (for example, units of product per unit of raw material) is stated in the "unit" argument.

II. Appendix: LPFORM Commands

LPFORM commands are used to control the interaction with the system. For example, they allow the user to specify the terminal type ("terminal"), to start the execution of the system ("run_problem"), to dump various system files ("dump_lpspec", "dump_template"), to maintain the LPSPEC definition ("lpspec"), and to maintain model templates ("template"). The "/" is used to specify the abbreviation of a command.

help

Access a help utility.

terminal/t

Allows user to specify the terminal type in order to improve the screen interface.

run_problem/r

Formulate the problem in either file or interactive mode.

lpspec

Access a system maintenance utility to list or modify the LPSPEC language.

dumplpspec/dl

Dump definitions of all LPSPEC statements into a file called "lpspec.dum".

template

Access the template maintenance utility to list existing templates and create new ones.

dumptemplate/dt

Dump a listing of all model templates from the LPFORM model base into a file called "lpform.dum".

ctrl-z

Quit LPFORM, and return to monitor level. Summary statistical information about the usage of the system is displayed.

III. Appendix: Model Template Library

A listing of all existing model templates can be dumped into a file by the command, "dump_template" or "dt":

```
| ?- dt.

* Dumping Template Library *

* The Template library have been dumped into file: lpform.dum.
```

Here we list all model templates (structure and symbol convention) currently available in LPFORM system as following:

```
PROBLEM/MODEL/FRAGMENT = exog_supply.

ROW\COL  X(i,j,k)        RHS
Use[i;k] +S{j}1[i;j;k]  < +s[i;k]


              * Symbol convention of exog_supply *

Set Reference:
SYMBOL:     SET NAME:
-----------------------------------------------------------------
k           : Commodity
i           : From_block
j           : To_block

Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
-----------------------------------------------------------------
X(i,j,k)  : FLOW(From_block,To_block,Commodity)

Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
-----------------------------------------------------------------
1[i;j;k]  : 1[From_block,To_block,Commodity]
s[i;k]    : Supply[From_block,Commodity]



PROBLEM/MODEL/FRAGMENT = exog_demand.

ROW\COL       X(i,j,k)        RHS
Supply[j;k] +S{i}a[i;j;k]  > +d[j;k]


              * Symbol convention of exog_demand *

Set Reference:
SYMBOL:     SET NAME:
-----------------------------------------------------------------
k           : Commodity
i           : From_block
j           : To_block
```

```
Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
------------------------------------------------------------------
X(i,j,k)  : FLOW(From_block,To_block,Commodity)

Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
------------------------------------------------------------------
a[i;j;k]  : Gain_or_loss[From_block,To_block,Commodity]
d[j;k]    : Demand[To_block,Commodity]



PROBLEM/MODEL/FRAGMENT = input_cons.

ROW\COL  X(k,j)         RHS
Use[k;i] +S{j}a[k;i;j]  < +s[k;i]


              * Symbol convention of input_cons *

Set Reference:
SYMBOL:     SET NAME:
------------------------------------------------------------------
k         : Block
i         : Input
j         : Output

Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
------------------------------------------------------------------
X(k,j)    : VOLUME(Block,Output)

Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
------------------------------------------------------------------
a[k;i;j]  : Tech_coef[Block,Input,Output]
s[k;i]    : Available_input[Block,Input]



PROBLEM/MODEL/FRAGMENT = transportation.

ROW\COL       X(i,j,k)            RHS
OBJ=          +S{i;j;k}c[i;j;k]   MIN
Use[i;k]      +S{j}1[i;j;k]       < +s[i;k]
Supply[j;k] +S{i}a[i;j;k]         > +d[j;k]


            * Symbol convention of transportation *

Set Reference:
SYMBOL:     SET NAME:
------------------------------------------------------------------
k         : Commodity
i         : From_block
j         : To_block

Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
------------------------------------------------------------------
X(i,j,k)  : FLOW(From_block,To_block,Commodity)
```

```
Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
-----------------------------------------------------------------
c[i;j;k]  : Trans_cost[From_block,To_block,Commodity]
1[i;j;k]  : 1[From_block,To_block,Commodity]
s[i;k]    : Supply[From_block,Commodity]
a[i;j;k]  : Gain_or_loss[From_block,To_block,Commodity]
d[j;k]    : Demand[To_block,Commodity]
```

```
PROBLEM/MODEL/FRAGMENT = product_mix.

ROW\COL   X(k,j)          RHS
OBJ=      +S{k;j}p[k;j]   MAX
Use[k;i]  +S{j}a[k;i;j]   < +s[k;i]
```

```
              * Symbol convention of product_mix *

Set Reference:
SYMBOL:     SET NAME:
-----------------------------------------------------------------
k         : Block
i         : Input
j         : Output

Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
-----------------------------------------------------------------
X(k,j)    : VOLUME(Block,Output)

Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
-----------------------------------------------------------------
p[k;j]    : Profit[Block,Output]
a[k;i;j]  : Tech_coef[Block,Input,Output]
s[k;i]    : Available_input[Block,Input]
```

```
PROBLEM/MODEL/FRAGMENT = inventory.

ROW\COL          I(t-1)    X(t)   I(t)    RHS
Inventory[time]  +1[t-1]   +1[t]  -1[t]   = +0[t]
```

```
              * Symbol convention of inventory *

Set Reference:
SYMBOL:  SET NAME:
-----------------------------------------------------------------
t       : T
t-1     : T-1

Activity Reference:
SYMBOL:  ACTIVITY (VARIABLE):
-----------------------------------------------------------------
I(t-1)  : INVENTORY(T-1)
X(t)    : VOLUME(T)
I(t)    : INVENTORY(T)
```

```
Coefficient Reference:
SYMBOL:  COEFFICIENT (DATA):
-----------------------------------------------------------------
1[t-1] : 1[T-1]
1[t]   : 1[T]
0[t]   : 0[T]
```

```
PROBLEM/MODEL/FRAGMENT = process_selection.

ROW\COL      X(k,j,m)            RHS
OBJ=         +S{k;j;m}c[k;j;m]   MIN
Use[k;i]     +S{j;m}t[k;i;j;m]   < +a[k;i]
Supply[k;j] +S{m}1[k;j;m]        > +b[k;j]
```

```
          * Symbol convention of process_selection *

Set Reference:
SYMBOL:       SET NAME:
-----------------------------------------------------------------
k            : Block
m            : Form_mode
i            : Input
j            : Output

Activity Reference:
SYMBOL:       ACTIVITY (VARIABLE):
-----------------------------------------------------------------
X(k,j,m)     : VOLUME(Block,Output,Form_mode)

Coefficient Reference:
SYMBOL:       COEFFICIENT (DATA):
-----------------------------------------------------------------
c[k;j;m]     : Production_cost[Block,Output,Form_mode]
t[k;i;j;m]   : Tech_coef[Block,Input,Output,Form_mode]
a[k;i]       : Available_input[Block,Input]
1[k;j;m]     : 1[Block,Output,Form_mode]
b[k;j]       : Minimum_production[Block,Output]
```

```
PROBLEM/MODEL/FRAGMENT = general_lp_max.

ROW\COL X(j)          RHS
OBJ=    +S{j}p[j]     MAX
Use[i]  +S{j}a[i;j]   < +b[i]
```

```
           * Symbol convention of general_lp_max *

Set Reference:
SYMBOL:  SET NAME:
-----------------------------------------------------------------
i       : I_set
j       : J_set

Activity Reference:
SYMBOL:  ACTIVITY (VARIABLE):
-----------------------------------------------------------------
X(j)    : X(J_set)
```

```
Coefficient Reference:
SYMBOL:  COEFFICIENT (DATA):
-----------------------------------------------------------------
p[j]    : P[J_set]
a[i;j]  : A[I_set,J_set]
b[i]    : B[I_set]
```

```
PROBLEM/MODEL/FRAGMENT = general_lp_min.

ROW\COL    X(j)           RHS
OBJ=       +S{j}c[j]      MIN
Supply[i] +S{j}a[i;j]   > +b[i]
```

```
            * Symbol convention of general_lp_min *

Set Reference:
SYMBOL:  SET NAME:
-----------------------------------------------------------------
i       : I_set
j       : J_set

Activity Reference:
SYMBOL:  ACTIVITY (VARIABLE):
-----------------------------------------------------------------
X(j)    : X(J_set)

Coefficient Reference:
SYMBOL:  COEFFICIENT (DATA):
-----------------------------------------------------------------
c[j]    : C[J_set]
a[i;j]  : A[I_set,J_set]
b[i]    : B[I_set]
```