PharmaSUG 2023 – Paper SA-92

Picking Scabs and Digging Scarabs: Refactoring User-Defined Decision Table Interpretation Using the SAS® Hash Object To Maximize Efficiency and Minimize Metaprogramming

Troy Martin Hughes
Louise S. Hadden, Abt Associates Inc.

ABSTRACT

Decision tables allow users to express business rules and other decision rules within tables rather than coded statically as conditional logic statements. In the first author's 2019 book, *SAS® Data-Driven Development*, he describes how decision tables embody the data independence that data-driven programming requires, and demonstrates a reusable solution that enables decision tables to be interpreted and operationalized through the SAS macro language. In their 2019 white paper *Should I wear pants?*, the authors demonstrate the configurability and reusability of this solution by utilizing the same data structure and underlying code to interpret unrelated business rules that describe unrelated domain data—pants wearing and vacationing in the Portuguese expanse. Finally, in the current paper, the authors refactor this code by replacing metaprogramming techniques and macro statements with a user-defined function (that leverages a dynamic hash object) that performs the decision table lookup. The new hash-based interpreter is unencumbered by inherent macro metaprogramming limitations. This anecdotal "scab picking"—the subtle refactoring of software to expand functionality or improve performance—yields a more flexible interpreter that is more robust to diverse or difficult data, including special-character-laden data sets. In recognition of the authors' combined love for all-things-archaeological, the decision rules in this text separately model Mayan ceramics excavation and Egyptian scarab analysis.

INTRODUCTION

The International Organization for Standardization (ISO) defines a *decision table* as a "table of all contingencies that are to be considered in the description of a problem together with the action to be taken." (ISO 5806, 1984) Decision tables represent one method to effect dynamic outcomes or output from dynamic input, in which *decision rules* prescribe how data shall be interpreted or transformed. Decision tables are fully explored in the author's book, *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality, Second Edition*. (Hughes, 2022)

One hallmark of decision tables is the data-driven design they espouse, in which *software modularity* and *control data independence* dictate that business rules and the code interpreting them are maintained separately. That is, rather than effecting dynamic outcomes through conditional logic (like IF-THEN-ELSE statements) or other hardcoded methods, decision tables are maintained as external control tables that must be interpreted by underlying software. This software design paradigm shift—from outmoded hardcoding to superior data-driven design—provides several advantages, including software configurability, reusability, scalability, modularity, maintainability, and extensibility, and can promote both higher-quality and higher-performing software.

Software *configurability* is promoted because users can rely on a decision table data structure (and its underlying code) to interpret various decision rules that describe diverse domain data. Thus, despite the archaeological-focused examples in this text, the solution presented is flexible enough to meet the needs of other users and use cases. This flexibility also promotes *reusability*, as users are able to reuse decision table data structures for different analyses or transformations. Finally, this flexibility facilitates *scalability* because the solution can accommodate a limitless number of decision rules and decision outcomes.

Software *maintainability* is delivered primarily through the software *modularity* that decision tables embody—because the separation of decision rules from the code interpreting them allows each to be altered without modification of the other. This allows developers to modify the underlying code that interprets decision tables to improve its performance, or in the case of software *extensibility*, developers can modify software to expand its functionality. Extensibility is extensively modeled in this text, as a previous decision table solution is refactored to operationalize decision tables using the SAS hash object rather than

metaprogramming achieved via the SAS macro language. Thus, this text refactors a functionally equivalent decision table solution demonstrated by the authors in *Should I Wear Pants in the Portuguese Expanse? Automating Business Rules and Decision Rules Through Reusable Decision Table Data Structures that Leverage SAS Arrays*. (Troy Martin Hughes, 2019)

METAPROGRAMMING—THE GREAT MACRO CATASTROPHE

Metaprogramming describes "code that writes code," and the SAS macro language is the chippie who concocts this canoodling of code and metacode, the delicate dance of dynamically and statically coded functionality. Metaprogramming can facilitate more dynamic software, but can be grossly overused, with SAS practitioners reaching for the SAS macro language while overlooking more appropriate methods. This section summarizes the *Should I Wear Pants?* decision table metaprogramming method, only so it can be supplanted by the function- and hash-based solution demonstrated in subsequent sections.

Consider the single decision point of archaeological intoxication as it relates to the selection of an archaeological tool for excavation of a Mayan burial site: The lack of rum might lead you to select a delicate toothbrush; a little rum might lead you to select a handheld pick-axe; and a lot of rum might lead you to select a shovel—because at this point, you're stumbling around the dig site, and are less focused on preserving artifacts.

Far from the realm of science fiction, Figure 1 demonstrates One Barrel rum-induced excavation by the first author at the Mayan site of Minanhá, Belize, an outcrop of the more widely recognized Xunantunich civilization. The wizened, withered palm frond provided scant shade while we excavated and sifted dirt in search of potsherds, stone, bone, and other relics.

Figure 1. Troy Martin Hughes Digging in Mayan Archaeological Site, Minanhá, Belize

Figure 2 demonstrates a monochromatic ceramic, photographed in situ by the first author, discovered at Minanhá, Belize.

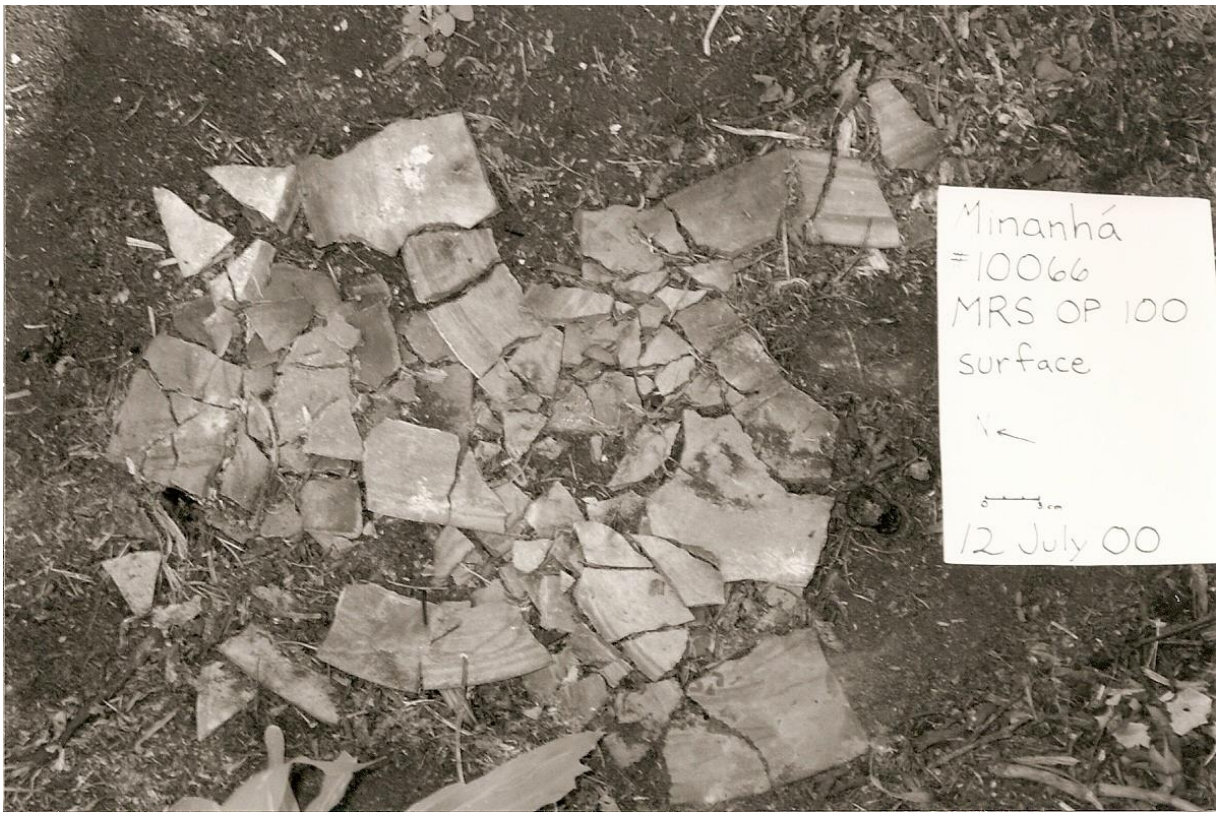

Figure 2. Monochromatic Mayan Ceramic Lying in Situ at Minanhá, Belize

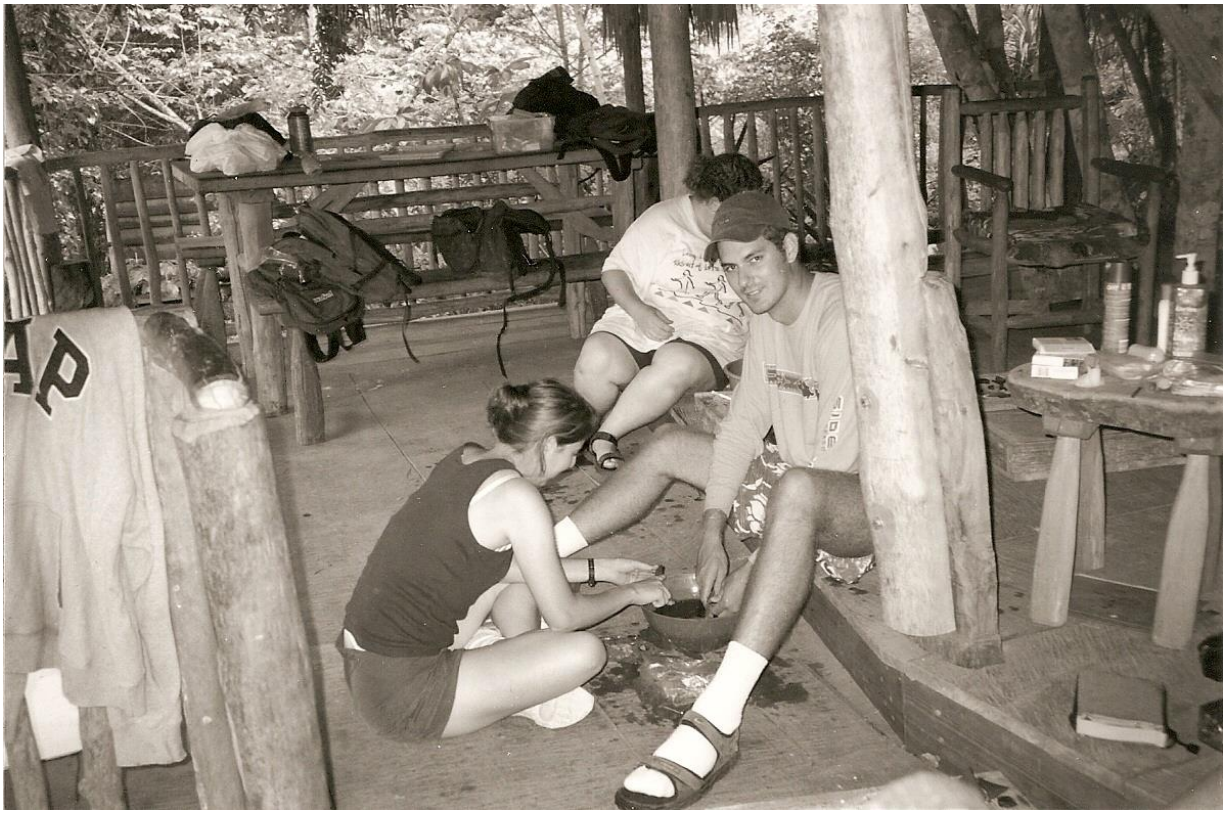

Figure 3. Troy Martin Hughes and Erica Fabo Polishing Mayan Potsherds at Minanhá, Belize

More commonly, however, only ceramic fragments were discovered, and these potsherds were carted back to camp for cleaning and analysis. Figure 3 shows the first author cleaning and cataloging potsherds. The open palapa, constructed of rough-hewn timbers and capped by a vaulted palm-frond roof, stands on the Martz Farm outside Benque Viejo del Carmen in the Cayo District of Belize.

With this scant introduction to archaeology, the preceding business rules can be modeled through static code within a DATA step:

```
data pick_a_tool;
   set arch_data;
   if rum_level='none' then tool='toothbrush';
   else if rum_level='some' then tool='handheld pickaxe';
   else if rum_level='lots' then tool='shovel';
run;
```

A downfall of hardcoded business rules is the necessity to modify code whenever the rules must be changed. Thus, speaking as a field-tested archaeologist, I can assure you that if no One Barrel rum is provided at the Mayan site being excavated, no tool will be selected because no work will be done! Thus, a more accurate representation of the decision rules could be stated in the following DATA step:

```
data pick_a_tool;
   set arch_data;
   if rum_level='none' then tool='N/A';
   else if rum_level='some' then tool='handheld pickaxe';
   else if rum_level='lots' then tool='shovel';
run;
```

Tired of continuously modifying rum-to-tool relationships within the code, a savvy developer might instead invoke data-driven design to model tool selection. This pursuit is accomplished by creating a decision table—a control table that includes discrete conditions mapped to their respective outcomes. Rum_tool_table.csv contains a header row, a single decision point column (rum_level), and a single outcome column (tool):

```
rum_level,tool
none,N/A
some,handheld pickaxe
lots,shovel
```

Multiple methods could operationalize this decision table into functional business logic, and one method transforms these raw values into the conditional IF-THEN-ELSE statements from the previous DATA step:

```
%let loc=C:\sas\;
data rum_to_tools;
   infile "&loc.rum_tool_table.csv" truncover firstobs=1 end=eof;
   length line $10000 decision_point_var outcome_var inp_val out_val $32
      macro_code $10000;
   format line $10000.;
   retain decision_point_var outcome_var macro_code '';
   input line & $;
   if _n_=1 then do;
      decision_point_var=strip(scan(line,1,','));
      outcome_var=strip(scan(line,2,','));
      end;
   else do;
      inp_val=strip(scan(line,1,','));
      out_val=strip(scan(line,2,','));
      if _n_>2 then macro_code=catx('',macro_code,'else');
      macro_code=strip(macro_code) || ' if ' || strip(decision_point_var)
         || '="' || strip(inp_val) || '" then '
         || strip(outcome_var) || '="' || strip(out_val) || '";';
      end;
   if eof then call symputx('macro_code',macro_code,'g');
run;
```

When the DATA step executes, it ingests the Rum_tool_table decision table, and the Macro_code variable is incrementally filled with conditional logic statements. After ingesting the final row of data, the END option (i.e., IF EOF...) initializes the &MACRO_CODE global macro variable to the value of the Macro_code variable.

The following %PUT statement prints the (unformatted) &MACRO_CODE variable to the log:

```
%put _global_;
GLOBAL MACRO_CODE if rum_level="none" then tool="N/A"; else if rum_level="some" then
tool="handheld pickaxe"; else if rum_level="lots" then tool="shovel";
```

This dynamically generated conditional logic can be subsequently called from within a DATA step to operationalize the decision rules contained within the decision table. First, a data set containing rum levels (of archaeological compadres) can be created:

```
data arch_data;
   length archaeologist $32 rum_level $10;
   archaeologist='Matt'; rum_level='lots'; output;
   archaeologist='Troy'; rum_level='some'; output;
   archaeologist='Erica'; rum_level='some'; output;
   archaeologist='Catherine'; rum_level='none'; output;
run;
```

Next, the &MACRO_CODE variable can be placed inside a DATA step to conditionally execute this dynamic code:

```
data pick_a_tool;
   set arch_data;
   length tool $32;
   &macro_code
run;
```

This method is functional and produces correct results, successfully transforming level of rum consumption into archaeological tool selection. However, the solution could fail were special characters such as single quotes, double quotes, ampersands, or percentage signs encountered in the data. Most egregiously, this metaprogramming method unnecessarily transforms a built-in data structure—the Rum_to_tools SAS data set—into rambling text, crammed into a single macro variable. In general, whenever control data are already maintained within a built-in data structure, it is far better to forego unnecessary transformation and to access them in situ. Thus, the refactoring of this initial metaprogramming approach overcomes these limitations.

ARCHAEOLOGICAL TOOL SELECTION DATA MODEL

Despite the facile demonstration of tool selection presented in the previous section, tool selection in fact encompasses far more finesse—you must account for soil substrate in addition to the intoxication level of the archaeologist performing the work! Thus, any software interpreter of decision tables must scale to accommodate multiple decision points—that is, a diversity of input parameters that collectively determine the decision outcome.

Table 1 demonstrates a decision table that captures business rules that identify tool selection.

	A	B	C
1	soil_content	rum_level	tool
2	dirt	none	N/A
3	dirt	some	shovel
4	dirt	lots	shovel
5	potsherds	none	pickaxe
6	potsherds	some	trowel
7	potsherds	lots	shovel
8	pottery	none	toothbrush
9	pottery	some	pickaxe
10	pottery	lots	trowel

Table 1. Decision Table for Tool Selection by Soil Content and Archaeologist Intoxication

The decision table represented in Table 1 can be saved as Rum_soil_tool_table.csv:

```
soil_content,rum_level,tool
dirt,none,N/A
dirt,some,shovel
dirt,lots,shovel
potsherds,none,pickaxe
potsherds,some,trowel
potsherds,lots,shovel
pottery,none,toothbrush
pottery,some,pickaxe
pottery,lots,trowel
```

This CSV file can be imported into the Rum_soil_tool_table SAS data set using the following code:

```
proc import datafile="&loc.rum_soil_tool_table.csv"
   out=rum_soil_tool_table
   dbms=csv
   replace;
run;
```

This decision table represents the data model that will be used to operationalize input variables (i.e., archaeologist intoxication and soil content) to prescribe selection of the tool that will be used. Note the scalability of the decision table data structure, with multiple decision points now incorporated. In this user-defined data structure, the final column (i.e., the Tool variable, in this instance) will always represent the decision outcome—the variable that the model produces.

Sample data have also been updated to reflect the soil in which digging is occurring, and saved to Arch_data_v2:

```
data arch_data_v2;
   length archaeologist $32 rum_level soil_content $32;
   archaeologist='Matt'; rum_level='lots'; soil_content='dirt'; output;
   archaeologist='Matt'; rum_level='lots'; soil_content='pottery'; output;
   archaeologist='Troy'; rum_level='some'; soil_content='potsherds'; output;
   archaeologist='Erica'; rum_level='some'; soil_content='pottery'; output;
   archaeologist='Catherine'; rum_level='none'; soil_content='potsherds'; output;
   archaeologist='Catherine'; rum_level='none'; soil_content='pottery'; output;
run;
```

Table 2 demonstrates the Arch_dat_v2 data set, which denotes the level of rum consumption and soil content for various archaeologists on various days. And by applying the decision table depicted in Table 1, Table 2 can be evaluated programmatically to determine which tool should be selected for each observation.

	archaeologist	rum_level	soil
1	Matt	lots	dirt
2	Matt	lots	pottery
3	Troy	some	potsherds
4	Erica	some	pottery
5	Catherine	none	potsherds
6	Catherine	none	pottery

Table 2. Dig Scenarios To Be Evaluated by Decision Table

In the next section, the decision table data model is applied to the archaeological data set to interpret its data.

OPERATIONALIZING A DECISION TABLE THROUGH A STATIC FUNCTION

The previous Metaprogramming section interpreted a decision table, and dynamically generated SAS code using the SAS macro language. A more robust (and concise) solution instead realizes that the decision table can be operationalized as a lookup table maintained within a hash object. In this paradigm, the one or more decision points represent the single- or multi-key index, and the outcome/output variable represents the value in the hash key-value lookup.

The following FCMP procedure declares the SELECT_TOOL user-defined function, which ingests the decision table as a hash object:

```
proc fcmp outlib=work.myfuncs.decision;
   function select_tool(soil_content $, rum_level $) $;
      length tool $100;
      declare hash h (dataset: 'rum_soil_tool_table');
      rc=h.defineKey('soil_content','rum_level');
      rc=h.defineData('tool');
      rc=h.defineDone();
      rc=h.find();
      return(tool);
      endfunc;
quit;
```

Once SELECT_TOOL has been defined, it can be called to transform the two decision point variables (Soil_content and Rum_level) into the outcome variable; note that the CMPLIB SAS system option must first instruct SAS in which data set (WORK.Myfuncs) to search for user-defined functions and subroutines:

```
options cmplib=work.myfuncs;
data pick_a_tool;
   set arch_data_v2;
   length tool $32;
   tool = select_tool(soil_content, rum_level);
run;
```

This hardcoded solution generates the Pick_a_tool data set, shown in Table 3.

	archaeologist	rum_level	soil	tool
1	Matt	lots	dirt	shovel
2	Matt	lots	pottery	trowel
3	Troy	some	potsherds	trowel
4	Erica	some	pottery	pickaxe
5	Catherine	none	potsherds	pickaxe
6	Catherine	none	pottery	toothbrush

Table 3. Pick_a_tool SAS Data Set Showing Decision Table Outcomes

This solution is functional and does demonstrate data-driven design, as the business rules for tool selection are entirely contained within the external decision table; however, the solution nevertheless lacks configurability because it can be used to operationalize only this single use case—tool selection. Thus, despite demonstrating data-driven design, it can be made more flexible to increase its configurability, reusability, and scalability, as demonstrated in the next section.

OPERATIONALIZING A DECISION TABLE THROUGH A DYNAMIC FUNCTION

To imbue the SELECT_TOOL function with configurability, scalability, and reusability, several aspects of the function's declaration must be dynamically (rather than statically) created; these are highlighted in the following code, and include the function's name, decision point variables, outcome variable, and data set holding the decision table:

```
proc fcmp outlib=work.myfuncs.decision;
   function select_tool(soil_content $, rum_level $) $;
      length tool $100;
      declare hash h (dataset: 'rum_soil_tool_table');
      rc=h.defineKey('soil_content','rum_level');
      rc=h.defineData('tool');
      rc=h.defineDone();
      rc=h.find();
      return(tool);
      endfunc;
quit;
```

Some of these dynamic elements can be parameterized and passed to the function when it is called, including the function name and name of the underlying decision table. However, the list of decision point variables and the name of the outcome variable must be programmatically identified, and supplied dynamically using the SAS macro language.

This demonstrates a more appropriate use of metaprogramming and the SAS macro language—to generate SAS syntax dynamically, rather than encapsulating the business rules themselves. The following MAKE_DECISION_FUNCTION macro dynamically creates a user-defined FCMP function:

```
%macro make_decision_function(function /* function name */,
   dsn /* decision table in simplified format */);
* dynamically determine last variable, the outcome variable;
proc sql noprint;
   select strip(upcase(name)) into : outcome_var
      from dictionary.columns
         where libname='WORK' and memname="%upcase(&dsn)"
            order by varnum desc;
quit;
%let outcome_var=%trim(&outcome_var);
* dynamically create parameter list;
proc sql noprint;
   select strip(name) || ' $' into : param_list separated by ','
      from dictionary.columns
         where libname='WORK' and memname="%upcase(&dsn)" and
upcase(name)^="&outcome_var";
```

```
quit;
* dynamically create key list;
proc sql noprint;
   select quote(strip(name)) into : key_list separated by ' ,'
      from dictionary.columns
         where libname='WORK' and memname="%upcase(&dsn)" and
upcase(name)^="&outcome_var";
quit;
* dynamically build FCMP function;
proc fcmp outlib=work.myfuncs.decision;
   function &function(&param_list) $;
      length &outcome_var $100;
      declare hash h (dataset: "&dsn");
      rc=h.defineKey(&key_list);
      rc=h.defineData("&outcome_var");
      rc=h.defineDone();
      rc=h.find();
      return(&outcome_var);
      endfunc;
quit;
%mend;
```

The first SQL procedure dynamically evaluates the last variable in the decision table, which is prescribed to be the outcome variable that is created. This variable, initialized as &OUTCOME_VAR, must have trailing blanks truncated with the %LEFT macro function, after which it can be used in the FCMP procedure.

The second SQL procedure dynamically evaluates the input variable(s) that comprise the single- or multi-indexed key. This variable (&PARAM_LIST) represents a comma-delimited list of the input parameters that must be declared in the FCMP FUNCTION statement. Each parameter also includes a trailing $ to declare it as a character parameter within the function.

The third SQL procedure dynamically evaluates the input variable(s), and generates a second comma-delimited list of parameters (&KEY_LIST) that does not include any trailing $. These three SQL procedures collectively generate the required dynamic input that is used in the subsequent FCMP procedure.

The following code calls the MAKE_DECISION_FUNCTION macro to create the SELECT_TOOL FCMP function dynamically. Thereafter, the identical DATA step (demonstrated in the previous section) calls the dynamically created SELECT_TOOL function:

```
%make_decision_function(select_tool, rum_soil_tool_table);

options cmplib=work.myfuncs;
data pick_a_tool;
   set arch_data_v2;
   length tool $32;
   tool = select_tool(soil_content, rum_level);
run;
```

The flexibility of this solution, which combines the benefits of macro metaprogramming and reusable user-defined functions, is further demonstrated throughout this text, in which separate data structures are utilized to build decision tables dynamically that answer different archaeological questions.

OPENING THE DOOR TO USER-DEFINED DATA STRUCTURE REUSE

The second author shares a strong interest with the first author in both archeology and travel to archeological sites. Egyptian history is a particular favorite. Hence, we decided to test the extensibility of the dynamic function building routines described previously using a different test scenario involving archeological sites in Egypt. The beauty of crafting a dynamically created function is that the process can ingest innumerable types of data, as long as the rules of engagement are followed.

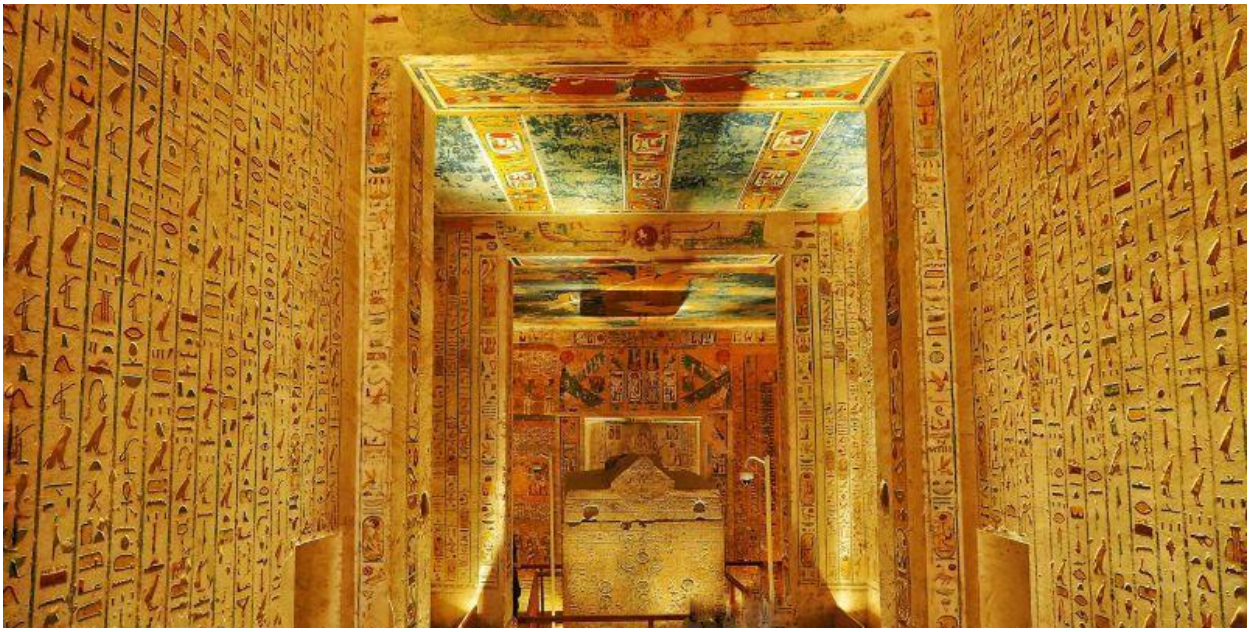

Figure 4. Hieroglyphics in Tomb of Ramses IV, Valley of the Kings, Egypt

In a prior paper, the authors applied similar decision table rules to planning travel in Portugal. We followed that example for this paper and presentation by applying the improved FCMP / hash solution to planning visits to archeological sites, with a focus on searching for Egyptian antiquities and other activities. Many years ago, pre-post college life, the second author took a short summer trip to Egypt, and came home with a small collection of scarabs, and has been fascinated ever since.

Scarabs have long played an integral role in Egyptology. Scarabs are a variety of dung beetle, and have rolled dung balls on the earth for the past 115 million years. The first documented references to scarabs in Egypt occur around 2000 BCE. The import of scarabs has evolved from rebirth, immortality, and metamorphosis, to carvings representing good fortune and protection, and then to uses in ornamentation, such as jewelry. Along with the use of scarabs in carvings and jewelry, scarab tattoos are common in Egypt. Scarabs have been found in archeological digs throughout Egypt, and relatively inexpensive reproductions can be found in marketplaces and souvenir stores in many of the destinations discussed in this paper. Last, but not least, scarabs embody some of the tenets of data-driven programming, namely reusability and transformation.

Figure 5. Scarabaeidae (Scarab Beetle)

Flexibility of software solutions is of paramount importance, and we wanted to explore how adaptive the FCMP / hash solution is. To demonstrate the reusability of the decision-making tool, a number of relevant areas in Egypt were selected. Each area has a tourist attraction associated with it, a none/some/lots variable that indicates whether scarabs can be obtained while engaging in a given activity, and a yes/no variable

that indicates whether hookah is available in proximity to the given activity. Cairo boasts the Giza Pyramids and the ability to procure lots of scarabs; Luxor boasts the Valley of the Kings Tombs and the ability to procure some scarabs; Sharm El-Sheikh provides stunning beaches and reefs; and so on.

Figure 6. From Left to Right: Karnak Temple, Valley of Kings Tombs, Philae Temple, Ras Muhammad Park, and Sharm El-Sheikh

A CSV file (ScarabsV1.csv) was created using the various decision rules – with columns or variables for the area, whether or not there are scarabs to be found at the attraction, whether hookah is available at the attraction, and the attraction name. The structure mimics the decision table inputs described previously, in which the left three columns comprise three decision points, and the rightmost column represents the decision outcome, as shown in Table 4.

	A	B	C	D
1	area	scarabs	Hookah	attraction
2	Cairo	None	No	Egyptian Museum
3	Cairo	Some	Yes	Sphinx
4	Cairo	Lots	Yes	Giza Pyramids
5	Luxor	Lots	No	Karnak Temple
6	Luxor	Some	No	Valley of Kings Tombs
7	Sharm El-Sheikh	None	Yes	Beaches and Reefs
8	Sharm El-Sheikh	Some	Yes	Ras Muhammad Park
9	Abu Simbel	Lots	Yes	Abu Simbel Temple
10	Abu Simbel	None	Yes	Sun Festival
11	Aswan	None	Yes	Agilika Island
12	Aswan	Some	No	Philae Temple

Table 4. Decision Table for Site Selection by Activity and Availability of Scarabs and Hookah

The IMPORT procedure imports the decision table:

```
%let loc=C:\sas\;

proc import datafile="&loc.scarabsV1.csv"
   out=area_scarabs_hookah_attr1
   dbms=csv
   replace;
run;
```

The following invocation of the MAKE_DECISION_FUNCTION macro creates the SELECT_ATTR1 user-defined function:

```
%make_decision_function(select_attr1,area_scarabs_hookah_attr1);
```

The decision table macro is invoked, and then the created function is applied to the data:

```
data pick_attr1;
   length area $ 50 scarabs $ 4 hookah $ 3 attr $ 50 composite_key $ 500;
```

```
   set area_scarabs_hookah_attr1;
   attr = select_attr1(area,scarabs,hookah);
   composite_key=catx('||',area,scarabs,hookah,attr);
run;
```

This step results in a decision table that neatly informs the viewer of locations where various tourist attractions can be enjoyed, and maybe find some number of scarabs and enjoy some hookah along the way.

	area	scarabs	hookah	attr
1	Cairo	None	No	Egyptian Museum
2	Cairo	Some	Yes	Sphinx
3	Cairo	Lots	Yes	Giza Pyramids
4	Luxor	Lots	No	Karnak Temple
5	Luxor	Some	No	Valley of Kings Tombs
6	Sharm El-Sheikh	None	Yes	Beaches and Reefs
7	Sharm El-Sheikh	Some	Yes	Ras Muhammad Park
8	Abu Simbel	Lots	Yes	Abu Simbel Temple
9	Abu Simbel	None	Yes	Sun Festival
10	Aswan	None	Yes	Agilika Island
11	Aswan	Some	No	Philae Temple

Table 5. Pick_Attr1 Data Set Showing Decision Table Outcomes

Our next step is to apply the hash table function to a file without tourist attractions to lay out the parameters of our decision-making process. We have created a function based on the decision table above, and all that needs to happen is to specify the input file and the key variable names. As seen in the code snippet below, there are now 3 activities associated with tourist activity. No changes need to be made to the function we have created, unless we add key variables or decision points. Tourist attractions are assigned to all rows, based on the function based on the decision table inputs shown previously. Since the input columns area, scarabs, and hookah are identical to what went into the function creation, the resulting output is identical (the composite key created is not shown in Table 6 below).

```
data pick_attr2;
   length area $ 50 scarabs $ 4 hookah $ 3 attr $ 50 composite_key $ 500;
   set area_scarabs_hookah2 ;
   attr = select_attr1(area,scarabs,hookah);
   composite_key=catx('||',area,scarabs,hookah,attr);
run;
```

	area	scarabs	hookah	attr
1	Cairo	None	No	Egyptian Museum
2	Cairo	Some	Yes	Sphinx
3	Cairo	Lots	Yes	Giza Pyramids
4	Luxor	Lots	No	Karnak Temple
5	Luxor	Some	No	Valley of Kings Tombs
6	Sharm El-Sheikh	None	Yes	Beaches and Reefs
7	Sharm El-Sheikh	Some	Yes	Ras Muhammad Park
8	Abu Simbel	Lots	Yes	Abu Simbel Temple
9	Abu Simbel	None	Yes	Sun Festival
10	Aswan	None	Yes	Agilika Island
11	Aswan	Some	No	Philae Temple

Table 6. Pick_Attr2 Data Set Showing Decision Table Outcomes

A third version of the "Pick Attr" decision table was created to explore the effect of adding some noise: duplicate and missing keys. The duplicate keys in the Hash solution have an unexpected result: the result belonging to the first instance of a given combination of key variables is propagated to all other records with the same combination of key variables, regardless of the decision table assignment. Hash takes the first result (in this case, tourist attraction) for a given key by default, and returns that for all instances of a given key, regardless of what is assigned to the duplicate keys in the decision table. You can modify this behavior, but a demonstration of complex hash objects is out of scope for this paper. Additionally, since the combination of area=Cairo, scarabs=Some, and hookah=No does not exist in the decision table, an attraction is not returned, and the cell (i.e., outcome variable) is left blank.

	A	B	C
1	area	scarabs	Hookah
2	Cairo	Some	No
3	Cairo	Some	Yes
4	Cairo	Some	Yes
5	Luxor	Lots	No
6	Luxor	Some	No
7	Sharm El-Sheikh	None	Yes
8	Sharm El-Sheikh	Some	Yes
9	Abu Simbel	Lots	Yes
10	Abu Simbel	None	Yes
11	Aswan	None	Yes
12	Aswan	Some	No

Table 7. Input Table for Area, Availability of Scarabs, and Availability of Hookah – Duplicate Keys

	area	scarabs	Hookah	attr	attraction (from decision table)
1	Cairo	Some (was None)	No		Egyptian Museum
2	Cairo	Some	Yes	Sphinx	Sphinx
3	Cairo	Some (was Lots)	Yes	Sphinx	Giza Pyramids
4	Luxor	Lots	No	Karnak Temple	Karnak Temple
5	Luxor	Some	No	Valley of Kings Tombs	Valley of Kings Tombs
6	Sharm El-Sheikh	None	Yes	Beaches and Reefs	Beaches and Reefs
7	Sharm El-Sheikh	Some	Yes	Ras Muhammad Park	Ras Muhammad Park
8	Abu Simbel	Lots	Yes	Abu Simbel Temple	Abu Simbel Temple
9	Abu Simbel	None	Yes	Sun Festival	Sun Festival
10	Aswan	None	Yes	Agilika Island	Agilika Island
11	Aswan	Some	No	Philae Temple	Philae Temple

Table 8. Pick_Attr3 Data Set Showing Decision Table Outcomes – Duplicate Keys

Despite the resultant issues observed in Table 8 due to the faulty decision table (shown in Table 7), the underlying data-driven design facilitates rapid remediation, as corrective maintenance can be performed through *configuration*—modification of the control table—rather than *customization*—modification of the underlying code. That is, although not demonstrated, a business analyst or other SME having access to only the decision table could alter (and fix) software functionality without altering either the MAKE_DECISION_FUNCTION macro or the user-defined FCMP function it dynamically creates.

CONCLUSION

Decision tables represent a data-driven solution that evaluates dynamic input to generate an associated decision outcome. This text refactored a previous macro-based solution by the authors, which relied on metaprogramming to generate conditional logic statements dynamically. The current solution instead leverages a hash object instantiated inside a user-defined function, which enables the underlying decision table to be maintained exclusively within built-in data structures—the SAS data set and the hash object. This approach improves software quality through the flexibility and robustness of the interpreter that operationalizes the decision table. Finally, the importance and benefit of defining and standardizing user-defined data structures (like the decision table data structure used separately by both authors) is illustrated, in that the same data structure (and code that interprets it) can be reused to evaluate vastly different archaeological data and business rules.

REFERENCES

Hughes, T. M. (2014). From a One-Horse to a One-Stoplight Town: A Base SAS Solution to Preventing Data Access Collisions through the Detection and Deployment of Shared and Exclusive File Locks. *Western Users of SAS Software (WUSS).* San Jose, California.

Hughes, T. M. (2022). *SAS® Data-Driven Development: From Abstract Design to Dynamic Functionality, Second Edition.* Filthy Data, Inc.

ISO 5806. (1984). *Information processing - Specification of single-hit decision tables.* Geneva, Switzerland: International Organization for Standarization.

Troy Martin Hughes, L. H. (2019). Should I Wear Pants in the Portuguese Expanse? Automating Business Rules and Decision Rules Through Reusable Decision Table Data Structures that Leverage SAS Arrays. *SESUG.* Williamsburg, VA.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

Name: Louise Hadden
E-mail: louise_hadden@abtassoc.com

Figure 7. Temple of Halthor, Detail on Ceiling, Dendera, Egypt