Running Experiments and Performing Data Analysis Using SchemaAnalyst and DOMINO

Abdullah Alsharif
University of Sheffield

Gregory M. Kapfhammer
Allegheny College

Phil McMinn
University of Sheffield

I. INTRODUCTION

SchemaAnalyst is a tool, developed in the Java programming language, that automatically generates tests for complex, real-world relational database schemas. It features several data generators including DOMINO (DOMain-specific approach to INtegrity cOnstraint test data generation), Alternating Variable Method (AVM), and Random$^+$ [1], [2], [4]. *SchemaAnalyst* generates tests that support three database management systems (DBMSs): PostgreSQL, SQLite, and HyperSQL. It also provides a mutation testing tool to mutate (i.e., remove, add, or flip) the integrity constraints in the schema under test.

It is important to test a relational database schema because small mistakes, such as omitting the definition of a schema's integrity constraint (i.e., a `PRIMARY KEY` or `UNIQUE`) can compromise application correctness and increase maintenance costs. For example, forgetting to mark each `username` as `UNIQUE` will lead to the duplication of user names within a relational database. To counter this issue, *SchemaAnalyst* generates, for a specific schema, a test suite that systematically satisfies integrity constraint coverage criteria [2], requiring each generated test case to exercise the relational schema's integrity constraints as true or false (i.e., accepted or rejected by the DBMS).

This paper explains how to run test generation experiments and data analysis with *SchemaAnalyst* and its data analysis package written in the R language for statistical computation. It will help others to use *SchemaAnalyst*, replicate prior experiments (e.g., [4]), and conduct new studies of schema testing.

II. RUNNING EXPERIMENTS WITH *SchemaAnalyst*

Since it is implemented in the Java language, *SchemaAnalyst* is cross platform. It uses the Gradle tool to manage its building, testing, and dependencies. Testers can follow the instructions at the tool's GitHub repository[1] and a previous tool paper [3] to learn how to install and run *SchemaAnalyst*. As these resources do not show how to experimentally evaluate *SchemaAnalyst*, this paper explains how to run experiments using a provided Python script called `runExperiments.py`. Because *SchemaAnalyst*'s search-based test generation methods are stochastic, testers can parameterize this script with the number of trials and a random seed in addition to giving the name of a test data generator, DBMS, and the schema under test. These are the steps for configuring and running the experiments:

1) Install *SchemaAnalyst* and one or more DBMSs.
2) Edit the `config/database.properties` file so that it provides the access details for each of the DBMSs.
3) Run the Gradle compile command, `./gradlew compile`, to install all of *SchemaAnalyst*'s dependencies.
4) Set the `CLASSPATH` to point to the tool's build directory.
5) Modify `scripts/runExperiments.py` to configure the experiment (e.g., specify the number of trials).

An experimenter now runs the Python script, performing mutation analysis on tests generated by *SchemaAnalyst*, thereby generating the results files. Located in the `results/` directory, these files include: (1) `mutationanalysis.dat` with basic test generation and mutation information for each run; (2) `mutanttiming.dat` with details for each schema mutant both killed and alive; (3) `alive_mutant/` a directory with files and directories furnishing details about each run of data generation and mutation analysis, with notes about every live mutant.

III. ANALYZING DATA FROM STUDIES OF *SchemaAnalyst*

We created an R package[2] to replicate our paper's data and tables [4]. Researchers can use `devtools` [5] to download and install the replication package and then take these steps:

1) Load the empirical results from prior experiments with:
```
mutants <- dominoR::read_analysis()
analysis <- dominoR::read_mutants()
```
2) To re-generate the tables in our main paper [4], a researcher can run the R package's functions (e.g., `dominoR::table_generator_coverage`), following the provided instructions for details about inputs and outputs.
3) While the default format of the result tables is like that of our main paper, researchers can modify the replication package's code to customize table output as needed.
4) To support the generation of tables with different entries, the results analysis functions can be parameterized to, for instance, compute either mean or median values.

To conclude, this paper explained both how to easily run experiments with *SchemaAnalyst* and to perform data analysis with an R package. This supports the reproduction of prior experimental results and guides future researchers who want to conduct their own analyses of schema testing methods. We invite practitioners and researchers to use the test generators and mutation analysis methods provided by *SchemaAnalyst*.

REFERENCES

[1] G. M. Kapfhammer, P. McMinn, and C. J. Wright, "Search-based testing of relational schema integrity constraints across multiple database management systems," in *Proc. of ICST*, 2013.
[2] P. McMinn, C. J. Wright, and G. M. Kapfhammer, "The effectiveness of test coverage criteria for relational database schema integrity constraints," *TOSEM*, vol. 25, no. 1, 2015.
[3] P. McMinn, C. J. Wright, C. Kinneer, C. J. McCurdy, M. Camara, and G. M. Kapfhammer, "SchemaAnalyst: Search-based test data generation for relational database schemas," in *Proc. of ICMSE*, 2016.
[4] A. Alsharif, G. M. Kapfhammer, and P. McMinn, "DOMINO: Fast and effective test data generation for relational database schemas," in *Proc. of ICST*, 2018.
[5] "Devtools" https://github.com/hadley/devtools.

[1] https://github.com/schemaanalyst/schemaanalyst

[2] https://github.com/schemaanalyst/domino-replicate