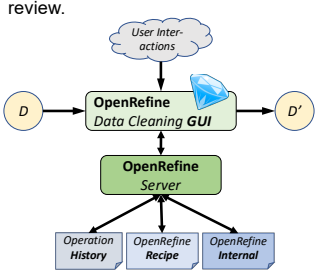
Modeling Provenance and Understanding Reproducibility for OpenRefine Data Cleaning Workflows

Timothy McPhillips, Lan Li, Nikolaus Parulian, and Bertram Ludäscher
School of Information Sciences, University of Illinois at Urbana-Champaign

Introduction

Revealing the steps taken to clean a dataset is critical to making any research using this data transparent and reproducible. OpenRefine [1] is a popular tool for exploring, profiling, and cleaning datasets using a spreadsheet-like interface. We report early results from an investigation into how records captured by OpenRefine can:

1. Facilitate reproduction of complete, real-world data cleaning workflows.
2. Support queries and visualizations of the provenance of cleaned datasets for review.



*OpenRefine stores records of operations carried out by a researcher in the process of transforming dataset D to yield cleaned D'. It exposes these records as a browsable **operation history** that serves as the interface to its **undo/redo feature**; and as exportable **recipes** that can be reused for other datasets.*

Goals & Desiderata

Modeling. OpenRefine has its own terminology for describing the records it keeps and the capabilities it provides using them. We are mapping the OpenRefine concepts of data, operations, and recipes to those of the reproducible research and provenance communities.

Reproducibility. *Recipes* exported from OpenRefine include neither the initial data *import step*, nor any edits made manually to *individual cells*. We are providing means to use the records kept by OpenRefine to facilitate end-to-end reproducibility of data cleaning workflows.

Transparency. Displaying lists of operations and the ability to revisit past states one at a time provide limited means for interrogating *how data was cleaned*. We aim to make data cleaning workflows and their products easy to *query* — ***prospectively*** and ***retrospectively*** — to answer any questions researchers may have about the provenance of cleaned datasets.

Tools & Methods

OpenRefine. We carry out our experiments on OpenRefine version 3.1 [1] in Java 8 environments on multiple platforms. We automate interactions with OpenRefine via its native HTTP API by using and extending the OpenRefine Python Client Library [2].

Queries. We employ XSB Prolog [3] for expressing and performing Datalog-style graph and provenance queries, and Graphviz for rendering visualizations of query results.

Reproducible computing environments. We depend on Ansible, Vagrant, and Docker for making research environments reproducible across coauthors' computers and enabling other researchers to repeat our experiments on their own systems.

Example – Work in Progress

```
%%%% State after initial import step %%%%%

% source(SourceId, SourceUri, SourceFormat).
source(source_1, 'biblio.csv', 'text/csv').

% dataset(DatasetId, SourceId).
dataset(dataset_1, source_1).

% array(ArrayId, DatasetId).
array(array_1, dataset_1).

% column(ColId, ArrayId).
column(col_1, array_1).
column(col_2, array_1).
column(col_3, array_1).

% row(RowId, ArrayId).
row(row_1, array_1).
row(row_2, array_1).
row(row_3, array_1).

% cell(CellId, ColId, RowId).
cell(cell_1, col_1, row_1). cell(cell_4, col_2, row_1). cell(cell_7, col_3, row_1).
cell(cell_2, col_1, row_2). cell(cell_5, col_2, row_2). cell(cell_8, col_3, row_2).
cell(cell_3, col_1, row_3). cell(cell_6, col_2, row_3). cell(cell_9, col_3, row_3).

% state(StateId, ArrayId, PrevStateId).
state(state_1, array_1, nil).

% content(ContentId, CellId, StateId, ValId, PrevContentId).
content(content_1, cell_1, state_1, val_1, nil).
content(content_2, cell_2, state_1, val_2, nil).
content(content_3, cell_3, state_1, val_3, nil).
content(content_4, cell_4, state_1, val_4, nil).
content(content_5, cell_5, state_1, val_5, nil).
content(content_6, cell_6, state_1, val_6, nil).
content(content_7, cell_7, state_1, val_7, nil).
content(content_8, cell_8, state_1, val_8, nil).
content(content_9, cell_9, state_1, val_9, nil).

% value(ValId, ValText).
value(val_1, 'Against Method').      value(val_4, 'Paul Feyerabend').  value(val_7, '1975').
value(val_2, 'Changing Order').      value(val_5, 'H.M. Collins').     value(val_8, '1985').
value(val_3, 'Exceeding our Grasp'). value(val_6, 'P. Kyle Stanford'). value(val_9, '2006').

% column_schema(ColSchemaId, ColId, StateId, ColType, ColName, PrevColId, PrevColSchemaId).
column_schema(col_schema_1, col_1, state_1, 'string', 'Book Title', nil, nil).
column_schema(col_schema_2, col_2, state_1, 'string', 'Author',     col_1, nil).
column_schema(col_schema_3, col_3, state_1, 'string', 'Date',       col_2, nil).

% row_position(RowPosId, RowId, StateId, PrevRowId).
row_position(row_pos_1, row_1, state_1, nil).
row_position(row_pos_2, row_2, state_1, row_1).
row_position(row_pos_3, row_3, state_1, row_2).

%%%% Rename column 1 from Book Title to Title %%%%%
state(state_2, array_1, state_1).
column_schema(col_schema_4, col_1, state_2, 'string', 'Title', nil, col_schema_1).

%%%% Place surname first and abbreviate given names to initials in Author column %%%%%
state(state_3, array_1, state_2).
value(val_10, 'Feyerabend, P.').
value(val_11, 'Collins, H.M.').
value(val_12, 'Stanford, P.K.').
content(content_10, cell_4, state_3, val_10, content_4).
content(content_11, cell_5, state_3, val_11, content_5).
content(content_12, cell_6, state_3, val_12, content_6).

%%%% Sort rows by Author %%%%%
state(state_4, array_1, state_3).
row_position(row_pos_4, row_2, state_4, nil).
row_position(row_pos_5, row_1, state_4, row_2).
row_position(row_pos_6, row_3, state_4, row_1).

%%%% Swap order of second and third columns %%%%%
state(state_5, array_1, state_4).
column_schema(col_schema_5, col_3, state_5, 'string', 'Date',   col_1, col_schema_3).
column_schema(col_schema_6, col_2, state_5, 'string', 'Author', col_3, col_schema_2).
```

	All	Book Title	Author	Date
1.		Against Method	Paul Feyerabend	1975
2.		Changing Order	H.M. Collins	1985
3.		Exceeding Our Grasp	P. Kyle Stanford	2006

	All	Title	Date	Author
1.		Changing Order	1985	Collins, H.M.
2.		Against Method	1975	Feyerabend, P
3.		Exceeding Our Grasp	2006	Stanford, P.K.

Queries and Reports – Examples

1. *What columns in the original dataset were renamed?*
2. *What columns in the final dataset contain cells with updated values?*
3. *What cells were assigned new values during the same step?*
4. *What fraction of columns had their schemas changed or contain values with updated cells?*
5. *What fraction of rows had cell values updated?*
6. *What column had the highest fraction of cells updated to new values?*
7. *Did any cell take on the same value at two or more different times?*
8. *What new columns were created?*
9. *What new rows were inserted, and which were deleted?*

Future Work

Dependencies. We plan to model and query data dependencies between values in a dataset, as well as on external databases used for reconciliation.

Parallel workflows. Using multiple arrays per dataset we expect to be able model data cleaning workflows that operate on different subsets of the data independently in parallel, while continuing to represent history accurately.

State-oriented logic languages. We will compare and analyze our explicit state management approach with designs enabled by state-oriented Datalog extensions such as Statelog [4] and Dedalus [5].

Strategies

Extract, transform, load. We extract the data cleaning records captured by OpenRefine and store them using a relational schema more amenable to query and visualization.

Objects with identities. Queries useful to researchers are posed and interpreted in light of the meanings of columns, rows, cells, and values in a dataset. We support such meaningful queries by assigning to these objects identifiers that persist as they are rearranged, renamed, reformatted, copied, transformed, corrected, and edited. And we associate each update of an object with the state at which the change was introduced.

Views of dataset states. OpenRefine natively provides access to past dataset states only by undoing all changes made subsequent to the state of interest. Our schema supports views and queries over multiple past states concurrently—without storing complete snapshots of the dataset for all states.

Separate concerns. The schema we use to represent the data-cleaning history of a dataset is independent of the columns, rows, and values comprising the dataset. A researcher neither needs to understand the schema representing the history, nor take into account the queries of interest to them prior to cleaning a dataset.

Logic programming. We support rapid experimentation and prototyping by representing data cleaning histories in formats easily queried using logic programming languages such as Datalog, Answer Set Programs, and Prolog.

Reproducible reproducibility research. Research into provenance and reproducibility should be held to the highest standards of reproducibility and transparency. To this end we employ reproducible computing environments and aim to make it easy for other researchers to repeat our observations and to evaluate our claims.

Demonstrations

Provenance queries. The data cleaning example workflow shown here, associated provenance queries, and supporting rules are available in the *openrefine-provenance* repository [6].

Reproducible data cleaning. We additionally demonstrate progress supporting end-to-end reproducibility of data cleaning workflows in the *openrefine-reproducibility* repository [7]. Our aim is to enable any data cleaning workflow later to be repeated automatically in a different instance of OpenRefine on the same or different dataset using information gathered by OpenRefine but not easily exported as recipes.

References

[1] OpenRefine: A free, open source, powerful tool for working with messy data. (2018) http://openrefine.org/.

[2] Makepeace, P., Lohmeier, F. OpenRefine Python client library (2018) https://github.com/openculturereconsulting/openrefine-client/.

[3] The XSB logic programming system, version 3.7 (2017).

[4] Lausen, G., Ludäscher, B., May, W. (1998) On Active Deductive Databases: The Statelog Approach. *Transactions and Change in Logic Databases*, LNCS 1472: 69–106.

[5] Hellerstein, J. (2010) The declarative imperative: experiences and conjectures in distributed logic. *ACM SIGMOD Record* 39.1: 5-19.

[6] https://github.com/idaks/openrefine-provenance (2019).

[7] https://github.com/idaks/openrefine-reproducibility (2019).

ILLINOIS

Supported in part by NSF grant #1541450