TRINITY: An Extensible Synthesis Framework for Data Science

Ruben Martins
Carnegie Mellon University
rubenm@andrew.cmu.edu

Jia Chen
UT Austin
jchen@cs.utexas.edu

Yanju Chen
UCSB
yanju@cs.ucsb.edu

Yu Feng
UCSB
yufeng@cs.ucsb.edu

Isil Dillig
UT Austin
isil@cs.utexas.edu

Demo 1: Data Wrangling in R

Given only this simple input-output example, TRINITY can automatically synthesize the corresponding R program using the `tidyr` library:

Table 1: Input table

Student	Grade	Score1	Score2
Greg	A	75	76
Greg	B	86	85
Sally	A	85	86
Sally	B	80	78

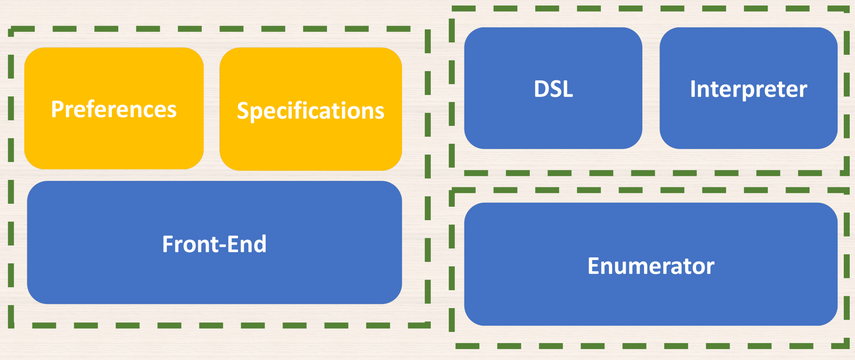
The user can optionally provide preference predicates, such as occurs(`gather`, 60%), which would bias the search in synthesizer.

Table 2: Output table

Student	B_Score1	B_Score2	A_Score1	A_Score2
Greg	86	85	75	76
Sally	80	78	85	86

```
df1=gather(input,Score,Value,Score1,Score2)
df2=unite(df1,AllScores,Grade,Score)
output=spread(df2,AllScores,Value)
```

TRINITY: Usage Overview



End Users

Automate various PBE tasks by:
- ❖ specifying your input-output examples
- ❖ specifying preference predicates

Advanced Users

Build and extend synthesizers by:
- ❖ providing a domain-specific language (DSL)
- ❖ adding logical specifications to prune search space

Synthesis Expert

Customize the synthesis engine by:
- ❖ modifying backtracking search over DSL
- ❖ changing the search strategy
- ❖ integrating statistical models

https://fredfeng.github.io/Trinity/
Documentation & Source Code

Features

Usability

TRINITY can be easily adapted to new application domains by providing a suitable DSL (and its semantics) for the target application scenarios.

Customizability

TRINITY gives users fine-grained control over inductive bias by providing preference predicates that constrain the space of synthesized programs.

Efficient Synthesis

TRINITY is based on an efficient synthesis algorithm that combines search and lightweight deduction.

Demo 2: Extending the DSL for Data Wrangling

TRINITY is a modular synthesizer that allows the user to extend existing synthesizers or create new ones. The user can extend the synthesizer by adding this new `summarise` function to the existing DSL.

```
TABLE → input | select(TABLE, LIST)
        unite(TABLE, COL, COL) | separate(TABLE, COL)
        spread(TABLE, COL, COL) | gather(TABLE, LIST)
        filter(TABLE, COL, OP, C)
 LIST → [1] | [1,2] | ...| [4,5]
  COL → 1 | ...| 5
   OP → < | = | >
    C → 0 | ...| 10
```

Figure 2: Part of the DSL for data wrangling tasks in R

```
1  class MorpheusInterpreter(Interpreter):
2  def eval_summarise(self, node, args):
3    ...
4    _script =
5    'ret_df} <-
6     {table} %>% summarise({TMP} =
7     {aggr} (.[[{col}]]))'
8    .format(ret_df=ret_df_name,
9            table=args[0],
10           TMP=get_fresh_col(),
11           aggr=str(args[1]),
12           col=str(args[2]))
13   return robjects.r(_script)
```

adding interpreter wrapper for `summarise`

```
TABLE → summarise(TABLE, AGG, COL)
  AGG → min | max | mean | sum
```

adding new function `summarise` to DSL

```
1  func summarise: Table r -> Table a,
2                  Aggr b, ColInt c {
3    row(r) < row(a);
4    col(r) <= col(a) + 1;
5  }
```

adding coarse specification for `summarise`

Demo 3: Modifying the Search Engine

Expert users can also customize the underlying search engine of TRINITY to further speed up their synthesizer. In particular, users can provide statistical models (e.g., deep neural network, n-gram) that can be used to predict the most likely programs.