

Automatic Module Detection in Data Cleaning Workflows: Enabling Transparency and Recipe Reuse

Lan Li
School of Information Sciences
University of Illinois, Urbana-Champaign

Nikolaus Parulian
School of Information Sciences
University of Illinois, Urbana-Champaign

Bertram Ludäscher
School of Information Sciences
National Center for Supercomputing
Applications (NCSA)
University of Illinois, Urbana-Champaign

Abstract

Before data from multiple sources can be analyzed, data cleaning workflows (“recipes”) usually need to be employed to improve data quality. We identify a number of technical problems that make application of FAIR principles to data cleaning recipes challenging. We then demonstrate how *transparency* and *reusability* of recipes can be improved by analyzing dataflow dependencies within recipes. In particular column-level dependencies can be used to automatically detect independent subworkflows, which then can be reused individually as data cleaning modules. We have prototypically implemented this approach as part of an ongoing project to develop open-source companion tools for OpenRefine.

Keywords: Data Cleaning · Provenance · Transparency · Workflow Analysis

Submitted 27 January 2021 ~ Accepted 12 March 2021

Correspondence should be addressed to Lan Li and Bertram Ludäscher. Email: [lanl2,ludaesch}@illinois.edu](mailto:{lanl2,ludaesch}@illinois.edu)

This paper was presented at International Journal of Digital Curation IDCC21, Edinburgh, 19 April 2021

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: <http://www.ijdc.net/>

Copyright rests with the authors. This work is released under a Creative Commons Attribution 4.0 International Licence. For details please see <http://creativecommons.org/licenses/by/4.0/>



Introduction and Overview

The FAIR guiding principles for scientific data aim to ensure that data is *findable*, *accessible*, *interoperable*, and *reusable* (Wilkinson et al., 2016). Metadata in general and *provenance* information in particular can be used to improve reusability of data and the reproducibility of results by increasing *transparency* (Nosek et al., 2015; McPhillips et al., 2019). A major reason to provide provenance information is to increase the trustworthiness of data by laying open its *lineage*, i.e., its origins and processing history. It is also often pointed out that data analysis consists of at least 80% “wrangling and cleaning” of data, leaving only 20% for the data analysis proper. Not surprisingly, data cleaning “recipes” (or *workflows*) are of central importance in data analysis and should thus follow principles similar to FAIR data; in particular they should be *transparent* and *reusable*.

Consider a user U (a researcher or data curator) who wants to prepare a dataset D_0 for analysis as part of a scientific study. Upon inspection she quickly realizes that D_0 is not “fit for purpose” (Chapman et al., 2020), i.e., its organization and data quality needs to be improved for the intended use cases. She then employs a data cleaning tool such as OpenRefine (OR, 2021) to obtain a “cleaner” version that can be used for the subsequent analyses. An important byproduct of U ’s interactive data cleaning process is an *operation history* H that can be used to describe her data cleaning workflow as a sequence of *steps* S_1, \dots, S_n that transform the given D_0 via intermediate snapshots into a final version D_n :

$$D_0 \xrightarrow{S_1} D_1 \xrightarrow{S_2} D_2 \xrightarrow{S_3} \dots \xrightarrow{S_n} D_n . \quad (H)$$

This operation history H can be understood as a form of *provenance* information that increases the transparency of the data cleaning process. It contains *prospective* elements at the *workflow level*, e.g., the names of operations used, and *retrospective* provenance at the *trace level*, e.g., the concrete value changes and parameter settings used in each step.

Technical Challenges and Approach

The first practical challenge is how to present the history H in a form that helps users (the original workflow creator U , possible auditors, and other potential users) *understand* the data cleaning workflow, both at a higher, conceptual level, and at a more detailed level. We call this the **transparency problem** for H .

A second challenge for U is how to identify and extract from H one or more *reusable recipes* $R \subseteq H$, i.e., those elements and subworkflows of the data cleaning history that can be reused for other datasets. We call this the **reusability problem**.

There are several other technical challenges worth mentioning: While U is exploring and cleaning a dataset, she occasionally has to backtrack and undo earlier steps to modify her workflow. To this end, OpenRefine provides an *undo/redo* operation stack to support the back-and-forth of this interactive process. She can backtrack from the current step S_k to an earlier step S_i , inspect the dataset at that point, and then redo the steps S_{i+1}, \dots, S_k . Once she decides to modify S_i , however, the subsequent steps S_{i+1}, \dots, S_k are typically discarded.¹ If a user wants to retain some or all of these steps, they can try to work around the limitations of the tool, e.g., by saving the steps to be re-executed and then trying to

¹ The rationale is that a change in S_i may invalidate the subsequent steps S_{i+1}, \dots, S_k .

reapply them after the modified step S_i . The **history update problem** is to determine which of the steps S_{i+1}, \dots, S_k need to be discarded because they no longer apply, or how to modify them so they remain applicable.

Recipe migration problem. When presented with a new but similar dataset D'_0 , a curator U may want to reuse the parts of the history H that seem applicable not just to D_0 , but more generally to other datasets similar to it (such as D'_0). Thus U wants to select a recipe $R \subseteq H$ for reuse, possibly revising it as needed. Indeed R is often not applicable to D'_0 directly, but may have to be modified further to account for the *schema differences* between D_0 and D'_0 .

Recipe evolution problem. Finally, a user may want to share a recipe with a wider audience, in the spirit of the FAIR principles. To this end, she plans to subject each candidate recipe R_0 to a number of tests before publishing it. Lessons learned from the tests result in a sequence $R_0 \rightsquigarrow R_1 \rightsquigarrow \dots \rightsquigarrow R_\ell$ of modifications to and variants of the original recipe. The problem is how to support this evolution from R_0 to a publishable recipe R_ℓ , i.e., how to effectively modify, organize, and test recipes.

Contributions. In this paper, we focus on solving the *transparency problem* and one aspect of the *reusability problem*. We employ an underlying provenance model for data cleaning that can be used to describe data dependencies at different levels of granularity: e.g., in a **table view**, we view a history H as a sequence of snapshots D_0, D_1, \dots, D_n in which each new snapshot D_{i+1} is derived from its predecessor D_i via a transformation step S_{i+1} . In this way, using a coarse-grained table view, a user can get a first, high-level overview about a data cleaning workflow. On the other hand, a finer-grained dependency analysis will take into account the *columns* that an operation reads and writes, using the input-output *signatures* of the data transformation functions that implement a step S_i in the workflow. The resulting **schema view** paints a more detailed and transparent picture of how the workflow operations act on the schema, as the dataset is being transformed, step by step. Finally, the **modular view** provides another column-oriented view on the workflow: Similar to the schema view, column-level dependencies are obtained from the recipe's dataflow graph, but now *independent subworkflows are automatically identified*, which can then form the basic building blocks (or *modules*) of reusable recipes.

Both the schema view and the modular view rely on a *classification of operations* in order to highlight the different kinds of effects that workflow steps have on a dataset and its schema: many transformations operate on a single column, while others read or write multiple columns or rename columns; some operations are *generic* (i.e., can be applied, in principle, on any dataset column, e.g., `toLower` or `toUpper`), while others are *domain-specific* and *user-defined* (e.g., a custom mapping that fixes common misspellings over a controlled vocabulary of city names); yet other operations convert data types (e.g., from string to number), etc. We have prototypically implemented our approach as part of an ongoing project to develop open-source companion tools for OpenRefine (Li, 2021).

Data Cleaning Workflows and OpenRefine

OpenRefine is a popular open-source data cleaning tool that allows users to execute data transformations in a browser-based, spreadsheet-like user interface (OR, 2021).

As shown in Figure 1, the tool allows users to export an operation history H as a recipe R by selecting a set of operations $R \subseteq H$. The resulting recipe can be copied and reused

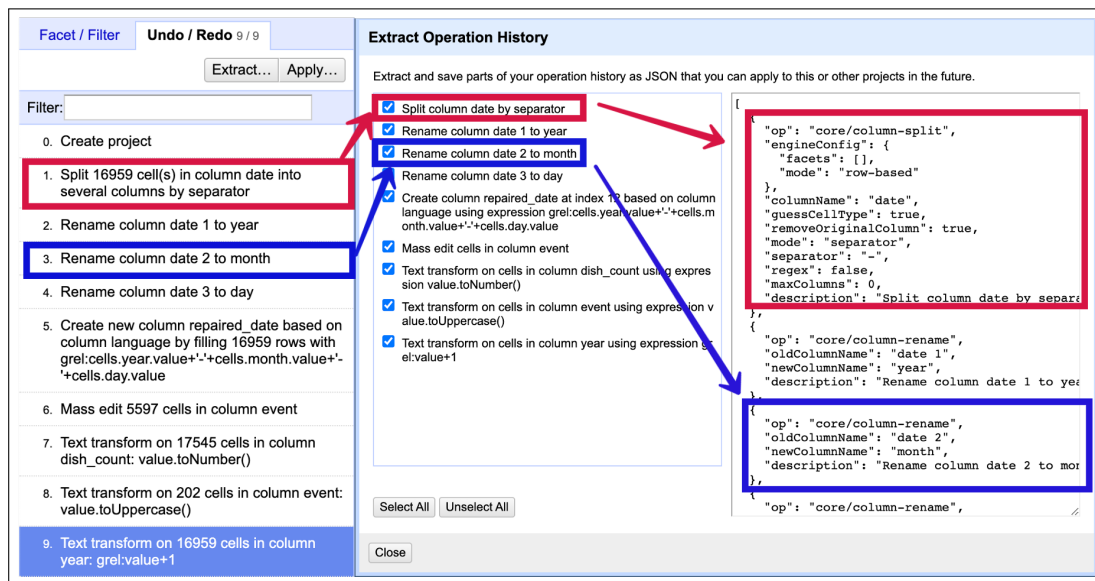


Figure 1. OpenRefine History and Recipe. The undo/redo history H (left) shows all steps previously executed. After clicking “Extract”, the user can select the operations to be exported via a checklist (middle). The selected recipe operations $R \subseteq H$ then appear in the JSON panel (right). Steps 1 (column-split) and 3 (column-rename) are highlighted in the history, the checklist, and the JSON recipe. Example taken from a recipe for the *menus* dataset, crowd-sourced by volunteers for the NYPL (New York Public Library, 2020).

later for other datasets.

On the left of Figure 2, a typical data cleaning use case is depicted: The user employs OpenRefine to clean a dataset D_0 . The process $D_0 \xrightarrow{H} D_n$ generates an operation history H with n steps S_1, \dots, S_n from which the user can select a recipe $R \subseteq H$ (see Fig. 1). This recipe can then be exported in an OpenRefine-specific JSON format for reuse on another dataset. Since R is not meant for “human consumption”, it is rather opaque and hard to digest for users.

A first step towards improving transparency is to create a *workflow model* from a recipe R . We previously developed *or2yw*², which automatically generates YW (YesWorkflow) annotations from recipes. The YW tool uses these to create a workflow model that can be queried and visualized (McPhillips et al., 2015). YW was originally developed to allow researchers a simple way to manually annotate their program scripts in order to reveal workflow steps and dataflow dependencies implicit in those scripts, thereby turning an opaque script into a transparent workflow. With the help of *or2yw* the user can *automatically* generate YW models from OpenRefine recipes (rather than manually creating them), thus providing a more transparent workflow view on a previously executed sequence of data cleaning operations. For a given workflow model, the YW tool can generate several different views, e.g., a *combined view* which depicts a workflow as a directed graph, linking steps via the data that flows between them; a *process view* that emphasizes steps; and a *data view* that highlights data derivations.

We have since implemented ORMA (OpenRefine Model Analysis), a new OpenRefine companion tool that uses additional details about H from *project files* created by OpenRefine

² *or2yw*: an OpenRefine-to-YesWorkflow mapping tool (Li et al., 2021).

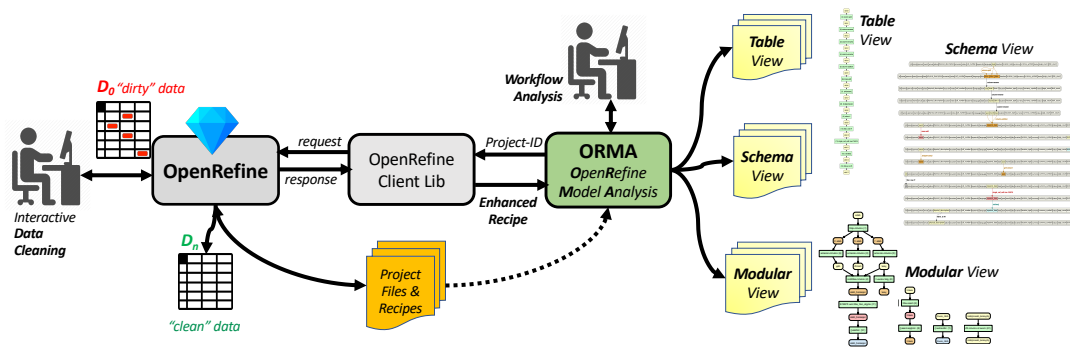


Figure 2. A user cleans dataset D_0 with OpenRefine, obtaining an improved version D_n , along with internal project files and recipes. During workflow analysis, the user provides ORMA with a project-ID, which is then used to generate an enriched recipe (i.e., a workflow model). The tool then analyzes data dependencies to generate multiple views: a high-level, linear *table view* closely corresponding to the sequential recipe, a *schema view* showing how columns are used and updated by each step; and a *modular view*, automatically restructuring the recipe into its independent subworkflows (*modules*).

when a data cleaning project is saved. The ORMA prototype can then create “enriched” recipes from this information which in turn are used to create new knowledge products (*table view*, *schema view*, and *modular view*) as illustrated in Figure 2.

Modeling Data Cleaning Workflows

Before taking a closer look at some of the new views and visualizations that ORMA supports, let us first consider the underlying workflow, provenance, and data models.

Modeling Workflows and Provenance

On the left of Figure 3, two modeling levels for data cleaning recipes are depicted:

- At the upper **workflow level** an operation such as “ $Y = F(X)$ ” stands for any data transformation that reads some data X (e.g., from a table, a column, or a cell) and that produces some output data Y .³
- At the lower **trace level**, a concrete value $x_i \in X$ is used by an invocation “ $y_i = F_i(x_i)$ ” to generate a result value $y_i \in Y$.

The data cleaning recipes of OpenRefine can thus be understood as workflow-level, *prospective* provenance, since they prescribe what operations should be executed (and in which order) when the recipe is to be reused. But since recipes R are obtained from operation histories H , they also constitute *retrospective* provenance information, since at least a partial record of the processing history is recorded by them.

In addition, OpenRefine captures more detailed retrospective provenance in its internal *project files* (cf. Figure 2), e.g., *parameter values* used at runtime by certain operations, and *deltas* that can be used to recreate any intermediate snapshot D_i when the user employs the undo/redo feature of OpenRefine.

³ Note that this includes the special case that Y represents the updated version of X .

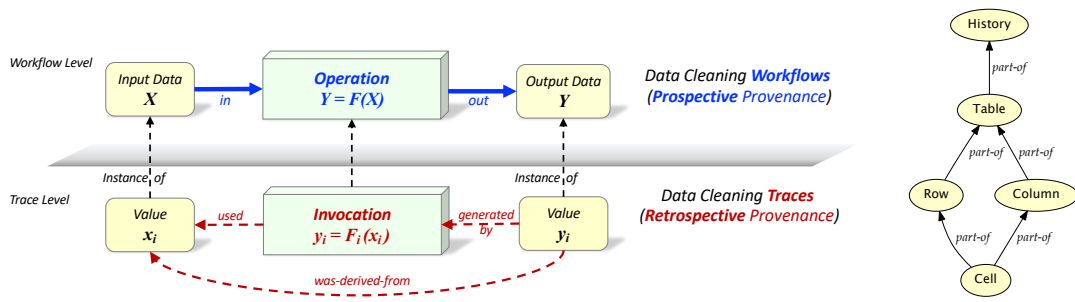


Figure 3. Workflows, Traces, and Data Granularity. A data cleaning recipe (*workflow*) consists of data transformations (*operations*) of the form “ $Y = F(X)$ ”, i.e., where a function is applied to an input X to obtain output Y (*upper left*). Workflows are a form of *prospective provenance*. At runtime, an *invocation* “ $y_i = F_i(x_i)$ ” yields *retrospective provenance*, e.g., describing that y_i *was generated by* F_i , which in turn *used* x_i (*lower left*). The data being processed can be modeled at different *levels of granularity* (*right*): a cell can be identified by the row and column it is part of; tables consist of rows and columns, and can have a history (implying that rows, columns, and cells also have a history).

On the right of Figure 3, a simplified version of the underlying **data model** is depicted. It shows that individual *cells* are part of *rows* and *columns*, which in turn are part of a data *table* D_i . Each table is part of a *table history*, i.e., a sequence of snapshots D_0, \dots, D_n . The power of ORMA and similar tools derives from the use of a flexible underlying model that combines prospective and retrospective information to describe data cleaning workflows, while simultaneously supporting different data granularities (e.g., *table*, *column*, *cell*) when describing relationships between elements. This combination allows one to create many types of relationships between the modeling elements, beyond the simple binary relationships (*in*, *out*, *used*, *generated-by*, and *was-derived-from*) depicted in Fig. 3.

Modeling Data Transformations

The final ingredient needed by ORMA to analyze data cleaning recipes as workflows is information about data transformations (operations): for each data cleaning operation or function F , we need to know at least its *signature* $F: \bar{X} \rightarrow \bar{Y}$, i.e., the column(s) $\bar{X} = X_1, \dots, X_k$ that F reads (or *uses*) and the column(s) $\bar{Y} = Y_1, \dots, Y_m$ that it writes (or *generates*). In this way, by analyzing direct and indirect dependencies between operations of a workflow, we can create different workflow views, including a modular view which automatically detects *independent* subworkflows.

Consider two consecutive steps S_i and S_{i+1} in a workflow. A fundamental question is to determine whether S_{i+1} depends on S_i (so execution must be serial) or whether S_{i+1} is in fact independent of S_i (which can be modeled using a parallel branch in the workflow).

If the two steps are implemented by operations $F_i: \bar{X}_i \rightarrow \bar{Y}_i$ and $F_{i+1}: \bar{X}_{i+1} \rightarrow \bar{Y}_{i+1}$, we can test whether $\bar{Y}_i \cap \bar{X}_{i+1} = \emptyset$. If this is the case, then the input \bar{X}_{i+1} of S_{i+1} does not make use of any output \bar{Y}_i of S_i , and we can say that S_{i+1} is independent of S_i . In this way, a data cleaning recipe can be modeled as a workflow graph consisting of *data nodes* that represent schemas or columns, and *transformation nodes* that represent the steps (data cleaning operations) of the workflow.

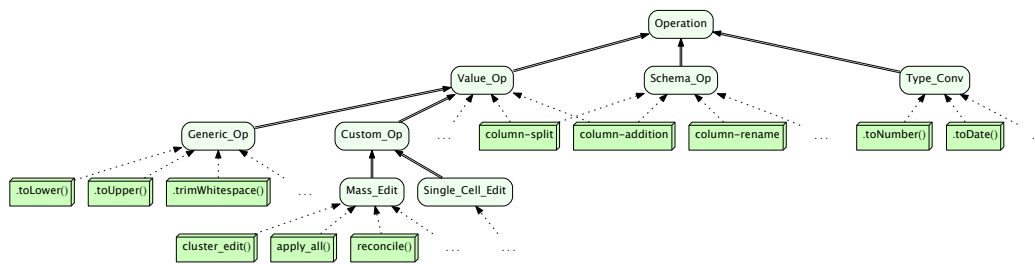


Figure 4. Classifying Operations: OpenRefine implements a range of data transformations, from simple, generic operations (`toLowerCase()`, `toUpperCase()`, ...) to powerful domain-specific operations requiring user interaction (e.g., for *clustering*, *normalizing*, and *reconciling* values), all the way to user-defined operations (e.g., using *regular expressions*).

Classifying Data Transformations

To make graphical workflow views more informative and intuitive for the user, different shapes and colors can be used in ORMA. For example a *table view* (as in Fig. 5 on the left) is depicted as a bipartite graph in which rounded yellow boxes represent table snapshots D_i , and green boxes represent workflow steps S_i that transform one snapshot to another. For more detailed views such as the *schema view* and the *modular view*, it is helpful to classify operations according to their shared properties, and then use colors (and/or shapes) to highlight the different effects that related operations have on the data tables.

Figure 4 depicts a (very incomplete) view of a possible classification of the built-in operations in OpenRefine: many operations change data values (`Value_Op`), some change the data schema (`Schema_Op`), and yet others the data type (`Type_Conv`). Often it is also useful to distinguish generic operations (`Generic_Op`) from domain-specific operations (`Custom_Op`). While the former can be applied, in principle, on any data column, the latter may be applicable only for certain domains of data (e.g., a mapping from common variants and misspellings of city names to their canonical representation). There are many other ways to organize and classify operations, depending on the desired perspective or use of the classification. In the current version of the ORMA tool, custom operations (such as single cell edits and certain mass edits) are prominently highlighted in red, while generic operations are highlighted in orange (cf. Figure 5 and Figure 6).

Transparency and Recipe Reuse through Workflow Views

We are now ready to describe how ORMA, our OpenRefine Model Analysis tool, can be used to increase the transparency of data cleaning recipes and facilitate their reuse.

ORMA Table View

When using OpenRefine to clean a dataset, an operation history H is created, allowing the user to extract some of the operations as a recipe $R \subseteq H$ (Figure 1). Additional provenance details can be obtained from internal project files (Figure 2) which ORMA can then use to create different workflow views. The *table view* on the far left in Figure 5 depicts a linear workflow that corresponds one-to-one to an operation history (similar to the one in

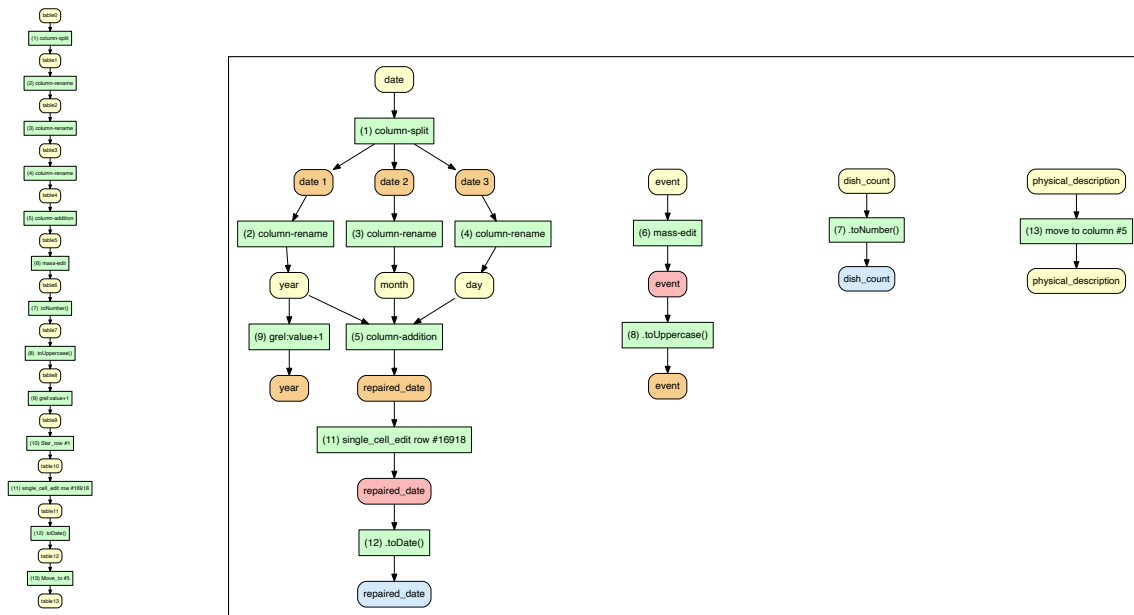


Figure 6. Table View vs Modular View. The table view on the left suggests (correctly) that each table snapshot depends on its predecessor via a workflow step. On the other hand, the **modular view** on the right highlights that the linear workflow can be decomposed into four *independent* subworkflows (*modules*), based on a finer-grained *column-level* dependency analysis: The largest module starts from the `date` column, expands it by creating (and renaming) separate columns for `year`, `month`, and `day`, eventually yielding a `repaired_date` column. The other three modules are single-column subworkflows, each working on their own column (`event`, `dish_count`, and `physical_description`).

are then executed on the `event` column (with an additional “interruption” by an operation on `dish_count`). Only near the end of the workflow does attention return to the `repaired_date` column that had been created in earlier steps.

ORMA Modular View

Figure 6 again shows a table view on the far left (for orientation and comparison) and an automatically created **modular view** in the center, right. Upon close inspection of the schema view in Figure 5, one can already guess at least some elements of the subworkflow structure exposed by the modular view. As in the schema view, the modular view depicted in Figure 6 uses different colors to indicate the nature or type of the operations involved (e.g., generic, custom, type conversion). ORMA automatically detects all independent subworkflows (*modules*) based on the column-level dependencies obtained from recipes and the function signatures of operations. The modular view shown in Figure 6 still contains the original sequence numbers of workflow steps, which helps in matching steps across alternative views of the same workflow. In particular, the coarser-grained linear table view could be constructed from the modular view (but not vice versa). Modular subworkflows as the ones shown here can be used as reusable building blocks for composing larger data cleaning recipes from a library of independent modules. Different levels of transparency and “verbosity” can be chosen for the different views supported by ORMA: e.g., detailed parameter values can be included in the views, along with other

derived information (e.g., how many canonical names are part of a mass-edit and how many variant spellings are recognized for each canonical name). We have omitted the display of these additional details in our figure here.

Conclusions

We have presented an approach to improve the transparency and reusability of data cleaning recipes. The key idea is to model recipes as workflows over a suitable data model for describing data cleaning processes. We have prototypically implemented an OpenRefine model analyzer tool, ORMA (Li, 2021) which can be used to create different types of workflow views, from simple linear table views, via informative schema views, to modular views that automatically identify independent subworkflows that facilitate reuse. In future work, we plan to tackle several other technical challenges mentioned earlier, e.g., the history update problem or the recipe migration problem. Another direction for future work includes the use of retrospective provenance information to obtain even more powerful analysis methods based on hybrid models of provenance.

References

- Chapman, A., Belbin, L., Zermoglio, P., Wieczorek, J., Morris, P., Nicholls, M., Rees, E. R., Veiga, A., Thompson, A., Saraiva, A., et al. (2020). Developing standards for improved data quality and for selecting fit for use biodiversity data. *Biodiversity Information Science and Standards*, 4:e50889.
- Li, L. (2021). OpenRefine Model Analysis (ORMA) prototype demonstration repository. github.com/idaks/ORMA-IDCC-2021.
- Li, L., Parulian, N., and Ludäscher, B. (2021). or2yw: Generating yesworkflow models from openrefine histories. github.com/idaks/OR2YWTool.
- McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, R. K., Cao, Y., Cheney, J., Chirigati, F., Dey, S., Freire, J., et al. (2015). YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *International Journal of Digital Curation (IJDC)*, 10(1):298–313.
- McPhillips, T., Willis, C., Gryk, M. R., Nuñez-Corrales, S., and Ludäscher, B. (2019). Reproducibility by Other Means: Transparent Research Objects. In *15th International Conference on eScience*, pages 502–509.
- New York Public Library (2020). Whats on the menu? <http://menus.nypl.org/data>.
- Nosek, B. A., Alter, G., Banks, G. C., Borsboom, D., Bowman, S. D., Breckler, S. J., Buck, S., Chambers, C. D., Chin, G., Christensen, G., et al. (2015). Promoting an open research culture. *Science*, 348(6242):1422–1425.
- OR (2021). Openrefine: A free, open source, power tool for working with messy data. github.com/OpenRefine.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., and Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3:160018.