UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE

CONFERENCE OF EUROPEAN STATISTICIANS

Expert meeting on Statistical Data Editing

3-7 October 2022, (virtual)

Rule Management

Mark van der Loo, Edwin de Jonge, Olav ten Bosch (Statistics Netherlands, The Netherlands)

mpj.vanderloo@cbs.nl

I. INTRODUCTION

1. An important aspect in the modernisation of statistical production systems is the idea that subject matter knowledge should be separated as much as possible from technical (IT) knowledge. This is apparent in the Common Statistical Production Architecture [ModernStats, 2015] which promotes a separation between the description of business functions and implementation detail. It also promotes a modular architecure where building a production system in principle amounts to assembling a set of prebuilt modules for different business functions and parameterizing them with business logic for the system at hand.

2. In practice these considerations lead to rule-driven data processing systems. In the area of data editing this is nothing new. At least since the seminal paper of Fellegi and Holt [1976] on error localisation, statistical offices have worked with rule sets to express subject matter knowledge. The idea of rule-based processing is also applied in the area to conditional, domain-specific correction rules Pannekoek et al. [2013], to make domain knowledge explicit and to make production processes reproducible and transparent. Other recent developments include the development of the Validation and Transformation Language [SDMX, 2015] and the development of data validation and rule-based data cleaning engines based in R and Python [van der Loo and de Jonge, 2021b,a, Bantilan, 2020].

3. The advent of rule-based production systems, as well as the ongoing integration of production systems accross institutes naturally leads to the issue of rule management. A need arises to manage and maintain rule sets that drive production processes. If done properly, rule management systems offer great advantages to statistical organisations. They facilitate reproducibility of production, exchange and reuse of formalized domain knowledge, and offer the possibility of comparing rule sets accross production systems.

4. The idea of rule management systems is currently developed in several contexts. For example, Eurostat is building a repository for the Exchange of data validation rules within the European Statistical System. Within Statistics Netherlands, two programmes for renewing social and economic statistics aim to create modular and rule-based production systems, which calls for rule management systems.

5. This paper contributes to the current discussion by starting with the definition of a few user stories, that express what a user might expect from a repository. Next, we try to formalize the discussion by formalizing the concept of a rules and rule sets. We find that the central object of rule management is the concept of a rule sequence: and ordered list of rules. Next, we define a basic set

of operations on rule sequences that are sufficient to support the user stories. We demonstrate an experimental software that implements these ideas in and R-based API that interfaces with a database [R Core Team, 2022]. Finally, we summarize our work and discuss a avenues for further research.

II. **User stories**

1. User stories are short statements that express tasks that software users may want to perform. They are always of the form 'as an X I want to have Y, so I can do Z', where X, Y, and Z are the type of user, the activity to be supported and the business goal to be achieved. User stories are common in agile development cycles, and are aimed to give a simple and understandable overview of what a system is supposed to be able to do, as well as a way to prioritize development.

2. For this work, we assume the following user stories for a rule management system: as a statistician producing official statistics, I want to

S1 Create, read, update, and delete rules so I can fix my current understanding of a statistical domain in the form of a formal ruleset.
S2 Select a sequence of rules so I can apply them to my data.
S3 Determine the order of rule execution so I have full control over data processing and validation.
S4 Be able to trace the evolution of my rule sets so I can (a) give full account of my production runs, and (b) reproduce production runs.
S5 Temporarily remove a rule from one or more rule sets so I can handle exceptional and transient data circumstances. This temporary removal should be documented.
S6 Temporarily update a rule from one or more rulesets so I can handle exceptional and transient data circumstances. This temoporary update should be documented.

3. There are also user stories from the perspectif of a statistical organisation. Although these will not be very influential to our design, it is still illuminating to mention them here. As a statistical organization, I want to

O1 Promote reuse of rules.
O2 Promote transparency and learning accross production systems, by comparing and benchmarking rules and rule sets.

III. **Formal considerations**

1. The most important thing to do in developing any software system is to design the formal concepts and relations that are necessary to support its function. Only then can one guarantee that the resulting system can be fully understood in the sense that all its capabilities and limitations are known; that one can reason about its behaviour because it derives from a compact set of formal concepts and relations; that it is extensible because one can build complex functionality based on the simple underlying concepts and relations.

2. In the current work, we informally work with data tranformation rules of various types, and data validation rules and certain tables to store and manage them. In the following subsections we provide formal definitions for generic data transformation rules, for the data structures to hold them, and for the basic operations to manipulate them.

A. What are rules?

1. The premise of many modern official statistical production systems is that they should be rule-driven. Yet, a formal definitioin of such data processing rules seems to be lacking. Here, we generalize the work on data validation rules first discussed in van der Loo [2015] but see van der Loo and de Jonge [2018, 2020] for extensive discussions. In this work, a data set is a finite collection of data points where a *data point* is defined as a key-value pair (k, x), where x is the value from a specified domain D and k is a key that fully identifies the value. In other words, the key represents necessary and sufficient metadata that identifies the value. To fully identify a value, it was demonstrated that each key should at least represent the population, the measurement or observation used to obtain the value, the unit of the population that was measured, and the variable that was measured.

2. To understand the ful generality of this definition of a data point, it helps to see an example. Suppose we ask two questions to two employees, Alice and Bob, of Statistics Netherlands (SN), namely their Age and in what Division they work: Economic Statistics (ES) or Social Statistics (SS). In this case there are four data points, identified by the following keys:

Population	measurement	unit	variable
Employees of SN	Our questionnaire	Alice	Age
Employees of SN	Our questionnaire	Alice	Division
Employees of SN	Our questionnaire	Bob	Age
Employees of SN	Our questionnaire	Bob	Division

We are quite permissive in our questionnaire, and it is possible that for example Bob mixes up the answers to the questions by filling in the Age in the 'Division' field, and the Division in the 'Age' field. Therefore, the domain D is the union of all possible answers to all possible questions.

$$D = \{\text{SS}, \text{ES}\} \cup \mathbb{N} \cup \{\text{NA}\},$$

where NA stands for Not Available, to allow for missing values. A combination of a key from the above table and a value from D completely identifies the value.

3. A *data set* is a collection of data points where each key occurs exactly once. This gives a data set the structure of a function (a fact that was also noted by Gelsema [2012]) $K \to D$, where K is a finite set of keys. Observe that we do not force any structure on the data set. It may consist of a single data point, a record, a few records, or a collection of random variables from random units from random populations. A data set is therefore not necessarily relational (as in relational databases) although in practice they often are. Given a finite set of keys K, and a domain D, the set of all possible datasets is the space of functions from K to D, denoted D^K. We are now ready to define the concept of a rule.

4. A *data transformation rule* r is a function that transforms one data set into another data set.

$$r : D^K \to D'^{K'}. \tag{1}$$

Here, D and D' may or may not be the same, and similarly K and K' may or may not be the same. This framework is general enough to cover a wide range of commonly used types of rules and data processing operations, as will be demonstrated by the next examples.

5. **Data validation rules.** The above general definition reduces to the definition of a data validation rule in van der Loo and de Jonge [2020] by choosing $D = \{0, 1\}$, setting K' equal to the singleton set $\{1\}$, and demanding that the function is surjective. Data validation rules are expressions that evaluate to a boolean that is interpreted as to whether a dataset satisfies the demand expressed in the rule or not.

6. **Conditional data cleaning rules.** These are conditional expressions of the form `IF` a certain data condition is met `THEN` change the value of a certain variable. In these transformations the key set does not change, but the Domain D may shrink from a wide range of possible values to a smaller (allowed) range of values. So we set $D' \subseteq D$ and $K' = K$ in Equation (1).

7. **Deriving new variables.** This includes creating new variables from one or more existing variables. Typical applications are change of units, classification, or deriving data quality indicators. In this case metadata changes as a new Also, the domain might change since new variables may have new value domains. So deriving variables falls in the most general category defined by Equation 1.

8. **Combining sources.** Merging two data sets, based on common properties is as common as it is important in statistical data processing. Before defining this in the context of the current framework, let us look at an example. Suppose we have a two measurements (for example questionnaires in a panel) and for a certain variable X we want to compare the value of the previous measurement τ' with the current measurement τ. To combine the sources, we need to check for each data point with key (U, τ', u, X) (u is the population unit) whether a data point with key (U, τ, u, X) exists. If so, we create a new data point where we copy the value from τ' to τ into a new variable:

$$((U, \tau', u, X), x') \mapsto ((U, \tau, u, Y), x').$$

This amounts to a left join where we can now compare $((U, \tau, u, Y), x')$ with $((U, \tau, u, X), x)$. Similar operations can be defined where we combine data from different populations U and U' for example. In this case, the metadata changes because we define a new, merged variable Y, but the domains stay the same as Y only contains copies of an existing variable X. Hence, an operation that combines sources is a function $D^K \to D^{K'}$ with $K \subseteq K'$.

9. **Aggregation.** This amounts to counting of otherwise combining multiple data points in to a single data point using arithmetical, statistical, or classifying operations. A fundamental aggregation is a function $D^K \to D'^{\{1\}}$.

10. Modeling data as a function from a set of content metadata elements to a domain of allowed values has the advantage that it allows for elegant and formal modeling. This is the case for formal database theories Codd [1970], as well as more current work based on category theory such as in the work of Gelsema [2012], and of Fong and Spivak [2019, Chapter 3]. The main disadvantage is that it is difficult to model certain technical issues. In particular the existence of multiple values for the same data (duplicate records) are not expressible as functions from content metadata to a value domain. For example, it is technically perfectly possible to create a dataset where the same person occurs twice, once with birth year 1978 and once with birth year 1987. The formal solution is to extend the key set as defined above and by adding technical metadata, such as the location in the file. For our current discussion on rule management, these limitations have no consequences.

B. Rule sequences

1. In general, it matters in which order data transformations are executed. Therefore, the main data structe that users encounter are not rule sets (as they are usually called) but rather *rule sequences*. That is, a list of rules $L = (r_1, r_2, \ldots, r_n)$ with a predefined order. When applying a rule sequence to data, we may think of it as a composed function $L = r_n \circ r_{n-1} \circ \cdots \circ r_1 : D^K \to D'^{K'}$, turing one data set into another dataset.

2. The above consideration suggests that a rule management system consists principally of a set of labeled rule sequences $L_1, L_2, \ldots, L_k$, where each L_i is a version of a rule sequence, index i labeling

the version. To keep track of changes, updates are only done on the rule set with the highest index, and any update results in a new rule sequence that is added to the set. Principal updates include the following basic operations:

(1) Inserting a rule
(2) Removing a rule
(3) Replacing (updating) a rule
(4) Changing rule order (permutation).

Formally, we only need two operations to express any of these operations. Concatenating a single rule at the end of a rule sequence and removing a single rule at the end of a rule sequence are the most basic operations necessary to express all the above operations. However, from a user perspective the above operations are the natural activities. User stories S1 and S5 demand that versions can be annotated. This means that the basic object is a list of objects $\langle L_i, M_i\rangle$ where L_i is a rule sequence and M_i holds metadata about the current version. Given a set of versioned and annotated rule sequences with k the latest version, any allowed operation

$$\phi_{k+1} : \{\langle L_1, M_1\rangle, \langle L_2, M_2\rangle, \ldots, \langle L_k, M_k\rangle\} \mapsto \{\langle L_1, M_1\rangle, \langle L_2, M_2\rangle, \ldots, \langle L_k, M_k\rangle, \langle L_{k+1}, M_{k+1}\rangle\}$$

where the operation ϕ_{k+1} is a combination of insertions, removals, replacements and permutations. It is not hard to convince oneself that these operations support user stories S1 (CRUD), S2 (select a sequence), S3 (determine execution order), S5 (temporarily remove rules), S6 (temoporarily update rules). For support of S4, and in particular the tracability of the evolution of rule sequences, we need to look into more detail what the contents of the metadata element M_{k+1} should hold.

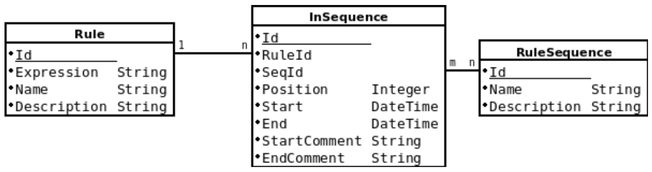
3. The problem is that the expression ‘trace the evolution of rules and rule sets’ is not very precise. If we interpret it as the ability to precicely reconstruct all changes to a rule sequence, then we need to set $M_i = \phi_i$ (plus some user-defined annotations), and in fact we can suffice by working only with the sequence of transformations $\phi_1, \phi_2 \ldots \phi_k$, where it is assumed that $L_1 = \phi_1(\ )$ (ϕ_1 applied to the empty list). If we interpret S4 as the ability to qualitatively reconstruct what changes were applied and why, we can suffice with M_i representing user-defined descriptions. The actual changes to rule sequences can be approximated, for example by computing the shortest sequence of insertions, removals, substitutions and permutations that change L_k into L_{k+1} (Damerau-Levenshtein distance, see Wagner and Lowrance [1975]). The only extra demand is that we are able to compare two rules and test if they are equal or not. In this work we shall assume that we only need the latter, less precise version of tracability of the evolution of rule sequences. We expect that users are less interested in the precise technical steps taken to alter a rule sequence, and more in the qualitative description of it.

IV. The rulemanager package

1. The `rulemanager` package [van der Loo, 2022] is an experimental implementation of the ideas presented above. It offers an API to set up and manage rule repositories based on standard relational database systems. The aim is to offer a low-level interface (set of functions) that can be used to build user-friendly (graphical) interfaces for rule management.

2. Relational data bases have set-based logic: users cannot (by default) assume any ordering of records. This means that the order of a rule sequence has to be represented explicitly. The simplest implementation would be to create a single table with attributes `Sequence`, `SequenceVersion` and `Rule` and `Position`, representing the rule sequence and version, and the rules with their position in the sequence. Any update to a rule sequence would require adding a complete new version of a rule to the table, but at least the whole history will be retrievable.

3. To save space, we choose an appraoch that is more natural to databases and implement the following database diagram.



We distinguish three entity types: `Rule`, `InSequence` and `RuleSequence`. The entity type `RuleSequence` only serves to add descriptions and other metadata to a sequence. For example, a reference to the production system for which they are aimed. The `Rule` serves to avoid duplication of rules. The central entity type is `InSequence` which defines which rules are element of which rule sequence and their position in the rule sequence. It is understood that both sequence membership and position are valid only in the range [`Start`, `End`). The `StartComment` and `EndComment` attributes represent the metadata M_i discussed above: it is used to store user comments on the reason for updates to the list. In this representation, each record in `InSequence` represents two events: the creation and deletion of a rule. Hence it contains two comment fields.

4. To find a particular version of a rule sequence, one filters the elements of `InSequence` for a chosen time `T` such that `Start <= T < End`, and then collects the actual expressions from `Rule`. This implements user stories S2, and S4. Based on this schema we can implement the basic operations of Inserting, Removing, Replacing or Permuting rules, which covers all other user stories that relate to manipulating the repository.

5. In the above schema rules and rule sequences have to be created before assigning rules to a position in a particular rule sequence. Therefore we define the following functions (in pseudocode).

```
new_rule(string: Expression, string: Name, string: Description)
new_seqn(string: Name, string: Expression)
```

that add a record to `Rule` or `RuleSequence`, respectively. In the above interfaces, as well as in all pseudocode below, we shall not explicitly include the primary keys unless it is strictly needed.

6. **Inserting a rule.** In pseudocode, inserting a rule looks as follows.

```
insert(Int: RuleId, Int SeqId, Int: Position, String: Comment):
   // Shift position of subsequent rules to make place for the new rule
   n := maximum Position in rule sequence SeqId
   FOR i = Position TO n:
     WHERE InSequence.SeqId == SeqId DO
       InSequence.Position = InSequence.Position + 1
   // Insert the new rule into the right position
   add <RuleId, SeqId, Position, CurrentTime(), NULL, Comment, NULL> to InSequence
```

Where `CurrentTime()` returns a time stamp. By setting this automatically, we prevent the user from 'changing history', which would violate user story S4. We initialise the `End` time with `NULL`, which is interpreted as ∞. We also initialise the `EndComment` with `NULL`.

7. **Removing a rule.** Since we wish to retain a historical record, rules are never really removed. In stead, we set their `End` time in the rule sequence.

```
remove(Int: InSeqId, DateTime: When, String: Comment):
  IF When == NULL THEN When := CurrentTime()
  WHERE InSequence.Id == InSeqId DO:
    IF InSequence.End <= When : Return()
```

```
      ELSE  InSequence.End := When
            InSequence.EndComment := Comment
```

Where `Return()` terminates function execution and returns normally. We avoid removing a rule twice by checking whether it was removed earlier. It is important to refer to the primary key of the `InSequence` record, because we allow for rules to occur multiple times in a single sequence. Given a rule sequence (`SeqId`), an element is uniquely defined by `RuleId`, `Position`, `Start` and `End`, so it is possible to base the API on these attributes as well. In a visual interface, users normally do not have to bother with these details as they can just select the rules graphically.

8. **Replacing a rule.** This amounts to removing one rule in a certain position and adding another, using the two functions above. If this is a single action from a user point of view, the `EndComment` of the removed rule also serves as the `StartComment` of the new rule since both actions were performed for a single reason.

9. **Permuting rules.** All permutations can be build from simple swaps of two positions. The following code does permutes the position of two rules.

```
swap(Int: SeqId, Int: Position1, Int: Position2):
  WHERE InSequence.SeqId == SeqId DO
    IF Position1 or Position2 does not exist: Return()
  WHERE InSeqence.SeqId == SeqId AND InSequence.Position == Position1 DO
    InSequence.Position := NULL
  WHERE InSeqence.SeqId == SeqId AND InSequence.Position == Position2 DO
    InSequence.Position = Position1
  WHERE InSeqence.SeqId == SeqId AND InSequence.Position == NULL DO
    InSequence.Position = Position2
```

The above function first checks whether the two positions exist before performing the actual swap.

V. Summary and outlook

Starting from the point of view of statistical users, we defined a set of wishes that a rule repository should, and should not perform. Next, we formalized the concept of rules, rule sequences and rule repositories, where we also identified the basic operations that are necessary to support the user stories. Finally, we define a database schema and API that implements these operations on a basic level.

1. In the current work, the rule repository is completely agnostic about the type of rules that are stored. From the point of view of maintainability and reusability this also highly desirable. From the point of view of the user one can think of extensions built upon rule repositories that are very interesting. In the final Section below we point out a few interesting avenues for further research.

A. Some Open Questions

1. From the examples in Section A it is clear that rules can in general not be placed in any order. Simple example: it is only possible to aggregate a derived variable, after the derivation has taken place. It is an interesting question to what extent a rule management system, or tools built upon a rule management system are capable of detecting such issues. We are not aware of much research in this area. Scholtus [2016] has done work in the context of a generalized Fellegi-Holt paradigm, based on affine transformations of numerical data. van der Loo and de Jonge [2018, Chapter 9] derive

some general conditions under which conditional data cleaning rules are idempotent or commutative. However, as rule-based production systems are slowly becoming the norm and considering the trend of internationalisation of (parts of) statistical production, amongst others through the exchange of rules ten Bosch and van der Loo [2022] investigating the quality of rule sets becomes a pressing issue. We principally distinguish the following research questions.

2. **Redundancy and consistency** Rule sets tend to grow organically in organisations. This means that it is possible that one part of a rule sequence defines data transformations that are undone further down the sequence. For reasons of understandability and maintainability it is preferable to avoid such inconsistencies, or at least to be able to detect them. Similarly, it is possible for rules to occur twice or have two rules with a different explicit definition perform the same activity. Again, it is preferable to avoid or detect such situations.

3. **Rule minimization and rule counting** It is hard to count rules. For example the rule $x \mapsto x+1$ applied twice can be expressed as $x \mapsto 2$. So it depends on the way changes to a data set are implemented how these changes are counted. The fact that it is hard to count rules makes it difficult to objectively compare rule sequences. The logical solution that comes to mind is to find a procedure that reduces any rule set to a minimal set of canonical instructions. Indeed this reminds of compiling code and it possible that, by specializing rule sets to activities typical for statistical data processing, we can learn something from that area of computer science.

References

Niels Bantilan. pandera: Statistical data validation of pandas dataframes. In *Proceedings of the Python in Science Conference (SciPy)*, pages 116–124, 2020.

E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13 (6):377–387, 1970.

Ivan P Fellegi and David Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical association*, 71(353):17–35, 1976.

B Fong and D.I. Spivak. *An invitation to applied category theory: seven sketches in compositionality.* Cambridge University Press, 2019.

T. Gelsema. The organization of information in the statistical office. *Journal of Official Statistics*, 28: 413–440, 2012.

ModernStats. Common statistica production architecture. Technical report, 2015.

Jeroen Pannekoek, Sander Scholtus, and Mark Van der Loo. Automated and manual data editing: a view on process design and methodology. *Journal of Official Statistics*, 29(4):511, 2013.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2022. URL https://www.R-project.org/.

Sander Scholtus. A generalized fellegi-holt paradigm for automatic error localization. *Survey Methodology*, 42(1):1–19, 2016.

SDMX. Validation and transformation language 2.0. Technical report, 2015.

O. ten Bosch and M.P.J. van der Loo. Quality assurance from an internationally standardized and generic data validation ecosystem. In *European conference on quality in official statistics*, Vilnius, 2022.

Mark van der Loo. *rulemanager: an API for versioned rule management*, 2022. URL https://github.com/SNStatComp/rulemanager. R package version 0.0.1.

Mark van der Loo and Edwin de Jonge. *dcmodify: Modify Data Using Externally Defined Modification Rules*, 2021a. URL https://CRAN.R-project.org/package=dcmodify. R package version 0.1.9.

Mark P. J. van der Loo and Edwin de Jonge. Data validation infrastructure for R. *Journal of Statistical Software*, 97(10):1–31, 2021b. doi: 10.18637/jss.v097.i10.

M.P.J. van der Loo. A formal typology of data validation functions. In *UNECE Work Session on Statistical Data Editing*, Budapest, 2015. URL http://www.unece.org/stats/documents/2015.09.sde.html.

M.P.J. van der Loo and E. de Jonge. *Statistical data cleaning with applications in R*. John Wiley and Sons, Inc, New York, 2018. ISBN 1118897153.

M.P.J. van der Loo and E de Jonge. *Data Validation*, pages 1–7. American Cancer Society, 2020. ISBN 9781118445112. doi: 10.1002/9781118445112.stat08255. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat08255.

Robert A Wagner and Roy Lowrance. An extension of the string-to-string correction problem. *Journal of the ACM (JACM)*, 22(2):177–183, 1975.