IJDC | *Conference Paper*

Reconciling Conflicting Data Curation Actions: Transparency Through Argumentation

Yilin Xia
School of Information Sciences,
University of Illinois, Urbana-Champaign

Shawn Bowers
Department of Computer Science
Gonzaga University

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

Bertram Ludäscher
School of Information Sciences,
University of Illinois, Urbana-Champaign

Abstract

We propose a new approach for modeling and reconciling conflicting data cleaning actions. Such conflicts arise naturally in collaborative data curation settings where multiple experts work independently and then aim to put their efforts together to improve and accelerate data cleaning. The key idea of our approach is to model conflicting updates as a formal *argumentation framework* (AF). Such argumentation frameworks can be automatically analyzed and solved by translating them to a logic program P_{AF} whose declarative semantics yield a transparent solution with many desirable properties, e.g., uncontroversial updates are accepted, unjustified ones are rejected, and the remaining ambiguities are exposed and presented to users for further analysis. After motivating the problem, we introduce our approach and illustrate it with a detailed running example introducing both well-founded and stable semantics to help understand the AF solutions. We have begun to develop open source tools and Jupyter notebooks that demonstrate the practicality of our approach. In future work we plan to develop a toolkit for conflict resolution that can be used in conjunction with OpenRefine, a popular interactive data cleaning tool.

Submitted 12 February 2024 ~ *Accepted* 22 February 2024

Correspondence should be addressed to Yilin Xia. Email: yilinx2@illinois.edu

This paper was presented at the International Digital Curation Conference IDCC24,19-21 February 2024.

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/

https://doi.org/10.2218/ijdc.v18i1.943
DOI: 10.2218/ijdc.v18i1.943

Introduction

Data curation and data wrangling are critically important, labor-intensive, and error-prone phases in data science. A popular claim is that about 80% of the effort involved in data analysis projects is spent on cleaning and preparing data sets (Dasu & Johnson, 2003; Wickham, 2014), while the subsequent analytical techniques often only constitute 20% of the effort. Not surprisingly, researchers and curators spend significant amounts of their time cleaning data, either with general purpose tools (e.g., Excel) and programming languages (e.g., Python, R), or using specialized tools such as OpenRefine (Verborgh & De Wilde, 2013) or Wrangler (Kandel et al., 2011).

A data cleaning *recipe* is a workflow W that describes the data cleaning *actions* (i.e., data transformations) that are performed on a "dirty" dataset D to improve its data quality and obtain a cleaner version $D' = W(D)$. Data analysis results are generally considered more trustworthy if the analysis pipeline—including the data cleaning workflow W—are *transparent* and *reproducible*. The state-of-the-art approach to increase transparency is to capture *provenance* information, preferably during the whole data-lifecyle, from data collection, through data wrangling, analysis, all the way to the scholarly publication and the creation of shared, digital research objects. In prior work, e.g., (Li et al., 2019; Parulian & Ludäscher, 2022, 2023), the value of prospective, retrospective, and hybrid provenance (i.e., combining the other two) has been demonstrated.

Collaborative Data Cleaning: A New Curation Challenge. In this paper, we consider the increasingly important setting where multiple researchers and curators work *collaboratively* on cleaning a dataset (Parulian, 2022). For example, a dataset D might be split in k-ways *horizontally*, i.e., $D = D_1 \cup \cdots \cup D_k$, based on a meaningful selection condition.[1] Another way to split the work and avoid merge conflicts is a *vertical* split, i.e., where experts are assigned specific columns (attributes) to work on. However, there are several reasons why data curation tasks cannot always be so neatly divided up: First, there are update operations that apply to disjoint regions (rows or columns) of a dataset, yet indirectly depend on each other, e.g., via logic dependencies such as *foreign keys*. We will not consider such indirect dependencies here (but explore them in future work). Another important use case involves the assignment of overlapping regions of D to multiple curators, e.g., because a clear (horizontal or vertical) cut is difficult to make, or because the overall data cleaning process can benefit from the *diversity of expertise*, in which case overlapping assignments are even desirable.

Resolving Conflicts Transparently Through Argumentation. It is easy to see that in collaborative settings, two update actions A and B can be in *conflict*: e.g., an existing value v_1 might be updated to v_2 by A but to a different value v_3 by B. Clearly the actions A and B are mutually exclusive. Asymmetric conflicts can also arise: If an update A applies to a row that another action C is deleting (for good reasons), then one could argue that A should be rejected, since the update through A is moot because the row no longer exists. In the following we propose to model conflicting data cleaning actions $A, B, C, \ldots$, as *arguments* in a formal *argumentation framework* (AF) (Dung, 1995). Such argumentation frameworks can be automatically analyzed and solved by translating them to a logic program P_{AF} whose declarative, well-founded semantics (Van Gelder et al.,

[1] An ecology dataset, e.g., may be split by species and then assigned to different domain experts.

1991) yields a *transparent* solution with many desirable properties: e.g., uncontroversial updates are *accepted*, unjustified ones are *rejected*, and the remaining *ambiguities* are exposed and presented to the user for further analysis e.g., via stable models (Gelfond & Lifschitz, 1988) and conflict resolution. If a researcher, curator, or auditor questions why certain updates have been accepted, while others have been rejected, the underlying AF solution, enhanced with a game-theoretic provenance semantics (Ludäscher et al., 2023) can be explored (interactively if desired) to provide a transparent, logical justification. In the following, we briefly review some background, then introduce our approach and illustrate it with a detailed running example. In the final section, we summarize and discuss plans for future work.

Background & Preliminaries

An *argumentation framework* AF is a finite, directed graph $G_{\mathsf{AF}} = (V, E)$, whose vertices V denote atomic *arguments* and whose edges $E \subseteq V \times V$ denote a binary *attacks* relation. An edge $(x, y) \in E$ states that argument x *attacks* argument y.

(a) *Attack* graph **(b)** *Grounded* extension **(c)** *Stable* extension #1 **(d)** *Stable* extension #2

Figure 1. (a) AF with four arguments a, b, c, d and their *attack* relation. (b) The unique, 3-valued *grounded* solution: a is *accepted* (blue), b is *defeated* (orange), and c, d are *undecided* (yellow). G_{AF} has two *stable* solutions: The undecided argument c can be chosen as accepted and d as defeated, as in (c), or vice versa as in (d), yielding two separate stable solutions.

An example AF consisting of four arguments (vertices) $V = \{\mathsf{a}, \ldots, \mathsf{d}\}$ and an *attack* relation E (directed edges) is shown in Figure 1. A subset $S \subseteq V$ of acceptable arguments is called an *extension* (or *solution*), provided S satisfies certain conditions. An extension S is said to *attack* an argument x if an argument $y \in S$ attacks x. The *attackers* of S are the arguments that attack at least one argument in S. An extension S is *conflict-free* if no argument in S attacks another argument in S. Conversely, an extension S *defends* an argument x if it attacks all attackers of x. The arguments *defended by* S are those that S defends; this is often described via the *characteristic function* of an argumentation framework. (Dung, 1995) and others have defined various *extension semantics*. We consider the skeptical *grounded extension* semantics, which has several advantages, e.g., it can be efficiently computed, always yields a unique, 3-valued model in which arguments (and thus edit actions) are either *accepted*, *rejected*, or flagged as *undecided*. We will also consider *stable extensions*, i.e., 2-valued solutions that refine the grounded solution by choosing acceptance or defeat of arguments in certain ways (Baroni et al., 2018).

The overall idea and appeal of formal argumentation results from the fact that the

solutions to controversial arguments can be computed automatically. As it turns out the unique well-founded model (Van Gelder et al., 1991) (and the set of stable solutions) of an argumentation framework can be obtained from a simple but powerful recursive rule:

$$\mathsf{defeated}(X) \leftarrow \mathsf{attacks}(Y, X), \neg\, \mathsf{defeated}(Y). \qquad (P_{\mathsf{AF}})$$

The rule states that an argument X is *defeated* (in our terminology: a curation action is *rejected*), if there exists an argument Y that attacks it and that is not itself defeated, i.e., accepted in our data curation terminology. Note that the AF approach, according to (Dung, 1995), consists of two essential components: an *argument generation unit* (AGU) that models the arguments and the associated attack graph (in our case the data curation actions and their conflicts), and an APU (the P_{AF} above) that is used to *solve* an AF and determine which arguments are accepted, rejected, and undecided, respectively. For more on formal argumentation, see the comprehensive handbook by (Baroni et al., 2018).

A Running Example for Data Cleaning

We illustrate the key ideas of our approach to conflict resolution in collaborative curation settings with a running example. Assume that there are two data curators, called Alice and Bob, respectively, who are working independently on cleaning a dataset D. This dataset (Parulian & Ludäscher, 2023) consists of texts in the philosophy of science, a snippet of which is shown in Table 1. Each entry includes the title of the book, the author's name, and the year of publication. The task for Alice and Bob is to create a new column that adheres to the APA style guidelines for in-text citations, i.e., which require the author's last name and the year of publication.

Table 1. Example dataset provided to Alice and Bob for cleaning ("␣" denotes a whitespace).

Book Title	Author	Date
Against Method	Feyerabend, P.	1975
Changing Order	Collins, H.M.	␣␣1985 ␣
Exceeding Our Grasp	Kyle Stanford	2006
Theory of Information		1992

For cleaning D, Alice and Bob employ several *data cleaning operations* from OpenRefine. The subset of operations used by Alice and Bob is shown in Table 2, along with their parameters. These include schema-level, row-level, and cell-level operations (Li et al., 2021). Note that the split_col operation in OpenRefine automatically creates new columns, whereas the name of the new column created by the rename and join_col operations must be explicitly given via the *new_column_name* parameter.

Alice and Bob each execute their own data cleaning *recipe* (i.e., a sequence of data cleaning actions); see Table 3. Unfortunately, they arrive at distinct outcomes as can be seen from the two different results in Table 4. While the two result datasets are similar, there are also differences, e.g., in the Citation column. Moreover, Alice's table contains only three rows, in contrast to Bob's four as in the initial dataset (Table 1). Additionally,

Table 2. Data cleaning operations used by Alice and Bob (and available, e.g., in OpenRefine).

Data Cleaning Operation	Description
cell_edit(*row_id*, *column_name*, *new_value*)	OpenRefine's single cell edit function, allowing users to hover over a cell and click "Edit" to modify its value.
del_row(*row_id*)	Deletes a row by using the "Facet" feature, selecting a relevant condition, followed by "Remove Matching Rows".
del_col(*column_name*)	Removes a column by going to "Edit Column" and selecting "Remove this Column".
split_col(*column_name*, *separator*)	Accessed via "Edit Column" > "Split into several columns", this function splits a column into multiple ones using a specified separator and keeps the original column.
transform(*column_name*, *function*)	Found under "Edit cells" > "Transform...", it allows the transformation of column values using the General Refine Expression Language (GREL).
join_col(*set_of_column_names*, *separator*, *new_column_name*)	Combines multiple columns into a new one with a specific separator via "Edit column" > "Join columns...".
rename(*column_name*, *new_column_name*)	Rename a column under "Edit column" > "Rename the column...".

the columns differ: Alice's version includes an "Author 1" column, while Bob's version separates the author information into "Last Name" and "First Name" columns.

The key idea of our approach, described in the following sections, is to model data cleaning actions as arguments to perform the desired updates and then treat conflicting actions (like those by Alice and Bob) as arguments that can attack one another, in the sense of argumentation frameworks. By computing solutions (extensions) of the resulting argumentation frameworks, different reconciliation solutions to the conflicting recipes can be obtained automatically.

Table 3. Data cleaning recipes by Alice and Bob. Steps correspond to OpenRefine operations.

Step	Alice's Data Cleaning Steps	Effects of the Data Cleaning Operations
E	rename("Book Title", "Book-Title")	Replace whitespace in column name with '-' to simplify data manipulation
F	cell_edit(3, "Author", "Stanford, P.")	Edit cell value to make it consistent with the pattern from other cells
G	transform("Date", "value.toNumber()")	Data type conversion
H	del_row(4)	Remove a row with a missing cell value
I	split_col("Author", ",")	Extract the lastname from the "Author" column
J	del_col("Author 2")	Remove an unnecessary column
K	join_col("Author 1", "Date", "," , "Citation")	Create an in-text Citation column by combining two other columns

Step	Bob's Data Cleaning Steps	Effects of the Data Cleaning Operations
L	rename("Book Title", "Book_Title")	Replace whitespace in column name with '_' to simplify data manipulation
M	transform("Date", "value.trim()")	Trim whitespace characters in string value
N	cell_edit(4, "Author", "Shannon, C.E.")	Add missing information
O	cell_edit(3, "Author", "Stanford, P.K.")	Edit cell value to make it consistent with the pattern from other cells
P	split_col("Author", ",")	Extract the lastname from the "Author" column
Q	rename("Author 1", "Last Name")	Replace the column name with a more meaningful one
R	rename("Author 2", "First Name")	Replace the column name with a more meaningful one
S	join_col("Last Name", "Date", "," , "Citation")	Create an in-text Citation column by combining two other columns

Table 4. Data cleaning results for Alice (top) and Bob (bottom). Values depicted in light green have been converted to a numeric data type (all other columns have type string).

Book-Title	Author	Date	Author 1	Citation
Against Method	Feyerabend, P.	1975	Feyerabend	Feyerabend, 1975
Changing Order	Collins, H.M.	1985	Collins	Collins, 1985
Exceeding Our Grasp	Stanford, P.	2006	Stanford	Stanford, 2006

Book_Title	Author	Date	Last Name	First Name	Citation
Against Method	Feyerabend, P.	1975	Feyerabend	P.	Feyerabend, 1975
Changing Order	Collins, H.M.	1985	Collins	H.M.	Collins, 1985
Exceeding Our Grasp	Stanford, P.K.	2006	Stanford	P.K.	Stanford, 2006
Theory of Information	Shannon, C.E.	1992	Shannon	C.E.	Shannon, 1992

Modeling Data Cleaning Conflicts as Argumentation Frameworks

The key idea of our approach is to treat data curation actions as *arguments*, i.e., a curator claims that the corresponding operation is desirable or necessary for cleaning the data. Conflicting operations A and B from two different recipes are then modeled as *attacks*.

Table 5. Operation Conflicts: This matrix illustrates one possible conflict (attack) relationship between pairs of data operations A and B. For readability, the upper half of the matrix is omitted (as it can be deduced from the lower half by reversing the attack relation).

	Operation B						
Operation A	cell_edit(r, c, v_2)	del_row(r)	del_col(c)	split_col(c, sp_2)	transform(c, f_2)	join_col(c, ...c_j, sp_2, cn_2)	rename(c, c_2)
cell_edit(r, c, v_1)	A ↔ B						
del_row(r)	A → B	∅					
del_col(c)	A → B	∅	∅				
split_col(c, sp_1)	A ← B	∅	A ← B	∅			
transform(c, f_1)	A ↔ B	∅	A ← B	A → B	A ↔ B		
join_col(c, ...c_i, sp_1, cn_1)	A ← B	∅	A ← B	∅	A ← B	∅	
rename(c, c_1)	A → B	∅	A ↔ B	A → B	A → B	A → B	A ↔ B

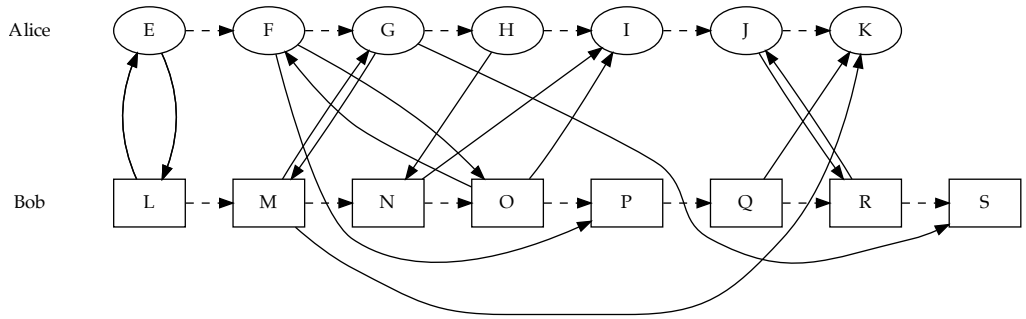
For example, $A \leftrightarrow B$ (mutual attack) means that A and B attack each other, so only one of them should be executed. Consider, e.g., the actions cell_edit(r , c, v_1) and cell_edit(r , c, v_2). They are considered a mutual attack whenever $v_1 \neq v_2$: Both curators agree that the cell in row r and column c need to be changed, but disagree on what the new value should be. Thus cell_edit(r , c, v_1) and cell_edit(r , c, v_2) are attacking each other, denoted cell_edit(r , c, v_1) ↔ cell_edit(r , c, v_2).

Conversely, $A \rightarrow B$ means that if A is accepted then B is rejected (but not vice versa). For example, if a curator C_1 wants to delete a row r and curator C_2 wants to edit a cell-value in r (so $A =$ del_row(r) and $B =$ cell_edit(r , c, v_2)), we could argue that B should be rejected, either because it works on a cell that has been deleted already, or it performs an edit on a cell that is about to be deleted. Table 5 specifies that in such cases, deletions take priority over edits.

Another asymmetric attack relations occurs, e.g., between operations transform(c, f_1) andsplit_col(c, sp_2). First, note that these two operations are not commutative, i.e., the

Attacks	Description
E ↔ L	rename("Book Title", "Book-Title") ↔ rename("Book Title", "Book_Title")
F ↔ O	cell_edit(3, "Author", "Stanford, P.") ↔ cell_edit(3, "Author", "Stanford, P.K.")
J ↔ R	del_col("Author 2") ↔ rename("Author 2", "First Name")
G ↔ M	transform("Date", "value.toNumber()") ↔ transform("Date", "value.trim()")
K ← Q	join_col("Author 1", "Date", "," , "Citation") ← rename("Author 1", "Last Name")
H → N	del_row(4) → cell_edit(4, "Author", "Shannon, C.E.")
I ← N, O	split_col("Author", ",") ← cell_edit(4, "Author", "Shannon, C.E."), cell_edit(3, "Author", "Stanford, P.K.")
F → P	cell_edit(3, "Author", "Stanford, P.") → split_col("Author", ",")
K ← M	join_col("Author 1", "Date", "," , "Citation") ← transform("Date", "value.trim()")
G → S	transform("Date", "value.toNumber()") → join_col("Last Name", "Date", "," , "Citation")

(a) Abstract attack relations and description of underlying data cleaning operations (cf. Table 3)



(b) Argumentation Framework (solid edges) and recipe execution order (dashed edges)

Figure 2. Individual attack relations and visualized attack graph (with recipe execution order)

result depends on the execution order. Here, for simplicity, we argue that a data cleaning transformation f_1 on column c should take priority over a column-split operation on c.[2]

By modeling the conflicts between Alice's and Bob's recipes as specified in Table 5, we obtain the attack relation described in Figure 2(a). Additionally, this attack relationship is visualized in Figure 2(b). Note that mutual attacks are displayed using two attack edges e.g., $E \longrightarrow L$ and $L \longrightarrow E$. Operations by Alice are shown as ovals, those by Bob are depicted as boxes. Dashed lines are not attack relations but represent the *execution order* of operations within a curator's recipe.

Solving AFs to Explain DC Conflicts

After modeling data-cleaning recipes as attack graphs, the corresponding grounded and stable extensions can: (*i*) help users better understand the conflicts among the recipe actions; and (*ii*) provide guidance on how to resolve the conflicts among actions to generate one or more unified (i.e., *merged*) recipes. In particular, given a solved attack graph built from the recipes, we assume a merged recipe will contain the accepted actions of the corresponding attack graphs (and will not include the rejected actions). Under the grounded semantics, actions that are undecided (i.e., neither accepted nor rejected), require further analysis by users for inclusion in the merged recipe. The stable-model

[2] Instead of rejecting the column-split operation, it might be preferable to impose an execution order, i.e., first execute f_1 and then split column c.

semantics can then be employed to enumerate the possibilities for inclusion of the remaining (undecided) actions. Specifically, after viewing the different stable extensions, a user could select the one that they deem to be most appropriate for resolving the remaining conflicts, adding the corresponding accepted actions to create a final, merged recipe.

As an example, Figure 3 shows the solved attack graphs of Figure 2 under the grounded semantics, where actions H and Q are accepted, N and K are rejected, and the remaining actions are undecided. Alice's action H (deletion of row 4) is accepted because there is no other action that attacks it. Because H is accepted and attacks N, it follows that Bob's action N (to edit row 4) is rejected. Similarly, Bob's action Q (to rename column "Author 1") is accepted because there is no other action that attacks it. Because Q is accepted and attacks K, it follows that Alice's action K (which required "Author 1" for a join operation) is rejected. Note, however, that this still leaves the remaining actions of Figure 3 unresolved.

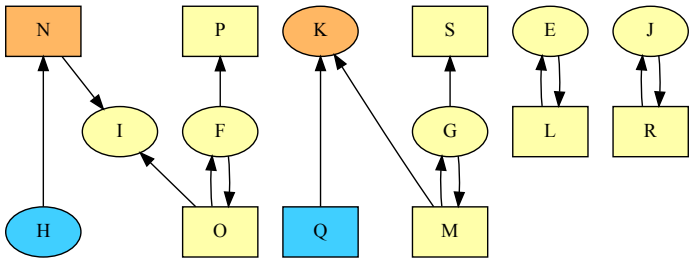
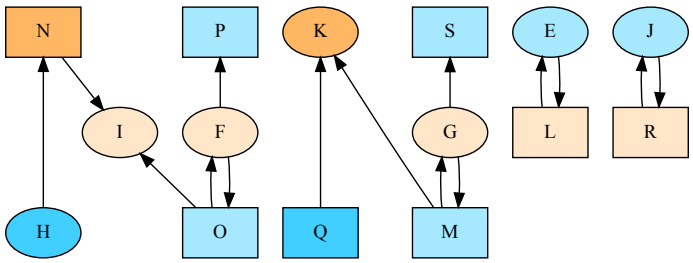


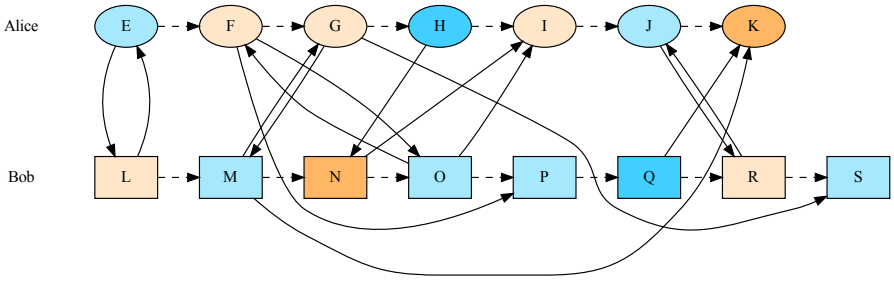
Figure 3. The grounded extension of Figure 2 where (blue) actions are accepted, (orange) actions are rejected, and (yellow) actions are undecided.

To help resolve the remaining conflicts, the stable models can be computed (of which there are 16 distinct solutions), one of which is shown in Figure 4(a). As shown in the figure, the stable model proposes to accept Alice's actions E and J along with Bob's actions M, O, P, and S. Figure 4(b) shows the same stable model as in Figure 4(a) but displayed according to Alice's and Bob's order of actions in their respective data-cleaning recipes.

Assuming that the stable model of Figure 4 is selected as the best resolution of the conflicting actions in Alice's and Bob's recipes, a corresponding merged recipe is shown in Table 6. The merged recipe adheres to the order of actions in the respective recipes of Alice and Bob: $E \dashrightarrow H \dashrightarrow J$ (from Alice) and $M \dashrightarrow O \dashrightarrow P \dashrightarrow Q \dashrightarrow S$ (from Bob). Note that the specific ordering shown in Table 6 is not the only possible ordering of the actions. However, both the relative ordering of Alice's and Bob's recipe must be maintained as well as an overall ordering that generates an appropriate final data product. Specifically, in the running example, the objective is to generate a column that incorporates APA in-text citations, which in this case means that Bob's action S must be the final step of the merged recipe. Moreover, there can exist additional order-based dependencies among accepted actions from different data curators. As an example, in Table 6, Bob's action P must occur before Alice's action J since P splits the "Author" column and J removes "Author 2"(generated by the split).



(a) One of the 16 possible stable solutions



(b) The stable solution in sequence order according to the original recipes of Alice and Bob

Figure 4. A stable extension of Figure 3 with actions appearing in light blue being additionally accepted and those appearing light orange additionally being rejected.

Table 6. One possible recipe when merging Alice and Bob's Actions.

Argument	Action	Data Curator
E	rename("Book Title", "Book-Title")	Alice
M	transform("Date", "value.trim()")	Bob
H	del_row(4)	Alice
O	cell_edit(3, "Author", "Stanford, P.K.")	Bob
P	split_col("Author", ",")	Bob
J	del_col("Author 2")	Alice
Q	rename("Author 1", "Last Name")	Bob
S	join_col("Last Name", "Date", "," , "Citation")	Bob

Table 7. The result of the merged recipe in Table 6 on the intial dataset in Table 1.

Book-Title	Author	Date	Last Name	Citation
Against Method	Feyerabend, P.	1975	Feyerabend	Feyerabend, 1975
Changing Order	Collins, H.M.	1985	Collins	Collins, 1985
Exceeding Our Grasp	Stanford, P.K.	2006	Stanford	Stanford, 2006

Finally, the result of applying the actions of the merged recipe in Table 6 over the initial dataset of Table 1 using OpenRefine is shown in Table 7.

Discussion & Future Work

The process of cleaning large and complex data sets can be time intensive and can require the work of multiple experts. However, conflicts can naturally arise in such collaborative data curation settings where multiple experts work independently on the same or overlapping regions of a dataset. This paper describes an approach based on formal argumentation frameworks for modeling the actions of users' data-cleaning recipes, identifying conflicting actions across recipes, and providing users with new tools to help resolve these conflicts to generate a single, unified, merged recipe. The recipe can then be used over the original dataset to produce the final cleaned data product. By leveraging the grounded semantics of formal argumentation frameworks, it is possible to identify an initial set of accepted and rejected actions. When ambiguity is still present, the remaining actions can be resolved by selecting among one of the potentially many stable extensions. While the use of argumentation frameworks has been employed previously for resolving the justifications for specific data-cleaning actions (see Santos & Galhardas, 2011), our work focuses on leveraging (and ultimately extending) systems such as OpenRefine, that provide a wide-range of data cleaning actions, to create new tooling for reasoning, visualizing, and automatically generating merged data-cleaning recipes. In this paper, we have described the underlying approach through a concrete data-cleaning example, highlighting the general idea and advantages of such tools. Finally, we have begun developing open-source software and corresponding Jupyter notebooks that demonstrate the practicality of the approach (Xia & Ludäscher, 2023). In future work we plan to develop a full-featured toolkit for conflict resolution that can be used within OpenRefine to support collaborate data cleaning projects.

References

Baroni, P., Gabbay, D., Giacomin, M., & Torre, L. v. d. (2018). *Handbook of Formal Argumentation*. London, England: College Publications.

Dasu, T., & Johnson, T. (2003). *Exploratory data mining and data cleaning*. John Wiley & Sons.

Dung, P. M. (1995). On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *AI*, *77*(2), 321–357.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. *ICLP/SLP*, *88*, 1070–1080.

Kandel, S., Paepcke, A., Hellerstein, J., & Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. *Proceedings of the sigchi conference on human factors in computing systems*, 3363–3372.

Li, L., Ludäscher, B., & Zhang, Q. (2019). Towards more transparent, reproducible, and reusable data cleaning with openrefine. *iConference 2019 Proceedings*.

Li, L., Parulian, N. N., & Ludäscher, B. (2021). Automatic Module Detection in Data Cleaning Workflows: Enabling Transparency and Recipe Reuse [https://doi.org/10.2218/ijdc.v16i1.771]. *16th International Digital Curation Conference (IDCC)*. doi:10.5281/zenodo.5606219.

Ludäscher, B., Bowers, S., & Xia, Y. (2023). Games, queries, and argumentation frameworks: Towards a family reunion [Accepted for publication]. *7th Workshop on Advances in Argumentation in Artificial Intelligence (AI^3)*.

Parulian, N. (2022, June). *Conceptual Model and Framework for Collaborative Data Cleaning*. https://zenodo.org/records/6781134.

Parulian, N., & Ludäscher, B. (2022). DCM Explorer: A tool to support transparent data cleaning through provenance exploration. *14th Intl. Workshop on the Theory and Practice of Provenance (TaPP)*, 1–6.

Parulian, N., & Ludäscher, B. (2023). Trust the process: Analyzing prospective provenance for data cleaning. *Companion Proceedings of the ACM Web Conference 2023*, 1513–1523.

Santos, E., & Galhardas, H. (2011). Using Argumentation to Support the User Involvement In Data Cleaning. *9th International Workshop on Quality in Databases (QDB)*. http://qdb2011.dia.uniroma3.it/participants/program/index.html

Van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The Well-founded Semantics for General Logic Programs. *Journal of the ACM*, *38*(3), 619–649.

Verborgh, R., & De Wilde, M. (2013). *Using OpenRefine*. Packt Publishing Ltd.[article]

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, *059*. Retrieved January 24, 2018, from https://doi.org/10.18637/jss.v059.i10

Xia, Y., & Ludäscher, B. (2023, August). Games and argumentation demo repository [github.com/idaks/Games-and-Argumentation/tree/idcc].