# Poster: Synthesizing Relation-Aware Entity Transformation by Examples

Jiarong Wu
sissel126@gmail.com
Nanjing University

Yanyan Jiang*
jyy@nju.edu.cn
Nanjing University

Chang Xu
changxu@nju.edu.cn
Nanjing University

Shing-Chi Cheung
scc@cse.ust.hk
Hong Kong University of Science and Technology

Xiaoxing Ma
xxm@nju.edu.cn
Nanjing University

Jian Lu
lj@nju.edu.cn
Nanjing University

## ABSTRACT

Recently, programming by examples (PBE) technique achieves a great success in processing and transforming data entities, yet existing approaches generally fall short on the tasks concerning entity relations. This paper presents ENTER, a domain-agnostic language for relation-aware entity transformation synthesis. It leverages the combination of two basic relations, the equivalence relation and the total order relation, to succinctly express complex entity relations. ENTER can be instantiated with domain-specific elements to solve a wide range of entity transformation tasks.

## CCS CONCEPTS

• **Software and its engineering** → **Programming by example**;

## KEYWORDS

Program Synthesis, Programming by Examples, Data Transformation

**ACM Reference Format:**
Jiarong Wu, Yanyan Jiang, Chang Xu, Shing-Chi Cheung, Xiaoxing Ma, and Jian Lu. 2018. Poster: Synthesizing Relation-Aware Entity Transformation by Examples. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194963

## 1 INTRODUCTION

Today, billions of people have access to computational devices, but many of them have no programming background and often struggle with repetitive tasks that could have been automated by small scripts [3]. *Programming by Examples* (PBE) helps automatically generate such scripts. Given a set of representative examples (input-output pairs), a PBE technique synthesizes a program that: (1) for any input in the examples, produces its corresponding output, and (2) generalizes well to new inputs of the same task.

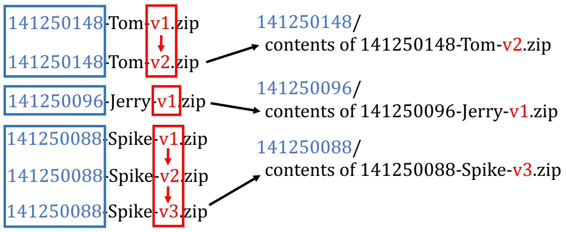

**Figure 1: A typical PBE task of archiving student submissions.**

```
partition (splitName('-',0));
chain (modificationTime,timeComparator);
if (isHead)
  then { unzip(splitName('-', 0)); }
  else { delete; }
```

**Figure 2: Synthesized program for Figure 1.**

PBE has been successfully applied in application domains such as spreadsheets [2, 5], text extraction [4], and file management [6]. A typical PBE task is shown in Figure 1, where a teacher archives students' homework in a shared folder. A student may submit multiple times, each with a distinct file named in the format of `id-name-version.zip`. Now the task is to decompress only each student's latest submission and delete the old ones. After observing some manual operations (*i.e.*, creating folders and decompressing selected submissions), a PBE tool would generate a program that can process the rest of the files (see Figure 2).

In this paper, we particularly focus on the *entity transformation* tasks in which entities are processed and transformed. Here an *entity* refers to an atomic data record in a specific domain (*e.g.*, a cell in a spreadsheet). Each entity should be transformed by a series of operations, depending on the attributes of the entity and its relations with other entities. The example in Figure 1 is a typical entity transformation task. Although existing research has successfully addressed a wide range of entity transformation tasks, the discovery and exploitation of latent relations between entities remain a challenge. [1] The domain-specific languages (DSL) adopted by prior

*Corresponding author

[1] For example, "`141250148-Tom-v2.zip`" in Figure 1 is unarchived because there is an old version file "`141250148-Tom-v1.zip`" submitted by Tom.

*ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden*

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194963

$$
\begin{array}{rrll}
\text{Program} & \text{P} & := & \text{S} \\
\text{Statement Sequence} & \text{S} & := & \text{partition}\ (f)\ ;\ \text{S} \\
 & & \mid & \text{chain}\ (f, cp)\ ;\ \text{TS} \\
 & & \mid & \text{if (ep) then} \{ \text{S} \} \text{ else } \{ \text{S} \} \\
 & & \mid & st\ ; \\
\text{Chained Sequence} & \text{TS} & := & \text{if (tp) then} \{ \text{S} \} \text{ else } \{ \text{S} \} \\
 & & \mid & tt\ ; \\
\text{Equivalence Predicate} & \text{ep} & := & f = c \\
\text{Comparison Predicate} & \text{tp} & := & f <_{cp} c \ \mid\ f \leq_{cp} c \\
 & & \mid & \text{isHead} \ \mid\ \text{isTail}
\end{array}
$$

**Figure 3: Syntax of ENTER. $f$ refers to a feature, $cp$ refers to a comparator, $c$ denotes a constant value, while $st$ is a transformation over unordered entities, and $tt$ is a transformation over sorted entities.**

work typically assumed that entities are independent and would fall short on the real-world cases where relations matter.

To better discover and exploit entity *relations*, we propose a novel language ENTER (ENtity TransformER) to capture the common relations in PBE tasks. Its design is based on our observation that two basic relations, the *equivalence relation* and the *total order relation*, can be combined and nested to express sophisticated relations between entities. For example, to identify the files that should be unarchived in Figure 1, the files are first partitioned according to the prefixes of the file names (an equivalence relation), and then the files in each subset are sorted according to their modification time (a total-order relation). As such, the target files can be easily interpreted as the frontest ones in each subset.

Moreover, the design of ENTER is domain-agnostic, enabling us to implement it as a PBE framework and instantiate PBE tools for different data transformation domains, *e.g.*, file management and spreadsheet reorganization.

## 2 LANGUAGE AND SYNTHESIS

The syntax of ENTER is defined in Figure 3. Note that ENTER is a domain-agnostic framework and the domain-specific elements, *i.e.*, features $f$, comparators $cp$, operations $st$ and $tt$ are provided by developers implementing PBE tools with ENTER.

Language elements are organized in an imperative style. A program P can be viewed as a loop-free statement sequence (S/TS) with branches (the `if` statement). The program runs on a state $\sigma = (E, S, T)$, where $E$ is a set of entities (an entity can be viewed as a tuple of certain attributes), $S$ is a mathematical partition of $E$ and $T$ is a set of chains (*i.e.*, totally ordered sets) on the sets in $S$.

A *feature* is a function that extracts an attribute from an entity, *e.g.*, name of a file. A *comparator* is a function that decides the total order over an feature, *e.g.*, the `timeComparator` in Figure 2 sort files from newer to older. A *transformation* is a function in the form $\lambda\sigma \to \sigma$, which means it can calculate the output state based on both entity set ($E$) and relations ($S$ and $T$) in the input state.

The `partition` operator further partitions the $S$ component in current state, according to the features of entities extracted by the parameter $f$, while the `chain` operator generates a new $T$ component from current state. The `if` operator divides the current component into two independent states, by deciding whether the entities satisfy the parameter predicate. Later, the two states will continue to execute separately.

The synthesis of ENTER programs from user-specified examples follows the syntax-guided enumeration [1]. More concretely, we perform a breadth-first search by deriving new states from visited states. Each round, a state in the queue is popped and checked. If all examples can be correctly transformed, we found a candidate program. Otherwise, we enumerate the operators (*e.g.*, partition, chain) and language elements (*e.g.*, features) to fill the “holes” in an intermediate state, and push the newly derived states into the queue. After a predefined number of rounds, we cease the search and backtrack the marked states to obtain the programs.

## 3 APPLICATIONS

Instantiated with domain-specific knowledge, ENTER can be used to perform various categories of PBE tasks. The key is to define the feature set (to guide *partition* and *chain*) and the transformation set (to perform the final transformation).

The feature set should reflect the concerns of common user tasks. For example, when we are organizing files, we may pay attention to the file names, path messages, file extensions, *etc.* We may also pick more complex ones. In Figure 2, the program splits the file names with a common separator “-” and extracts the first component.

The transformation set should contain the typical operations in the domain. In file management, they are copy, move, rename, archive, *etc.* ENTER also supports specializing a transformation with the previous extracted features. For example, in Figure 2, the student id is used as the directory name in the unarchive operation.

## 4 CONCLUSION & FUTURE WORK

In this paper, we propose a framework, ENTER, for synthesizing relation-aware entity transformations from examples. The central idea is to decompose sophisticated relations into combinations of the equivalence relation and the total order relation. We also illustrate the language design and how it models domain-knowledges.

Our future work will focus on instantiating and evaluating the framework for more application domains. Also, we expect to accelerate the synthesis process, which now relies on the syntax-guided enumeration.

## ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation (Grants #61690204, #61472174) and National Key R&D Program (Grant #2017YFB1001801) of China.

## REFERENCES

[1] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M K Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. *Formal Methods in Computer-Aided Design (FMCAD)* (2013), 1–8.

[2] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *POPL* 46, 1 (2011), 317–330.

[3] Sumit Gulwani. 2016. Programming by Examples (and its Applications in Data Wrangling). In *Verification and Synthesis of Correct and Secure Systems.* IOS Press.

[4] Vu Le and Sumit Gulwani. 2014. FlashExtract: A Framework for Data Extraction by Examples. *PLDI* 49, 6 (2014), 542–553.

[5] R Singh and S Gulwani. 2016. Transforming Spreadsheet Data Types using Examples. *Acm Sigplan Notices* 51, ii (2016), 343–356.

[6] Navid Yaghmazadeh and Christian Klinger. 2016. Synthesizing Transformations on Hierarchically Structured Data. *PLDI* (2016), 508–521.