Minimizing User Effort in Transforming Data by Example[*]

Bo Wu
Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey,CA
bowu@isi.edu

Pedro Szekely
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA
pszekely@isi.edu

Craig A. Knoblock
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA
knoblock@isi.edu

ABSTRACT
Programming by example enables users to transform data formats without coding. To be practical, the method must synthesize the correct transformation with minimal user input. We present a method that minimizes user effort by color-coding the transformation result and recommending specific records where the user should provide examples. Simulation results and a user study show that our method significantly reduces user effort and increases the success rate for synthesizing correct transformation programs by example.

Author Keywords
Data Transformation, Programming by Example, Recommendation

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation: User Interfaces

INTRODUCTION
Many mashup applications rely on data from multiple sources. Unfortunately, using data from multiple sources often requires format transformation, which is highly task dependent. It often requires the user to specify the transformation, which is typically done by writing scripts [4] or demonstrating a sequence of edit operations [5] [6]. In order to ease the burden of specifying the transformations, Gulwani [3] developed a programming by example approach.

[*]This research was supported in part by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-10-C-7058. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, AFRL, or the U.S. Government.

IUI'14, February 24–27, 2014, Haifa, Israel.
Copyright © 2014 ACM 978-1-4503-2184-6/14/02..$15.00.
http://dx.doi.org/10.1145/2557500.2557523

Table 1: Text Format Transformation Scenario

Raw Data	Target Data
Lois Anderson	Anderson, Lois
knud Merrild	Merrild, knud
robert boardman howard	howard, robert boardman
...	...
William J. Forsyth	Forsyth, William J.
John G. Dunn	Dunn, John G.

Programming by example approach asks a user to provide examples to synthesize the transformation program instead of having her write a program directly. In Table 1, a user wants to change the name format. To do this, a user only needs to provide an example for a record showing the target value. For instance, she would provide "Anderson, Lois" for the first record. Once she provides that example, the approach automatically generates the transformation program that is consistent with the examples. It then applies this program to the rest of the data and transforms those records automatically. The user then examines the results. If any record is transformed incorrectly, she enters the target value for that record. This process goes through multiple rounds until all the results are correct. During the process, she actively takes part in these activities:

1. Examining the results
2. Deciding which example to provide

All these activities repeat for several rounds until the user stops. Since a dataset can easily have thousands of records, the activities listed above can be very labor intensive and error prone. We identify several challenges as follows.

Firstly, the user needs to find incorrectly transformed results from possibly thousands of records. These records can either cause the transformation program to exit abnormally or simply have incorrect results. The user must identify these records so that the synthesizing approach can refine the program to handle the unseen cases correctly.

Secondly, the user cannot easily tell which record to label. Using different records as examples provides different amounts of information for the program to learn. Using the most informative record reduces the total number of examples that the user needs to provide.

Thirdly, the user does not know whether the synthesized transformation is as she expected. As the user generally does not want to read the synthesized program, the system does not typically show it. She usually has no idea of what transformations are synthesized and applied.

In order to address the challenges above, we developed an approach to recommend records for the user and visualize the transformation to facilitate the decision-making on which example she should provide. By recommending records and color-coding the transformations, our system can reduce user effort in making decisions on which record to provide as an example and make the user aware of incorrectly transformed results. This approach to recommending records and visualizing the transformations provides the following contributions:

1. Reduces the number of iterations
2. Reduces the effort in examining the results
3. Increases the success rate

SYNTHESIZING TRANSFORMATION PROGRAMS

Gulwani [3] developed a programming by example approach that defines a string transformation language. This language supports a restricted, but expressive form of regular expressions, which includes conditionals and loops. The approach synthesizes transformation programs from this language using examples. The transformation program is a concatenation of several segments. Each segment consists of a start and an end position expression.

As shown in Figure 1, the program consists of four segments: last name, comma, blank space and first name segment. A segment can either be a constant string like the comma segment or describe how to extract the corresponding parts from the raw data. The second type of segment has two position expressions: start and end position expressions. The positions can be specified using (1) an absolute position, or (2) restricted regular expressions that identify the context of the given position, which can be represented as (leftcxt, rightcxt, occ). "leftcxt" describes the left context of the position, "rightcxt" describes the right context and"occ" is the occurrence of the position. For example, the start position of "Lois" can be specified as (START, UWRD, 1) or an absolute position 0. Here, "START" represents the beginning of the raw value. "END" is for the end of the raw value. "UWRD" represents an uppercase letter. "LWRD" means a continuing sequence of lower letters. The "BNK" means a blank space. Therefore, (START,UWRD, 1) means the first occurrence of a position, which is at the beginning of the raw data and has a uppercase token at its right.

In order to generate such transformation programs, Gulwani's approach first tokenizes the original value and target value of an example into two token sequences. He then groups the target token sequence into different segments. The approach then determines different ways to generate these segments from the original token sequence. As shown in Figure 1, "Anderson, Lois" is divided into multiple segments. The segments shown in the brackets are "Anderson", ",", " ", and "Lois". These segments are then concatenated in order to make up the transformation program.

If there are multiple examples, the approach uses an efficient algorithm to construct a version space for each example and

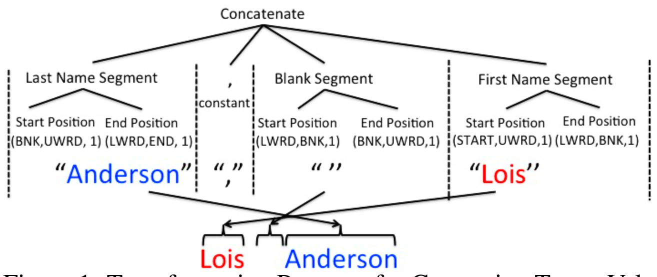


Figure 1: Transformation Program for Generating Target Values from Raw Values

to merge them into a version space consistent with the maximum number of examples. However, if the examples cannot be covered by a single version space, the algorithm partitions the examples and generates a version space for each partition individually. A conditional expression can be learned to distinguish multiple different version spaces. This approach also supports loop expressions by detecting whether continuous segments of one program can be merged.

APPROACH OVERVIEW

Our approach builds upon Gulwani's approach [3] to recommend incorrectly transformed records using active learning. Our system first recommend the most informative record from the records that cause transformation program to exit with error. When all the records are successfully transformed, it will then recommend the most questionable record that may have incorrect results.

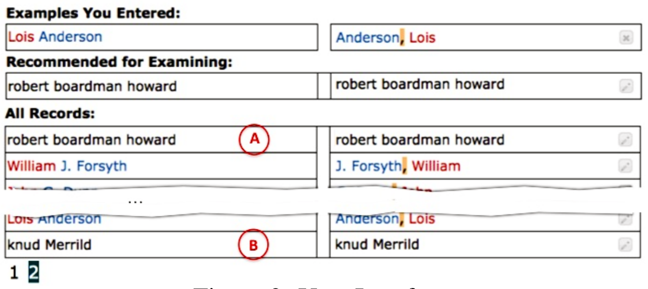
The screenshot of our user interface in Figure 2 has the following areas. (1) "Examples You Entered" shows all previous examples. (2) "Recommended for Examining" shows our recommended record. (3) "All Records" shows all the records in a multi-page table. On the left are raw data and on the right are transformed values.

The user first checks whether the recommended record is correct. If it is not, she can provide the target value for this entry to teach the system to learn this new variation. If the result is already correct, she can then check the transformed values in the "All Records" area to identify any incorrect result. As she might type an incorrect example by accident, she can easily find all the previous examples and cancel the one with an error.

We can see that all the original values and transformed values are color coded, which shows the correspondence between substrings in the transformed values and substrings in the original values. This color-coding can help the user understand what transformation is currently applied to the data and also makes it easier to identify incorrect results by displaying irregular color patterns.

RECOMMENDING INFORMATIVE RECORDS

Certain records can cause the transformation program to exit abnormally. For example in Figure 2, the user entered "Anderson, Lois" as the target value for "Lois Anderson". The synthesized program exits with an error on both record A and B. (Our system keeps the values the same as the raw values


Examples You Entered:

Lois Anderson	Anderson, Lois

Recommended for Examining:

robert boardman howard	robert boardman howard

All Records:

robert boardman howard	A	robert boardman howard
William J. Forsyth		J. Forsyth, William
...		
Lois Anderson		Anderson, Lois
knud Merrild	B	knud Merrild

1 2

Figure 2: User Interface

Table 2: Fail or Success for Each position Expression

Record	LStart	LEnd	BStart	BEnd	FStart	FEnd
A	0	1	1	0	0	1
B	1	1	1	1	0	1

and will not color-code these records.) Choosing the informative record to provide an example can minimize the number of examples to synthesize the correct program.

Our approach recommends such informative records for the user. As we mentioned before, the transformation program consists of a number of position expressions, which specify how to locate the start or end position of a segment. Any position expression that fails to locate any position on a record will cause the transformation program to fail in that record. By tracking the interpretation results of each individual position expression, we can get the number of failed position expressions for each record and then identify the record with the largest number of failed position expressions.

Table 2 shows the interpreted results for each position expression on two records. L represents last name, B represents blank and F represents first name. "LStart" means the start position expression for last name. According to Table 2, record A fails on both the last name and first name start position expression, while the record B fails on the first name start position. As seen from Figure 1, the last name start position expression is (BNK, UWRD, 1) and the first name start position is (START, UWRD). Both expressions require the right context to be an upper case token. However, record B has a lower case first name, while both the first name and last name in record A are lower case token. Therefore, if we provide an example for record A, we can teach the transformation program that both first name and last name can be lower case words.

Our program may contain conditional expression and loop expression. For conditional expression, our approach evaluates the condition of the record and chooses the correct program for the record. We then calculate the score as described before. For loop expression, the loop will break if any mismatch happens. Therefore, we actually use the number of mismatches in the last iteration to calculate the score for the record.

To identify the most informative record, we count the number of failed position expressions for each record and recommend the one with the largest number of failed position expressions.

RECOMMENDING QUESTIONABLE RECORDS

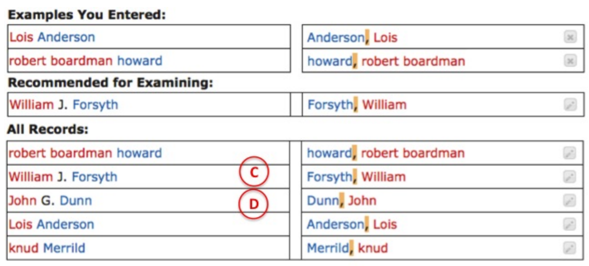
Records can have incorrect results, which requires the user to examine the results more carefully to detect the error. For example, in Figure 3 the synthesized program generates results for all records. However, after scrutinizing the data, we can see the results for record C and record D are still incorrect. The end position expression of the first or middle name is (LWRD,BNK,-1) after entering two examples as shown in Figure 3. The expression aims to find a location, whose left is a lower case token and right is a blank. "-1" means the first occurrence when searching backwards. As record C and D use "." as the left context for the end position, which doesn't match the condition, the expression cannot match at the correct position. Instead, it keeps scanning backwards until it reaches the end of the first word. Although the result satisfies the position expression, it is incorrect.

To capture these incorrect results, we found the transformation can also be evaluated based on the content before and after the transformation. From Figure 3, we can see that the examples keep all the content in the raw value. However, for record C and D, the middle names do not get copied to the transformed value. We can compare other records against the examples. Thus, the record that is the most different from the examples is most likely to be incorrect. We recommend this record as questionable record for the user to examine.

We represent the transformation using a set of features. There are two types of features:

Transformation features: These features aim to capture the content changed after applying the transformation program. We use these features to represent the transformation.

1. Token count difference: this type of features calculates the difference between the token counts before and after the transformation. Taking "." as an example, it appears once in record C before transformation and appears 0 times after. Thus, the feature value for "." count difference is 1 for record C. We track the count difference for a set of tokens such as all the punctuation, numbers and tokens appearing in at least 10% records. Each token in this set will be a feature in the final feature vector.
2. Reorder: this type of feature calculates the inversion number of the target string. We assign each token in the result a number o_i, which is its order in the original token sequence. We then compute the inversion number using $count(\{(o_i, o_j) \mid i < j \, and \, o_i > o_j\})$.


Examples You Entered:

Lois Anderson	Anderson, Lois
robert boardman howard	howard, robert boardman

Recommended for Examining:

William J. Forsyth	Forsyth, William

All Records:

robert boardman howard		howard, robert boardman
William J. Forsyth	C	Forsyth, William
John G. Dunn	D	Dunn, John
Lois Anderson		Anderson, Lois
knud Merrild		Merrild, knud

Figure 3: Transformed Results are Incorrect

Result format features: these features aim to capture the format of transformed result. Using these features, we can identify the records that have different formats from examples. We use the counts of various tokens in the result for these features. Each token count corresponds to one result format feature. The tokens used in this type of feature are the same as the tokens used in token count difference features.

Given the feature set, we follow these steps to detect the questionable record:

1. Convert all pairs of raw and transformed values into feature vectors V_i using the features defined above. Convert all examples into features vectors E_i in the same way.
2. Obtain the mean vector $\bar{E}$ of the m examples by $\bar{E} = \frac{\sum_i^m E_i}{m}$
3. Calculate the Euclidean distance of each record V_i with the mean vector $\bar{E}$ of examples and identify the questionable record V^* by $\left\|V^* - \bar{E}\right\| = max_{V_i}\{\left\|V_i - \bar{E}\right\|\}$

After identifying the questionable record, we will recommend it for the user to examine. As our program can contain conditional expressions, our approach will partition the records based on the conditions. We then identify the candidate questionable record for each partition and recommend the record with maximal distance to its partition's mean vector.

EVALUATION

In this section, we present the results of two experiments. One is a real user study and the other is a simulated evaluation. [1]

User Study

To test whether our approach reduces user effort and increases the likelihood that a user can correctly transforms a real dataset, we conducted a comparative user study between the system with and without extensions in real world transformation scenarios. The system without extensions hid the suggested records and color coding from users.

Dataset

The data came from 5 museums with thousands of records of artworks and artists represented in different formats. The goal was to extract the common properties across the 5 museums and convert them into the same format.

We identified 41 scenarios and grouped them based on the 8 common properties. We then randomly chose 1 from each group and used these 8 scenarios for the user study. We kept a maximum of 500 records and the average number of records was 300. We give the first several examples for each scenario to demonstrate the transformations in Table 3.

Participants and Method

We recruited 10 graduate students and randomly divided them into 2 equal groups. Group 1 used the system without extensions and Group 2 used the same system but with extensions. We first trained participants using a simple scenario and then asked them to work on the 8 scenarios by orally describing the task. We asked Group 2 participants to use the recommendation if it were applicable. If the user could not finish the scenario in 5 minutes, that scenario is regarded as a failure.

[1] Data is available at **www.isi.edu/integration/data/IUI2014**. The code is available as the data transformation tool of Karma(**http://www.isi.edu/integration/karma/**).

Table 4: User Study Results

Scen	**Without Extensions** (group 1)		**With Extensions** (group 2)	
	Avg time (sec.)	Success rate	Avg time (sec.)	Success rate
s1	30	1.0	35	1.0
s2	119	0.6	41	1.0
s3	110	0.6	40	1.0
s4	unsolved	0.0	unsolved	0.0
s5	201	0.2	95	1.0
s6	unsolved	0.0	142	1.0
s7	unsolved	0.0	unsolved	0.0
s8	191	0.4	95	1.0
All	130.2	0.35	74.6	0.75

Results

We used the following metrics to measure how each group performed on each scenario.

1. Average Time: the average time in seconds used by the users who transformed all the records in the scenario correctly.
2. Success Rate: the ratio of the users in the group that transformed all the records in the scenario correctly.

According to Table 4, both systems failed on s4 and s7, where the success rate was 0 and average time was "unsolved" as no user transformed the two scenarios correctly. We can see that group 2 used less time and achieved a higher rate on all the other scenarios except s1. Users saved 55 seconds on average of all successfully transformed scenarios and increased their success rate for 0.4 on average across all scenarios.

The time saving shown in the user study was largely because the user in group 2 did not need to examine all records to identify the ones with bad results. The more examples a scenario required, the more rounds it took the user to examine the results. Taking scenario 1 as example, because the user only needed to provide one example to derive the correct program and both groups examined the results before submitting, the group 1 users did not spend more time examining the results. The two groups had very close average times as seen in Table 4. On the other hand, for the rest of scenarios (s2, s3, s5 and s8), an average of 8.2 examples were required for group 1 users to correctly synthesize the program, while the users in group 2 only used an average of 6.5. The users in group 2 not only spent less time examining the results in each round, but also used fewer rounds. Therefore, these scenarios showed significant time savings.

The user study also showed the group 2 users achieved higher success rate. For group 1 users, many reported it was very exhausting to examine hundreds of records. They generally needed to provide 8.2 examples to synthesize the correct program. Even after providing a new example, they still needed to recheck all the records as previously correctly transformed records may have become incorrect this time. Many users simply did not check whether all the results are correct. They

Table 3: Scenarios used in the user study

Scenario	Orignal Value	Target Value
s1:extract the artist birth date	1860-1945 1870-1955	1860 1870
s2:extract the artist death date	active c. 1859 - 1910 born 1936	1910 none
s3:extract the first degree of the art work dimension	15 3/8 in. 20 x 24 1/4 in.	15 3/8 20
s4: extract the width of the artwork dimension	W: 26 in, H: 36 in. H: 28 in, W: 50 in H: 5 1/2 in, W: 8 1/2 in	26 50 8 1/2
s5: extract the third degree of the artwork dimension	11.5 in WIDE x 1.5 in DEEP(29.21 cm WIDE x 3.81 cm DEEP) 24 in HIGH(60.96 cm HIGH)	1.5 none
s6: extract the content in the parenthesizes	Despair Untitled (Grindelia) California Landscape ((Hills around Sonoma))	none Grindelia Hills around Sonoma
s7: extract the date that the artwork was made	California,1970 Los Angeles, Calif.,July 7, 1970 Los Angeles, Calif., Los Angeles, Calif.,	1970 July 7, 1970 none
s8: change the format to put surname at the end	Wyeth, Andrew Newell C. C. Bohm	Andrew Newell Wyeth C. C. Bohm

randomly chose several pages and browsed the results. They submitted the results when they found all the records in those pages were correct. However, they missed records with bad results in the pages that they did not check.

The reason that both groups failed on s4 and s7 is because the two scenarios were beyond the ability of Gulwani's approach. For s4, the approach cannot learn the end position expression for extracting the number after width. The right context for the end position is " in.", while the number after the "H" also has same context. The left context for the end position cannot be learned either. The first row's left context is ("W" ":" BNK NUM). The third row's left context is ("W" ":" NUM BNK NUM "/" NUM). After generalization over these two contexts, we get (NUM), which also has a false match position in the second row. As both the left and right context of the end position expression cannot locate the correct position, a transformation program cannot be learned in this case. For scenario 7, we want to extract the date or the year from the string. The start position is indiscernible here. The target substring may appear after the third comma, second comma or first comma. The substring itself may also start with a word or a digit too. As s4 and s7 are beyond the synthesizing programs capability, both our system and the baseline system cannot derive a correct program.

Finally, the system with recommendations succeeded in all the scenarios that can be solved by using more than one example with significantly less time and higher success rate compared to the system without recommendation.

Simulation Experiment

Our simulation tests whether our approach uses fewer examples and examines fewer records to transform all the records correctly, when compared to three other simulated record selection strategies.

Dataset

The dataset consists of 20 scenarios that we collected from Google user forum and the 6 solvable scenarios from the museum dataset. We went through all the posts from July 2012 to July 2013 and randomly collected 20 scenarios. As some posts may not post the data directly, we created the data based on the description in the post. The number of records for each scenario is 75.

Method

We designed three alternative record examining strategies to approximate user record examining behaviors and compare them with our recommendation-based strategy. Each strategy examines the results in the order described below till it identifies a record with an incorrect result.

1. Longest record: examine the record with the longest result first.
2. Shortest record: examine the record with the shortest result first.
3. Top down: examine the record from top down order.

Results

In this experiment, we measured the average number of examples and examined records on 2 datasets (Tables 5).

The "Example" shows the average number of examples for each strategy to solve each dataset. The percentage in the parenthesizes shows the percentage of examples that can be saved by using recommendation. The "Record" shows the average number of examined records. The percentage in the parenthesizes presents the percentage of examined records that can be saved by using recommendation. On average, our strategy needed 3 examples and examined 3 records on forum dataset; it needed 6.5 examples and examined 17 records on museum dataset. Our recommendation cannot alway identify the incorrect result. We noticed that for our strategy examined more records than the number of examples in scenario 6 of museum dataset. Our strategy used the first row as the first examples. The synthesized program transformed all the records to "none". Our approach cannot recommend the right record then, as our system did not know the transformation was to extract the content within parenthesizes before getting such examples.

In Tables 5, by using recommendation, the system used fewer examples and examined a lower percentage of records than

Table 5: Comparing Recommendation with Other Strategies

Datasets		Longest	Shortest	Top_Down	Recmd
Forum	Example	4.1 (26%)	4.3 (30%)	4.2 (29%)	3
	Record	48 (94%)	21 (86%)	18 (83%)	3
Museum	Example	8.3 (21%)	8.8 (26%)	9.2 (29%)	6.5
	Record	186 (91%)	162 (90%)	154 (89%)	17

the three alternative strategies. One tail t test suggested the improvements were statistically significant ($p < 0.05$).

RELATED WORK

In this section, we review the most related data transformation systems and active learning approaches. OpenRefine [4] and Potter's wheel [8] allow the user to specify edit operations. OpenRefine [4] is a tool for cleaning messy data. Its language supports regular expression style of string transformation and data layout transformation. Potter's Wheel [8] defines a set of transformation operations and let users gradually build transformations by adding or undoing transformations in an interactive GUI. Many PBD [2] approaches can learn edit operations by asking the user to demonstrate the operations. Lau's system [6] and Data Wrangler [5] learn from the user's edit operations. Lau [6] described a system that can learn from a user's edit operations and generate a sequence of text editing programs using the version space algebra. Data Wrangler [5] is an interactive tool for creating data transformation. It uses the transformation operations defined in Potter's wheel [8]. Besides supporting string level transformation, it also supports data layout transformation including column split, column merge, fold and unfold. Our approach is different from these two types of systems as it only requires users to enter the target data.

Gulwani [3] developed an approach to synthesis a program through input and output string pairs. Gulwani mentioned that his approach can highlight the entry, which has two or more alternative transformed results. This method needs to generate multiple programs and evaluate these programs on all the records, which requires more processing time. Our approach improves Gulwani's work by providing recommendation. To generate the recommendation, we only need one program and its results on the records. Topes [9] let the user specify the data pattern or learn the pattern from examples. Programmers can implement transformation functions between patterns to perform data transformation across different formats.

CueFlik [1] shows users an overview of the learned concept. Users can examine this overview to provide new examples. This overview is essentially a high level abstraction of the instances in the image feature space. Our approach recommends records from two spaces: program space and text feature space. LAPIS [7] highlight the texts that have potentially incorrect matches. Their approach identifies the matches that are different from the majority of matches. Besides helping the user identify problematic inputs, our approach also identifies the most informative record. Wolfman [10] extends Lau's [6] work by reducing the user effort using a mix initiative approach combining several interaction modes. Our work is inspired by his work of shifting the user's attention to a particular example. His approach suggests those examples that can reduce the ambiguity of the version space. However, our approach focus on acquiring the unseen examples. As our program can be a disjunct of multiple transformations, the new example does not necessarily reduce the ambiguity of version space.

CONCLUSION AND FUTURE WORK

This paper presents a general data transformation tool aiming to minimize user effort in synthesizing transformation programs by example. This tool recommends records to help the user avoid examining a large quantity of transformed results. In our simulated experiment and user study, the experimental results show the tool saves user time, reduces number of examples and increases success rates significantly.

We identified one interesting problem for the future work. When the users were working on the two unsolvable cases, they complained that adding new examples may correct some incorrect result but also make some previous correct results incorrect. It was very frustrating for the users. They did not know whether they were making progress in transforming the dataset toward the target format in general. However, the system can show the status of the current results so that the user can quit when most of the records are transformed into a good shape. It may be easier for user to process these semi-finished data than handle the raw data directly.

REFERENCES

1. Amershi, S., Fogarty, J., Kapoor, A., and Tan, D. Overview based example selection in end user interactive concept learning. In *UIST* (2009), 247–256.
2. Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., and Turransky, A., Eds. *Watch what I do: programming by demonstration*. MIT Press, 1993.
3. Gulwani, S. Automating string processing in spreadsheets using input-output examples. In *POPL* (2011), 317–330.
4. Huynh, D. F., and Stefano, M. *OpenRefine http://openrefine.org*.
5. Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. Wrangler: interactive visual specification of data transformation scripts. In *CHI* (2011), 3363–3372.
6. Lau, T., Wolfman, S. A., Domingos, P., and Weld, D. S. Programming by demonstration using version space algebra. *Mach. Learn.* (2003), 111–156.
7. Miller, R. C., and Myers, B. A. Outlier finding: Focusing user attention on possible errors. In *UIST* (2001), 81–90.
8. Raman, V., and Hellerstein, J. M. Potter's wheel: An interactive data cleaning system. In *VLDB* (2001).
9. Scaffidi, C., Myers, B., and Shaw, M. Topes: Reusable abstractions for validating data. In *ICSE* (2008), 1–10.
10. Wolfman, S. A., Lau, T. A., Domingos, P., and Weld, D. S. Mixed initiative interfaces for learning tasks: Smartedit talks back. In *IUI* (2001), 167–174.