

Programming By Rules

Magnus Mengelbier, Limelogic AB, Sweden

ABSTRACT

Rules are one of the cornerstones for analysing clinical trials, either defined through regulatory guidance, analysis plans or specifications. As standards for both the capture, analysis and reporting of data prevail in a business process; the flexibility is further constrained or more precisely defined. We can use this to our advantage by creating an evolving book of rules that can be implemented to verify that the capture data follows standards and study expectations as well as deriving the more standard variables for analysis and reports. We consider how a rule book can influence standard programming and the two SAS macros that are the cornerstone for rule based programming.

INTRODUCTION

Two popular myths are that all analysis is unique or all analysis is standard analysis and consistent. Both are valid but we are fast approaching consistency as CDISC and other industry standards have become mainstream and a de-facto standard.

Most organisations have a set of standards of varying detail, official and enforced through templates or unofficial propagated through copying the last or a similar analysis. As data standards mature, evolve and are refined, additional elements are incorporated into the standards as part of the natural process. More organisations are progressing to develop a book of standard derivations that drive consistency across all analysis.

It is not uncommon for data set standards, and associated standard derivations, to be implemented through large all-inclusive and ever complex standard macros. Another approach is to create many smaller functional and less flexible macros to execute a single defined derivation efficiently. Or simply eliminate macros entirely and focus on template code that efficiently derives one or a set of related rules.

Although each approach has its own merits and drawbacks, the functional approach, many small macros, is more efficient in meta-data driven environments as one macro would implement one derivation rule. Because the simplicity of the functional rule-based macros, creating test cases for validation is a straightforward exercise.

PROGRAMMING BY RULES

Programming by rules implements a modular approach and set pattern that combines the derivation of related variables, much like the structure of specifications.

A good example is Age versus Body Mass Index (BMI), where the variables AGE and BMI are derived in two separate rules, simply because they are not related.

```
%sdm_age_1_0( data = work.source_dm, out = work.age );
%sdm_bmi_1_1( dm = work.source_dm, vitals = work.source_vs, out = work.bmi );
```

Regardless of our coding approach, we need to consider that a derivation can be a simple affair, consisting of one line, or span multiple DATA steps and SAS procedures.

RULE MACRO APPROACH

The rule macro approach will result in many small data sets, conceivably one for each variable in extreme cases, that will need to be combined into a final data set. After each rule, we register the intermediate data set, the derived variables, and any necessary merge keys (`%rules_register()`) and at the end use that information to create the final output data set (`%rules_make()`).

The rule macro follows a set naming convention and structure with three reserved macro parameters *data*, *out* and *instep*. Additional parameters are assumed to be rule specific.

```
%my_rule_macro( data = , param1 = , param2 = , ... , out = , instep = );
```

The *data* and *out* parameters define the input and output data sets, respectively. If multiple input data sets are required, then each parameter corresponding to the required input data sets are considered rule specific parameters.

The *instep* parameter is reserved for future use to allow a rule macro to be combined with other rule macros within the same DATA step and procedure to improve execution efficiency.

The remaining macro parameters, e.g. *param1*, *param2*, etc., are used to define variable references and other rule options that are specific to each rule macro.

META-DATA DRIVEN

Changes to the specification (Figure 1) are subtle if we want to control derivation from meta-data.

- ✓ A rule reference
- ✓ Rule options
- ✓ Inclusion flag

The inclusion flag uses

- ☐ Required
- ☐ Permissible
- ☐ Not Applicable

Rule	Rule Options	Include	Deriv
dm_vs_1_2		Required	
dm_vs_1_2		Required	
dm_vs_1_2		Required	
dm_bsa_3_1		Required	

Figure 1. Rule reference in a data set specification

In this case, Permissible means use it if it is derived. We could also simply rely on Yes and No.

The concept of having meta data generate programs (`%rules_make_program()`) is not far off given that we have a pattern for rule programming, convention for standard rule macros and the documentation to drive the process.

A natural extension would be to wrap this entire process, from a book of rules to meta-data creating the programs, with a suitable interface and the promise of point-and-click reporting is a reality. The added benefit is that the approach retains a high degree of flexibility with regards to both analysis and the associated programming

THE MACROS

%RULES_REGISTER()

The macro is a utility that manages a list of intermediate data sets and associated variables so that the final output data set can easily be created.

```
%rules_register( registry = work._rules_registry_,
                rule = ,
                data = ,
                vars = ,
                keys = ,
                dupkey = E );
```

Parameter	Description
registry	Registry data set that is later used to build the final output data set. If the registry data set does not exist, it will be created the first time the macro is called.
rule	Reference to the derivation rule as defined in a book of rules or the specification that relates to the intermediate data set and variables (optional)
data	Intermediate data set that contains the merge keys and derived variables specified in the <i>keys</i> and <i>vars</i> parameters
vars	List of derived variables
keys	Defines the merge keys to be used when adding the variables to the final output data set
dupkey	Check if the keys identify a single record and will output a SAS error (E), warning (W), note (N) to the log

%RULES_MAKE()

The macro creates an output data set based on the intermediate data sets and variables as defined in the registry, starting with the core variables.

```
%rules_make( registry = work._rules_registry_,
            out = ,
            meta = ,
            strict = Yes );
```

Parameter	Description
registry	Registry data set with the list of intermediate data sets and associated merge key and derived variables
out	Output data set
meta	Meta data set (optional)
strict	Verify that all variables defined in meta data set exists. Output a SAS error (E) or warning (W) in the log, if a variable defined in meta is missing.

%RULES_MAKE_PROGRAM()

Macro that creates one or more standard programs using the meta data in the specification.

```
%rules_make_program( meta = , output = );
```

Parameter	Description
meta	Meta data set that defines all derivations to be completed
output	Output location for generated programs

There will be situations where the set of rule macros will not include all the rules required for an analysis. It may be a decision that only 80% of the derivations should be covered by rule macros or simply a rule macro is not available at the time of analysis. One approach is for the `%rules_make_program()` to automatically include a "custom" program just prior to creating the output data set.

```
%rules_custom();
```

The `%rules_custom()` macro is a simple solution that will look for the "custom" SAS program file in the current folder and include its content, if it exists. This will allow a system to auto-generate and maintain a SAS program based on meta-data alone, while at the same time allow for manually maintained code.

CONCLUSION

We all advocate, practice and adhere to programming by the rules set out in study protocols, Statistical Analysis Plans and associated specifications. In practice, we are practicing meta-data driven programming, or programming by rules, as we progress from source data sets to final report deliverables, but the path from one-off programs to SAS programs generated entirely from meta-data will require shifts in programming conventions, small new utilities and the predictable updates to input specifications.

The programming approach discussed highlights the results of taking a programming process from one-off programs to entirely meta-data driven with the flexibility you would expect. Add a simple meta-data interface and you have the ability to create study report deliverables with the click of a button while retaining a flexible analysis and programming process.