View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

N89-21757

1988

NASA/ASEE Summer Faculty Fellowship Program

Marshall Space Flight Center
The University of Alabama

Development of a Prototype Commonality Analysis Tool for use in Space Programs

Prepared by:	Dorian P. Yeager
Academic Rank:	Associate Professor
University and Department:	The University of Alabama Computer Science
NASA/MSFC:	
Laboratory:	Systems Analysis and Integration
Division:	Space Station Systems
Branch:	Systems Integration
NASA Colleague:	L. Dale Thomas
Date:	August 12, 1988
Contract No.	The University of Alabama NGT-01-002-099

ABSTRACT

A software tool to aid in performing commonality analyses, called Commonality Analysis Problem Solver (CAPS), was designed, and a prototype version (CAPS 1.0) was implemented and tested. CAPS 1.0 runs in an MS-DOS or IBM PC-DOS environment. CAPS is designed around a simple input language which provides a natural syntax for the description of feasibility constraints. It provides its users with the ability to load a database representing a set of design items, describe the feasibility constraints on items in that database, and do a comprehensive cost analysis to find the most economical substitution pattern.

ACKNOWLEDGMENTS

The author wishes to thank NASA and ASEE for their support of a fine program. My experiences here have been extremely rewarding. Thanks also to Mike Freeman and Ernestine Cothran for their capable administration and their personal involvement in the program.

Special thanks go to my NASA colleague, L. Dale Thomas, without whose help the fundamental nature of the problem would not have been evident.

Thanks go also to Charlie Cothran and the crew at EL83, now EJ12, who again made me feel very much at home and provided a pleasant and productive work environment.

INTRODUCTION AND OBJECTIVES

Commonality is an attribute of large systems. A system is said to incorporate a high level of commonality if there are few instances where functionally similar but unique designs are incorporated into the system, and component parts are designed with a high level of functionality, so that each separately designed component may be employed in multiple positions within the system.

Commonality Analysis is a process, as yet only rather poorly defined, for assessing the level of commonality in the design of a large system and, if necessary, making recommendations for increasing that level of commonality. The major considerations in making such recommendations are of necessity economic ones, and it must be true in some sense that the system will be more cost-effective if the recommended changes are made.

An automatic tool for structuring the commonality analysis process will always fall short of the goal of providing a comprehensive framework. There may indeed be more art than science involved in the process. A good tool is very definitely needed, nonetheless. A first step in the direction of providing such a tool is represented by the SCAT program developed under Work Package 1 (see MSFC 1987). An assessment of the benefits and drawbacks of that program is given in last year's report, along with recommendations for improvement. Some of those recommendations are now being incorporated into SCAT.

This report describes an alternative approach to commonality analysis. The Commonality Analysis Problem Solver, or CAPS, is conceived as a software tool with two major features: (1) more intelligent algorithms for investigating substitution strategies, and (2) a language for precisely describing substitution constraints. The mathematics which suggested the CAPS algorithms was discovered by the author during his 1987 tenure as a Summer Faculty Fellow. The design and implementation of the CAPS language was the purpose of this year's work, and its description is the subject of this report.

CAPS 1.0

CAPS 1.0 is a demonstration prototype consisting of about 2800 lines of C code developed in a four-week period. Some limitations were imposed by the short development time. Notable among these limitations are the restriction to numeric data fields only, the rather simplified input data file format, and the inability to short-circuit prohibitively long cost analyses. The last-mentioned drawback derives from CAPS 1.0's insistence on providing the absolute optimum solution, no matter what the cost in computation time. Future development of CAPS will give it the intelligence to choose and apply a suboptimal solution strategy when such a strategy is indicated. Future releases will also extend CAPS' functionality and improve its user interface.

DATA FILES

A CAPS 1.0 data file is organized as a series of lines of text. The first line contains a series of field names (attributes) applicable to records in the data base. The syntax of field names must conform to the same constraints as that of CAPS variable names (see below). The remainder of the file is a series of records, coded as one line of text per record. Each record consists of a series of numeric constants, and each record must supply a value for each attribute. The form of the file is therefore exactly like a table with headings, except no alignment conventions need be observed. The data file is a simple ASCII text file, such as could be constructed with any text editor, including the DOS EDLIN editor, or with any word processor capable of exporting documents to an ASCII file. Alternatively, a CAPS 1.0 file can easily be generated as a report file by any relational database. An example of a data file is given below:

ddt&e	prod	weight	volume	quantity
46.166	7.694	9.222	19.2	1
49.374	8.229	10.375	21.6	1
67.833	11.306	17.292	36.0	4
71.86	11.977	19.021	39.6	4
92.819	15.47	28.244	58.8	2
355.772	59.295	232.289	483.6	2
366.685	61.114	243.241	506.4	6
378.240	63.040	253.616	528.0	3
464.314	77.386	348.722	726.0	4

Table 1: Typical CAPS data file.

In the above example, there are five fields and nine records. Note that no quote marks are used to delimit the field names. In CAPS 1.0, no input line can be longer than 300 characters.

THE CAPS 1.0 USER INTERFACE

CAPS 1.0 uses a command line input scheme. When the program initially loads, it displays the prompt:

```
>>
```

CAPS then recognizes commands typed in by the user and responds to each command in turn by displaying (a) the value of an expression, (b) a graph of the data set, or (c) a report of current status or of an action taken. There are 16 input forms, as follows:

```
<expression>
load <file_name>
fields
for <logical expression> allow <logical expression>
for <logical expression> disallow <logical expression>
allow all
disallow all
status
define <variable> as <expression>
add <defined variable>
graph <field_name_1> vs <field_name_2>
cost
learn <constant>
learn
take <file_name>
quit
```

We will discuss each of these in turn below.

The following keywords are <u>reserved</u> <u>words</u> in CAPS 1.0. Their meanings are fixed and cannot be changed by the user, which means that they are not available as field names in data files nor as variable names.

abs	add	all	allow	and	as	cos	cost
define	disallow	exp	fields	for	graph	int	item
learn	ln	load	log	not	or	quit	sin
sqrt	status	take	tan	vs			

CAPS EXPRESSIONS

The primary use for CAPS expressions is the description of feasibility constraints for the substitution strategy in commonality analyses. In order to describe such constraints it must be possible to communicate any kind of relationship between two items in a data file. Expression syntax is essential for such communication.

CAPS expressions use standard expression syntax, like that used in FORTRAN and BASIC. Besides the standard arithmetic, relational, and logical operators, several essential functions are implemented, such as sine, cosine, tangent, square root, absolute value, and integer part.

Spaces are ignored wherever they occur in CAPS expressions, except that spaces are not allowed in variable names or numeric constants.

NUMERIC CONSTANTS

CAPS numeric constants may be integers or real numbers in fixed or floating point form, and may be signed or unsigned. Only decimal constants are allowed. Examples are as follows:

```
25      -340    .7     0       0.789      8.98E23
345.    -.001   1e2    1e-2    7.38E-21   500000
```

Very few restrictions of commonly accepted syntax are applicable. The range of values accepted is -3.4E38 to 3.4E38. Numbers smaller in magnitude than 1.2e-38 underflow to zero. Precision is about seven decimal digits. CAPS does not accept floating point constants with more than two digits in the exponent. Thus 1e010 is not a valid constant, but 1e01 and 1e10 are valid.

In CAPS, all numeric quantities are represented internally as real (floating point) numbers.

VARIABLE NAMES

CAPS variable names may be of any length, must begin with an alphabetic character, and must consist of alphabetic characters, decimal digits, underline characters (_), and ampersands (&). CAPS is NOT case-sensitive. Thus the names VOL, Vol, vol, and VOl all refer to the same variable.

ARITHMETIC OPERATORS

CAPS uses the FORTRAN set of arithmetic operators, with the same meaning and the same precedence rules, given below:

Operator	Precedence	Meaning
+	Lowest	Real number addition
-	Lowest	Real number subtraction
*	Intermediate	Real number multiplication
/	Intermediate	Real number division
**	Highest	Exponentiation

Since all numbers are represented internally in floating point, an operation performed on two integers will not necessarily yield an integer. For example, the expression 1/3 evaluates to 0.3333334, not 0 as it would in FORTRAN.

Exponentiation is allowed whenever it makes sense and the result is a real number. Thus 0**0 is not permitted, nor is (-1)**0.5, but (-2)**(-3) is permitted and yields -0.125.

The order of evaluation of operators with the same precedence depends on their **associativity.** The associativity of +, -, *, and / is from left to right, whereas that of ** is from right to left. This is exactly the convention observed by FORTRAN. Associativity and precedence rules may be overridden with the use of parentheses. Some examples follow:

Expression	Value
25 - 12 + 2	15
25 -(12 + 2)	11
5/ 2*2	5
5/(2*2)	1.25
2**2**3	256
(2**2)**3	64

Negation, or unary minus, is a separate operation, different from subtraction. Its precedence is higher than that of the additive operators + and - and lower than that of the multiplicative operators * and /. Thus the expression 4 + -5 is allowed, but the expression 2**-2 is not valid CAPS syntax. Parentheses are required around a negated CAPS expression which is used as a right-hand input to any operation other than + or -. Thus in the example it is necessary to use the syntax 2**(-2).

PRIMITIVE FUNCTIONS

CAPS 1.0 supports the following set of primitive functions:

Function Name	Meaning
sqrt	Square root
abs	Absolute value
sin	Trigonometric sine
cos	Trigonometric cosine
tan	Trigonometric tangent
ln	Logarithm base e
exp	Exponential function (e^x)
log	Logarithm base 10
int	Integer part

Most of the above have the obvious meaning, but a few explanations are needed. (1) The three trigonometric functions take their operands in radians, not degrees. (2) CAPS intercepts any attempts to use one of these functions with an inappropriate argument. For example, CAPS will print its own error message in response to sqrt(-2), and will display again the ">>" input prompt. In contrast, some other types of floating point errors, such as floating point overflow, will not be intercepted by CAPS and will cause immediate program termination. (3) The 'int' function returns the integer part of a number. Examples follow:

Expression	Value
int(5.7)	5
int(-5.7)	-5
int(3)	3

RELATIONAL OPERATORS

The CAPS relational operators are =, <>, <, >, <=, and >=. They are interpreted, respectively, as 'is equal to', 'is not equal to', 'is less than', 'is greater than', 'is less than or equal to', and 'is greater than or equal to'. The relational operators all have lower precedence than the arithmetic operators. Since the output of a relational expression is not normally given in turn as input to another relational operation (for example, 5<7<9 is not a valid CAPS expression), associativity rules are not needed.

A CAPS relational expression always returns a value of 1 or 0, meaning true or false respectively. Thus the expressions

5<7 and -7.3<-5.9 return the value 1 and the expressions 5>=7, 8=9, 1e2<99, and 3<>3 all return the value 0.

Parenthesized expressions are fair game for input into any CAPS relational operation. Thus (5<7)<9 is legal and yields in turn first 1<9, then 1, i.e. true.

BOOLEAN OPERATORS

The boolean operators are given, along with their precedences, in the table below:

Operator	Precedence
or	lowest
and	intermediate
not	highest

The precedence rules were chosen in order to observe standard conventions with almost universal acceptance. All boolean operators have lower precedence than all relational operators and all arithmetic operators. Thus

```
x < 3 and not y>=4 or p = q
```

is equivalent to

```
((x < 3) and (not (y>=4))) or (p = q)
```

All numeric results have a true-false interpretation. Very simply, zero is false and all nonzero numbers are true. Thus '8 or 0' has the value 1, or true, whereas '8 and 0' has the value 0, or false.

EVALUATION ORDER

Evaluation order of a CAPS expression is very precisely specified, as follows: (1) with the exception of unbound variables used as operands of '=', all operands of an operator are evaluated to yield a numeric value before the operation itself is executed; and (2) the left-hand operand is always evaluated before the right-hand operand.

VARIABLE BINDINGS AND THE ASSIGNMENT SIDE EFFECT

A CAPS variable, just like a variable in any standard programming language, may be associated with a value. Until

such an association takes effect, the variable is said to be unbound. A variable may be bound to a value using the = operator.

The = operator, being a relational operator, always returns as its value the number 1 or 0, indicating a true or false result. If one of the operands of = is an unbound variable, two things occur. First, the value of the other operand is computed and bound to the variable. Second, the value 1 is returned, since as a result of the binding that has taken place the two sides of the = operator have identical values.

Consider, for example, the expression below, in which the variable x is initially unbound:

```
4 = x and x > 3
```

The left-hand operand of the 'and' operation (i.e. 4 = x) is first evaluated, having the dual effect of assigning 4 to x and returning a true value. The right hand operand is then evaluated as 4 > 3, also yielding a true value. Since both operands are true, the 'and' operation also yields a true value, and along the way x picks up its binding to the value 4.

CALCULATOR OPERATION

Besides its primary use for commonality analyses, CAPS may be used as an interactive calculator. This feature was included in order to allow the user to experiment with CAPS expression evaluation to better understand its semantics. Calculator operation is achieved when the user types in an expression and CAPS evaluates the expression and displays its value. Consider, for example, the following series of CAPS inputs and responses:

```
>>principle = 40000
1.000000
>>rate = 0.10
1.000000
>>rate/12 = i
1.000000
>>principle*i / (1 - (1+i)**(-360)) = payment
1.000000
>>payment
351.028687
>>
```

Notice that the first four inputs are assignments, which always return the value 1. The last input is a single variable name,

'payment'. CAPS checks the symbol table for its current value, and displays that value as its response.

THE SYMBOL TABLE

The central data structure internal to CAPS, used for maintaining current variable bindings, is its symbol table. During a CAPS interactive session, all symbols which have any meaning to CAPS are represented in the symbol table. This includes the keywords listed above as well as all variables which have been used in previous commands and all field names from the currently loaded data set.

When CAPS is initially invoked, the only symbols represented in its symbol table are the keywords. As variables are introduced they are stored in the symbol table along with their bindings, if any. Once a variable is given a binding, that binding cannot be changed until a new data file is loaded. When a new file is loaded, the symbol table is cleared of all symbols except for the keywords and the field names for that file.

THE 'LOAD' COMMAND

The syntax of the load command is

    load <file name>

The rules of formation for CAPS 1.0 file names are exactly like those for CAPS variables. The DOS file name extension for CAPS 1.0 data files is always 'dat', and is provided automatically. The user is not allowed to provide an alternate extension. Since DOS requires that file names be limited to eight characters, CAPS automatically truncates any excess trailing characters from names which are longer than this limit.

CAPS' actions upon receiving a load command are to (a) clear the symbol table of all variable bindings, (b) read the field names from the file and install them in the symbol table so that they may not be used for variable names, and (c) read the data from the file into an internal table.

If during the loading of a file an error is discovered by CAPS in the way the file is organized, CAPS displays an error message and aborts the load process. A side effect of any load is the clearing of the field names of the last loaded file from

the symbol table. This means that after an unsuccessful load attempt CAPS will have no memory of the last file successfully loaded.

When a file is successfully loaded, CAPS reports the number of fields and the number of items of data. Following is an example.

```
>>load motors
12 fields, 34 items.
Load successfully completed.
>>
```

LIMITATIONS: CAPS 1.0 accommodates a maximum of 75 items and 256 fields.

THE 'FIELDS' COMMAND

The fields command requests a listing of all the field names for the currently loaded data set. For example:

```
>>fields
ddt&e  unitcost  volume
3 fields.
>>
```

THE FEASIBILITY MATRIX

There are three internal data structures which CAPS uses to reflect its current state. The first is the symbol table, and the second is the table containing the currently loaded data set. The third is the feasibility matrix, called alpha, which determines which substitutions are currently permitted. Alpha is boolean and is doubly indexed over the items in the data set. Thus $alpha_{i,j}$ is true if item i is allowed to substitute for item j, false otherwise.

COMMANDS USED TO ALLOW AND DISALLOW SUBSTITUTIONS

After a file has been loaded containing information on the items to be subjected to commonality analysis, it is necessary to communicate to CAPS the allowable patterns of substitution. This communication is achieved as follows: Initially, CAPS assumes no substitutions are allowed. This means that the feasibility matrix has 'true' entries only on the diagonal. Then, using a series of 'for', 'allow all', and 'disallow all' commands, the user modifies the feasibility matrix. When the

user has adequately described all allowable substitutions, s/he requests a cost analysis with the 'cost' command.

THE 'FOR' COMMANDS

The for commands are:

```
for <expression> allow <expression>
```

and

```
for <expression> disallow <expression>
```

The first allows new substitutions. It does not, of course, require that those substitutions be made. It simply tells the cost analysis portion of the program that it is permitted to consider the indicated substitutions as options when it performs its analysis. The second form forbids the cost analyzer to consider a given set of substitutions.

Examples:

```
for capacity=10 allow capacity=12
for boys_type and speeds=3 allow boys_type and (speeds=3 or
    speeds=10)
for not flammable disallow flammable
for portable disallow not portable
```

In the above, 'capacity', 'boys_type', 'speeds', 'flammable', and 'portable' are field names. The first statement allows all items for which the value in the field 'capacity' is 12 to substitute for all items having the value 10 in that field. The second can be summarized rather clearly if we think in terms of a data base representing bicycles. If for a given bicycle in the data base the boolean field 'boys_type' is set to true and the value of 'speeds' is 3, then any bike used as a substitute for that bike must also be a boys bike and must be either a 3-speed or a 10-speed.

The last two examples also involve boolean fields, but are phrased in the negative. Example 3 says that a flammable item may not substitute for an item with is not flammable, and example 4 says that an item which is not portable may not substitute for an item which is portable.

Note that disallowing substitutions has no effect until some substitutions have first been allowed, since CAPS initially assumes no substitutions are valid.

THE 'ALLOW ALL' AND 'DISALLOW ALL' COMMANDS

The command 'allow all' tells CAPS all substitutions are valid, whereas 'disallow all' clears the feasibility matrix (except, of course, for the diagonal) and puts CAPS back in its original state with respect to the currently loaded table.

USE OF VARIABLES

To provide as general a facility as possible for the description of substitution constraints, two alternative kinds of variable bindings in for statements (other than simple bindings to numerical values) may be employed. Firstly, a variable may be bound to a field name in the currently loaded table, and secondly it may be bound to an expression involving one or more field names. The example below will illustrate:

```
for size = x allow size >= x
```

This example basically says that any item of a given size may be replaced by an item of that size or larger. The actual procedure CAPS goes through is as follows.

```
Clear from the symbol table all variables having numeric
bindings.
For each item i in the table:
    Bind x to the size of item i
    For each item j in the table:
        Allow j to substitute for i provided the size of j
        is
          at least x
    Free x's binding
```

Consider now a second example:

```
for x=power/weight and x>=1.5 allow y=power/weight and y>=x
```

Here two variables named x and y are employed. The for statement says that if x is 'power' divided by 'weight' for item i, x is at least 1.5, and y is 'power' divided by 'weight' for item j, then j can replace i provided y is at least as large as x. The following is an alternate formulation of the same set of allowed substitutions, using only a single variable:

```
for x=power/weight and x>=1.5 allow power/weight>=x
```

THE KEYWORD 'ALL'

When used in 'for' statements, the keyword 'all' refers to all items in the currently loaded data set. For example:

```
for cylinders=4 allow all
```

allows all items in the data set to substitute for any item having the value 4 in its 'cylinders' field.

In particular, 'for all allow all' has the same effect as 'allow all', and 'for all disallow all' has the same effect as 'disallow all'.

THE 'DEFINE' COMMAND

The 'define' command has the form:

```
define <variable> as <expression>
```

This statement binds the variable to an expression, not to a value. It allows the user to establish meanings for variables which go beyond simple numeric bindings and which allow more brevity and flexibility in the coding of meaningful for statements. Also, since the for statement destroys numeric bindings, the define statement provides a way of protecting bindings from the effects of a 'for'.

Examples are as follows:

```
define pi as 3.1415926
define totalcost as
   weight*cost_per_pound+volume*cost_per_cubic_foot
```

Note in particular the first example, where pi is bound to an expression consisting only of the constant 3.1415926. Although the expression will always yield that constant as its value, it is nevertheless stored internally as an expression and is therefore protected from the effects of for statements. A defined variable may be used in subsequent expressions in for statements. If the variable being defined is not defined in terms of field names, it may be used in calculator mode. In that case, a numerical result is computed for the variable using the current bindings of other variables in the expression. The following CAPS dialog will illustrate.

```
>>define pi as 3.141593
Definition successful.
>>define angle1 as pi/6
Definition successful.
```

```
>>sin(angle1)
0.500000
```

Since 'define' does not immediately evaluate the expression provided as its second operand, but waits until the defined variable is used in calculator mode or in a for statement, the following is also possible.

```
>>define angle1 as pi/6
Definition successful.
>>define pi as 3.141593
Definition successful.
>>sin(angle1)
0.500000
```

The for statement above can be rephrased using a define statement, as follows:

```
define rel_power as power/weight
for x=rel_power and x>=1.5 allow rel_power>=x
```

THE KEYWORD 'ITEM'

When used in 'define' and 'for' statements, the keyword 'item' refers to an internal counter used to number the items in the table. Items in the table are automatically numbered from 1 to n, where n is the total number of items in the table. The status of 'item' is similar to that of a field name, but its value is not explicitly coded into the table. Examples:

```
for item=7 allow all
for item=x allow item>x
define even as int(item/2)*2 = item
```

The first example allows all items to replace the seventh item in the table. The second allows any item to be replaced by any item following it in the table. The third associates the variable 'even' with an expression which evaluates to true if and only if the item is numbered with an even number.

THE 'STATUS' COMMAND

The status command reports on the current status of the feasibility matrix by indicating which items each element of the data set is currently allowed to substitute for.

Example:

```
>>load data1
4 fields, 5 items.
Load successfully completed.
>>status
Allowable substitutions:
   1 -> {1}
   2 -> {2}
   3 -> {3}
   4 -> {4}
   5 -> {5}
>>for item=x allow item>x
10 new substitutions allowed.
>>status
Allowable substitutions:
   1 -> {1}
   2 -> {1,2}
   3 -> {1,2,3}
   4 -> {1,2,3,4}
   5 -> {1,2,3,4,5}
>>
```

HISTORY SENSITIVITY

The user of CAPS should always be aware that the purpose of a single for command is to augment or restrict the set of allowable substitutions, not to make an isolated declaration. CAPS is designed to communicate these allowed substitutions via a series of history-sensitive operations, the results of which are recorded in the feasibility matrix alpha.

Consider the following set of commands, involving valves:

```
define rel_diff as abs(diameter-x)/x
for diameter=x allow rel_diff < 0.05
for gas disallow liquid
```

The effect of the above is radically different from the following:

```
for gas disallow liquid
define rel_diff as abs(diameter-x)/x
for diameter=x allow rel_diff < 0.05
```

Note that the only difference in the two sets of commands is the order in which the commands were given. But because of the history-sensitive nature of CAPS, the results are quite different. In the first example no liquid valve will be allowed to substitute for a gas valve. But in the second

example two valves whose diameters differ by 5% or less will be allowed to substitute one for the other, regardless of whether they are liquid or gas valves.

THE 'ADD' COMMAND

The add command is used for adding new fields to a loaded data set. A variable may be used to add such a field only if it was bound using a 'define' statement. An example follows:

```
add rel_power
```

Here the variable rel_power is added as a new field. The mechanics of this process are simple - the size of the internal table used to hold the data is increased by one column, and data is placed in that column by stepping through the table and computing the value of the expression associated with rel_power for each data item in the table.

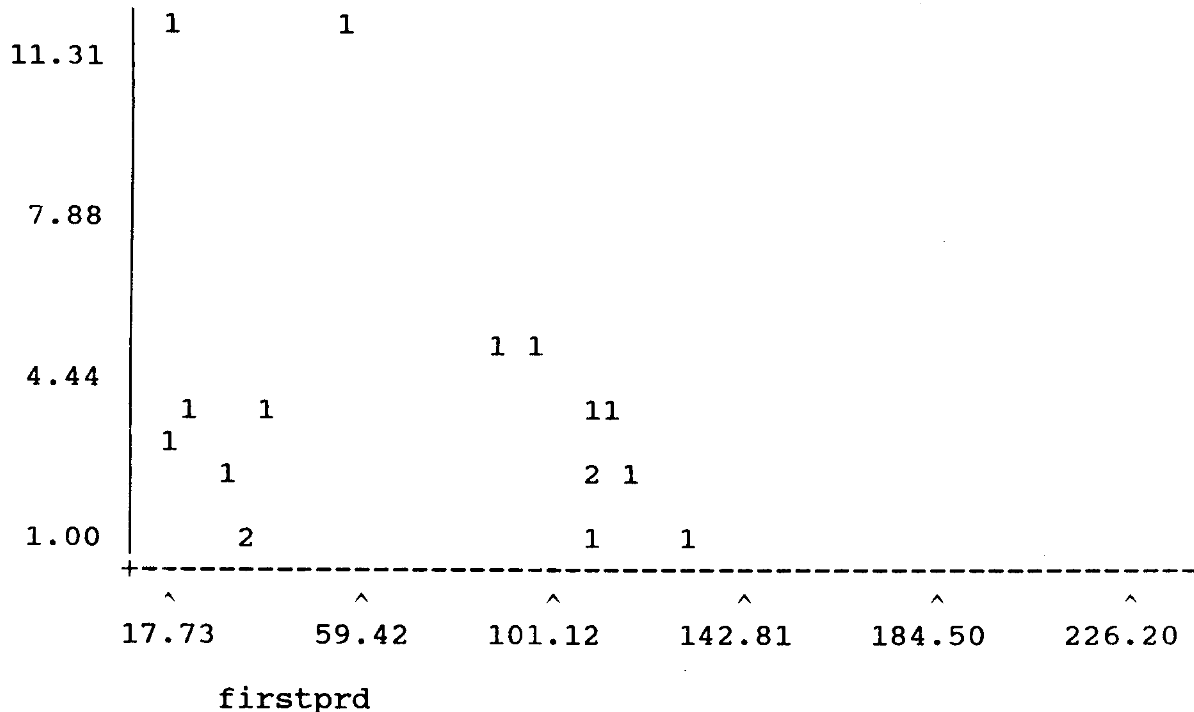
Note that the variable is now no longer associated with the expression. It has become a field name. In database terms it is a derived attribute based on the values of other attributes, or fields.

There are three reasons for converting defined variables into field names with the 'add' command. The first is increased speed of execution of 'for' commands. If a defined variable is used in the second operand of a for statement on a table with 30 elements, then the expression associated with that variable will be evaluated 900 times. Expression evaluation is much more expensive than table lookup, and installing the variable as a derived attribute requires only 30 evaluations of the expression. The second reason for adding new fields is to be able to use those fields in graphs (see below). The third reason is to provide necessary parameters for a cost analysis.

THE 'GRAPH' COMMAND

The graph command allows the user to take graphic snapshots of the data. CAPS 1.0 uses the text screen to provide a low-resolution graph of the data set. Two field names are supplied as parameters for the purpose of labelling the horizontal and vertical axes. An example follows.

```
>>graph firstprd vs quantity
```




```
>>
```

In the example, there are two items having quantity = 1 and having a value for firstprd between 17.73 and 59.42. There are five with quantity between 2 and 4 and firstprd between 101.12 and 142.81. CAPS 1.0 displays a digit between 1 and 9 in a given position if there are fewer than ten items falling into that position of the graph. If there are more than 9 data points, CAPS displays an asterisk (*) in that position.

THE 'COST' COMMAND

The cost command requests a cost analysis. In order for a meaningful cost analysis to be conducted, CAPS must have been supplied with a loaded data set containing one of the following sets of attributes:

1. ddt&e, quantity, unitcost, and firstprd.
2. ddt&e, quantity, and firstprd.
3. ddt&e, quantity, and unitcost.

CAPS must also have been supplied with a valid feasibility matrix via a series of 'for', 'allow all', and 'disallow all' commands.

The ddt&e field is the design, development, test, and engineering costs associated with the item. It is an "up front" cost, paid only once no matter how many copies of the item are to be produced. The ddt&e field is essential for any kind of cost analysis in CAPS 1.0.

The quantity field is the total number of copies of the item which will be produced. For manufacturing applications this tends to be quite large, but for space programs it is usually a relatively small number. Like the ddt&e field, the quantity field is essential for CAPS 1.0 cost analyses.

The unitcost field is the sum of all costs associated with each new copy of an item which are not subject to a learning curve. Its value is multiplied by the value in the quantity field. The firstprd field is the first product cost of an item. If this field is present in the data set, CAPS 1.0 will use the learning curve cost function.

The two cost functions employed by CAPS 1.0 are:

1. The linear cost function is used if firstprd is not present. With this model the cost associated with a given item is

   ddt&e + quantity*unitcost

2. The learning curve cost function is used if firstprd is present. Here the cost associated with a given item is

$$\text{ddt\&e} + \text{quantity}*\text{unitcost} + \text{firstprd} * \sum_{i=1}^{n} i^{t},$$

where n = quantity and t = ln(L/100)/ln(2). Here L is the learning curve parameter, adjustable in CAPS via the 'learn' command.

THE 'LEARN' COMMAND

It is possible to communicate the value of the learning curve parameter to CAPS' cost analysis component. This is a percentage, usually around 85%, but the value varies with the type of item being subjected to analysis. For example,

```
learn 90
```

sets the learning curve at 90%. CAPS will assume a learning curve of 85% until told to change that value.

If the learn command is used without an argument, CAPS displays the current value of the learning curve parameter.

THE 'TAKE' COMMAND

The 'take' command allows the CAPS user to prepare a series of commands and store them in an ASCII file. This avoids having to reenter the same or nearly the same series of commands each time s/he subjects the same data set to a cost analysis. For example, in response to the command

```
>>take bicycles
```

CAPS reads the commands in the file 'bicycles.tak' and executes them one by one. The 'tak' extension is always assumed.

INTERPRETING THE COST ANALYSIS OUTPUT

Following is a typical CAPS 1.0 cost analysis:

```
>>cost
Time required for an exhaustive analysis:  approximately
280 seconds.
CAPS 1.0 will find the optimum solution in approximately 3
seconds.
Proceed? y
Working.....................
Item      Replaces            Replacement Cost    Unique Cost
  3       {1,2,3}
                                        326.77         401.09
  4       {4}
                                        228.97         228.97
  7       {5,6,7,8}
                                       3389.18        4299.85
--------------------------------------------------------------
TOTALS                                 3944.92        4929.91
>>
```

The analysis above is for a data set of eight items. CAPS begins by giving an estimate of the time which would be required for an exhaustive analysis, followed by an estimate of the time for it to do its own analysis. The second number is, of course, never greater than the first. It is, however, quite often much smaller because CAPS employs its knowledge about the linear cost function and the learning curve cost function to attempt to reduce the number of potential solutions to be examined. (See Yeager, 1987a and 1988) The estimates are very rough, and are based on IBM PC/AT run times and the complexity of the algorithms being used.

In the example, CAPS recommends that only three of the items, items 3, 4, and 7, be produced. Items 1 and 2 are to be replaced by item 3, and items 5, 6, and 8 by item 7. Item 4 is a unique item. In making its recommendation, CAPS has followed the feasibility constraints placed on the data by the user in previous commands.

CAPS 1.0 displays the "Working...." message to give the user an indication of the progress of the analysis. Each dot ('.') represents three potential solutions (partitions) examined.

CAPS 1.0 cost figures are for comparison only. CAPS is intended as a tool for Commonality Analysis, not for projection of costs. It is, of course, true that the better the cost estimates provided to the 'ddt&e', 'unitcost', and 'firstprd' fields, the more accurate will be the recommendations of CAPS.

THE 'QUIT' COMMAND

The quit command returns control to DOS. Since no provision is made in the CAPS 1.0 prototype for saving results of an analysis to a file, the user is advised to use 'take' files to collect all relevant commands pertaining to a given data base, and to direct output to the printer during important analyses.

CASE STUDY #1

The first case study is for a set of storage tanks, the data for which appears in Table 1. The tanks appear in order by size, and size is the only determinant for substitutability. If the data is stored in file "tanks1.dat", the following CAPS dialogue yields a linear cost analysis.

```
>>load tanks1
5 fields, 9 items.
Load successfully completed.
>>fields
ddt&e  prod  weight  volume  quantity
5 fields.
>>define unitcost as prod+weight+volume
Definition successful.
>>add unitcost
Field successfully added.
>>for item=x allow item>x
36 new substitutions allowed.
>>cost
```

```
Time required for an exhaustive analysis:  approximately  630
seconds.
CAPS 1.0 will find the optimum solution in approximately    1
seconds.
Proceed? y
Working.......
Item       Replaces                  Replacement Cost    Unique Cost
  2        {1,2}
                                          129.78              171.86
  4        {3,4}
                                          636.64              680.48
  5        {5}
                                          297.85              297.85
  8        {6,7,8}
                                         9669.46            10049.56
  9        {9}
                                         5072.75             5072.75
------------------------------------------------------------------
TOTALS                                  15806.47            16272.49
```

CASE STUDY #2

The second case study also deals with storage tanks, but here a distinction is made between 'liquid' and 'gas' tanks. Specifically, we wish to enforce that only liquid tanks replace liquid tanks and that only gas tanks replace gas tanks. The data is in table 2.

tank	ddt&e	unitcost	firstprd	quantity	size	liquid	gas
1	148.65	167.52	29.73	2	2	1	0
2	549.65	1302.46	109.93	2	5	1	0
3	549.65	1302.46	109.93	4	5	1	0
4	88.64	74.43	17.73	3	0	1	0
5	588.91	1451.32	117.78	2	6	1	0
6	102.20	93.08	20.44	4	1	1	0
7	178.57	223.29	35.71	1	3	1	0
8	178.57	223.29	35.71	1	3	1	0
9	663.29	1750.64	132.66	1	7	1	0
10	549.65	1302.46	109.93	2	5	1	0
11	549.65	1302.46	109.93	1	5	1	0
12	579.24	3787.29	115.85	4	8	1	0
13	192.50	671.48	38.50	4	1	0	1
14	286.79	1257.69	57.36	12	4	1	0
15	95.29	223.83	19.06	12	0	0	1
16	1318.61	13767.10	263.72	5	9	1	0
17	438.09	2451.43	87.62	5	2	0	1

Table 2. Liquid and gas storage tanks input file.

```
>>load tanks2
```

```
3 fields, 17 items.
Load successfully completed.
>>for liquid and size=x allow liquid and size>=x
98 new substitutions allowed.
>>for gas and size=x allow gas and size>=x
3 new substitutions allowed.
>>cost
Time required for an exhaustive analysis: approximately
605404800 seconds.
CAPS 1.0 will find the optimum solution in approximately 12
seconds.
Proceed? y
Working.................................................
.....................
```

Item	Replaces	Replacement Cost	Unique Cost
7	{1,7,8}	1191.19	1413.83
2	{2,3,10,11}	12989.99	14805.17
5	{5}	3709.44	3709.44
9	{9}	2546.59	2546.59
6	{4,6}	862.54	901.33
12	{12}	16115.97	16115.97
13	{13}	3007.22	3007.22
14	{14}	15851.97	15851.97
15	{15}	2938.39	2938.39
16	{16}	71217.19	71217.19
17	{17}	13048.44	13048.44
TOTALS		143478.93	145555.55

CASE STUDY #3

The final example relates to interface plates for the modular racks used to organize work, storage, and living space in space station modules. Each plate can accommodate zero or more utility interfaces, chosen from the following set:

avionics
nominal_power
high_power

```
fire_detection
data_management
thermal_control
hygiene_water
nitrogen
potable_water
hygiene_waste
```

Each of the above is represented in the data set by a boolean attribute of the same name. For a given interface plate, the utility interface is present on that plate if its corresponding attribute has the value 1, and not present if that attribute has the value 0. The major constraint in this example is that no interface plate be allowed to substitute for another if the latter contains an interface which the former does not have. The following "take file" was prepared to define these constraints for CAPS:

```
allow all
for avionics disallow not avionics
for nominal_power disallow not nominal_power
for high_power disallow not high_power
for fire_detection disallow not fire_detection
for data_management disallow not data_management
for thermal_control disallow not thermal_control
for hygiene_water disallow not hygiene_water
for nitrogen disallow not nitrogen
for potable_water disallow not potable_water
for hygiene_waste disallow not hygiene_waste
```

A relatively simple approach to cost is to count one monetary unit for each interface included on a given plate. With this approach we can define the 'unitcost' field by adding the following lines to the 'take file':

```
define unitcost as avionics+nominal_power+high_power+
  fire_detection+ data_management+thermal_control+
  hygiene_water+nitrogen+ potable_water+hygiene_waste
add unitcost
```

The data used for the example is not reproduced here. It was simply an arbitrary matrix of 1's and 0's with the 'ddt&e' field set to 11 monetary units for each item. The following analysis results:

```
>>status
Allowable substitutions:
   1 -> {1,6,7,8}
   2 -> {2,3,8}
   3 -> {3}
   4 -> {4}
   5 -> {4,5,9,10,11}
   6 -> {6}
   7 -> {7,8}
   8 -> {8}
   9 -> {9}
  10 -> {9,10}
  11 -> {11}
>>cost
Time required for an exhaustive analysis:  approximately 17
seconds.
CAPS 1.0 will find the optimum solution in approximately 17
seconds.
Proceed? y
Working..................................................
.................................
Item      Replaces          Replacement Cost         Unique Cost
  1       {1,6,7}
                                       38.00               54.00
  2       {2,3,8}
                                       26.00               43.00
  5       {4,5,9,10,11}
                                       51.00               80.00
-----------------------------------------------------------------
TOTALS                                115.00              177.00
```

The first two examples are nearly identical to the two case studies used in Yeager, 1987a and 1988. The third is a deliberately scaled down version of a study in Thomas, 1988. The scaling down was necessary because of CAPS' exhaustive solution strategy. This strategy will be compromised in future versions of CAPS, so that CAPS will be able to produce a suboptimal solution to any size problem and optimal solutions to those which have good properties, such as those given here.

CONCLUSIONS AND RECOMMENDATIONS

The CAPS 1.0 prototype was successfully designed, implemented, and tested, and conclusively demonstrates the feasibility of an intelligent software tool which can not only provide a more sophisticated approach to the solution of commonality analysis problems, but can also aid in structuring the analysis procedure itself. Future development of CAPS will increase its flexibility and functionality dramatically.

REFERENCES

MSFC. 1987. Commonality Analysis Study, User Manual for the System Commonality Analysis Tool (SCAT), D483-10064, March 1987. Contract NAS8-36413, NASA George C. Marshall Space Flight Center, Alabama.

THOMAS, L. D. 1988. Commonality Analysis Using Clustering Methods, submitted to _IEEE Transactions on Systems, Man, and Cybernetics_.

YEAGER, D. P. 1987a. Expert System Development for Commonality Analysis in Space Programs, Final Report, NASA/ASEE Summer Faculty Fellowship Program, pp. XXXV-i ff. Contract NGT-01-008-021, NASA George C. Marshall Space Flight Center, Alabama.

YEAGER, D. P. 1987b. Commonality Analysis as a Knowledge Acquisition Problem, Proceedings of the Third Annual Conference on Artificial Intelligence for Space Applications, November 3, 1987.

YEAGER, D. P. 1988. A Formalization of the Commonality Analysis Problem and Some Partial Solutions, submitted to _Mathematics of Operations Research_.