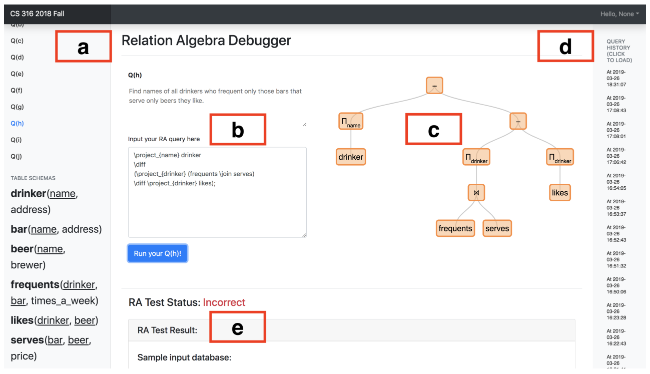

CS 316 2018 Fall

Hello, None

Q(b)
Q(c)
Q(d)
Q(e)
Q(f)
Q(g)
Q(h)
Q(i)
Q(j)

TABLE SCHEMAS

drinker(name, address)

bar(name, address)

beer(name, brewer)

frequents(drinker, bar, times_a_week)

likes(drinker, beer)

serves(bar, beer, price)

Relation Algebra Debugger

Q(h)

Find names of all drinkers who frequent only those bars that serve only beers they like.

Input your RA query here

```
\project_{name} drinker
\diff
(\project_{drinker} (frequents \join serves)
\diff \project_{drinker} likes);
```

Run your Q(h)!

RA Test Status: Incorrect

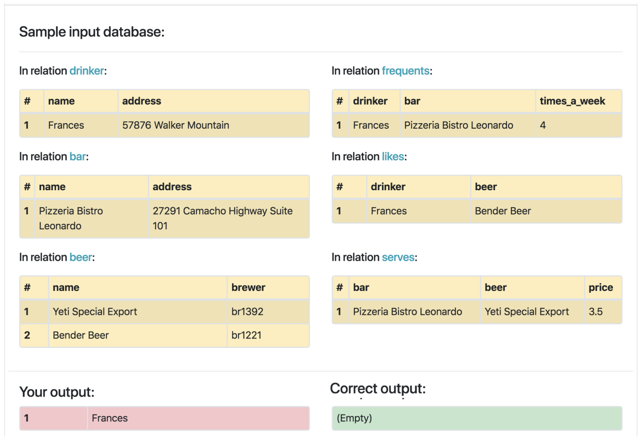
RA Test Result:

Sample input database:

QUERY HISTORY (CLICK TO LOAD)

At 2019-03-26 18:31:07
At 2019-03-26 17:08:43
At 2019-03-26 17:08:01
At 2019-03-26 17:06:42
At 2019-03-26 16:54:05
At 2019-03-26 16:53:37
At 2019-03-26 16:52:43
At 2019-03-26 16:51:32
At 2019-03-26 16:50:06
At 2019-03-26 16:23:28
At 2019-03-26 16:22:43
At 2019-03-26

Figure 2: Overview of RATEST interface.


Sample input database:

In relation drinker:

#	name	address
1	Frances	57876 Walker Mountain

In relation frequents:

#	drinker	bar	times_a_week
1	Frances	Pizzeria Bistro Leonardo	4

In relation bar:

#	name	address
1	Pizzeria Bistro Leonardo	27291 Camacho Highway Suite 101

In relation likes:

#	drinker	beer
1	Frances	Bender Beer

In relation beer:

#	name	brewer
1	Yeti Special Export	br1392
2	Bender Beer	br1221

In relation serves:

#	bar	beer	price
1	Pizzeria Bistro Leonardo	Yeti Special Export	3.5

Your output:

1	Frances

Correct output:

(Empty)

Figure 3: Counterexample for the query in Figure 2.

pick the overall smallest counterexample. However, computing provenance on all result tuples incurs high overhead. Instead, we can pick one tuple t in the difference, and compute provenance just for t, by adding a selection using t's value on top of the difference query. Pushing this selection down as much as possible through the difference query drastically reduces the cost of provenance computation. Although this approach may not give us the smallest counterexample, it can always find a counterexample, and in practice it works well: it can reduce the running time significantly (up to 42× in our experiments) while still finding counterexamples that are almost as small as the smallest possible.

Note that since the counterexample contains a subset of D's tuples, it trivially satisfies all keys, functional dependencies, and `NOT NULL` constraints. However, referential constraints need explicit enforcement. We capture such constraints using Boolean formulas (involving Boolean variables corresponding to D's tuples), and additionally assert these constraints when invoking the solver.

4 DEMONSTRATION

Figure 2 shows the web interface of RATEST. The demonstration goes through a use case of RATEST in an educational setting, where students are asked to write relational algebra queries to answer questions against a database about bars, beers, drinkers, and their relationships (bars *serve* beers; drinkers *like* beers and *frequent* bars); see bottom left of Figure 2. Each table contains hundreds to thousands of tuples, and the whole database contains 100,000 tuples.

As an example, consider one of the hardest problems in the assignment: "Find all drinkers who frequent only those bars that serve only beers they like." The solution requires non-trivial uses of joins and differences:

$$\pi_{name} drinker - \pi_{drinker}(\pi_{drinker, beer}(frequents \bowtie serves) - likes).$$

Now consider a student query

$$\pi_{name} drinker \; - \; (\pi_{drinker}(frequents \bowtie serves) - \pi_{drinker} likes),$$

where the second input to the first difference operator is incorrect—this input would find "drinkers who frequent some bars and like no beers." RATEST shows a small counterexample explaining this mistake (Figure 3): drinker "Frances" likes "Bender Beer" and frequents "Pizzeria Bistro Leonardo," where only another beer "Yeti Special Export" is served. Thus, "Frances" should not be in the answer.

The user can revise the query and get further feedback that may reveal more bugs; a history feature allows convenient access to queries attempted recently. We also walk through the backend of RATEST for those interested.

Overall, this demonstration shows how RATEST uses small counterexamples to help users understand why their queries are wrong. We are also releasing RATEST to the public to encourage adoption by database courses at other universities.

ACKNOWLEDGMENTS

This work is supported in part by NSF Awards IIS-1408846, IIS-1552538, IIS-1703431, IIS-1718398, IIS-1814493, and by NIH award 1R01EB025021-01.

REFERENCES

[1] Yael Amsterdamer, Daniel Deutch, and Val Tannen. 2011. Provenance for aggregate queries. In *PODS*. 153–164.
[2] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. 2010. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*, Vol. 13. 14.
[3] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *ICDT*. 316–330.
[4] Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. 2017. Cosette: An Automated Prover for SQL. In *CIDR*.
[5] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. 337–340.
[6] Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40.
[7] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining Wrong Queries Using Small Examples. In *SIGMOD*.