YERITH_QVGE-definitions---cheat--sheet 2
YERITH_QVGE-user-guide 14

User's Guide for YRI_SD_RUNTIME_VERIF : A C^{++} Functional Library for Specifying "SDMM" (State Diagram Mealy Machine)

AUTHOR: Xavier Noumbissi Noundou [Pr. Prof. Dr.–Ing.]
Contact: YERITH.XAVIER@gmail.com

Contents

List of Figures 2

List of Tables 3

1 Motivation for "YRI_SD_RUNTIME_VERIF" 6

2 FORMAL description and definition of STATE DIAGRAM mealy machines 6
2.1 HAREL–statecharts 6
2.2 SDMM (State Diagram MEALY MACHINE) 6

3 HOW TO Setup C^{++} Library "YRI_SD_RUNTIME_VERIF" for Usage in A C^{++} PROGRAM SOURCE CODE 6

4 METHODS of C^{++} Library "YRI_SD_RUNTIME_VERIF" 7

5 Formal Scientific and Engineering Project Description 9

6 Conclusion 9

List of Figures

1 A motivating example, as previous bug found in YERITH−ERP−9.0. $\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name). $\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name). . . . 4
2 YERITH−ERP−9.0 administration section displaying departments ($\neg\underline{Q0}$). . . . 4
3 YERITH−ERP−9.0 stock asset window listing some assets ($\overline{Q1}$). . . . 4
4 **A SAMPLE state diagram mealy machine file. KEYWORDS belonging both to 'engineering ("ERROR_STATE_AUTO")', and 'science (START_STATE)' can be intermingled in the same SDMM specification file.** . . . 4
5 A SCREENSHOT OF YERITH_QVGE. . . . 5
6 A SCREENSHOT OF YRI-DB-RUNTIME-VERIF SQL EVENT LOG. . . . 5
7 YRI-DB-RUNTIME-VERIF operation WORKFLOW (inspired from "Jon Eyolfson's RV'13 paper on TRACERORY" (https://hdl.handle.net/10012/6206)). . . . 5
8 A motivating example, as previous bug found in YERITH−ERP−9.0. $\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name). $\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name). . . . 7

List of Tables

1 STATE DIAGRAM MEALY MACHINE SPECIFICATION KEYWORDS IN YERITH_QVGE. **'AUTO'** KEYWORDS SPECIFIES ALSO SQL QUERY FOR GOING OUT AUTOMATICALLY FROM A FAIL (FORBIDDEN) STATE. ("SEE SECTION **??**.") 4
2 YERITH_QVGE Design and Testing System Dependencies 8

Table 1: STATE DIAGRAM MEALY MACHINE SPECIFICATION KEYWORDS IN YERITH_QVGE. **'AUTO'** KEYWORDS SPECIFIES ALSO SQL QUERY FOR GOING OUT AUTOMATICALLY FROM A FAIL (FORBIDDEN) STATE. ("SEE SECTION **??**.")

scientific keywords	engineering keywords
in_set_trace	in_sql_event_log
not_in_set_trace	not_in_sql_event_log
`recovery_sql_query`	`recovery_sql_query`
STATE	STATE
START_STATE	BEGIN_STATE
FINAL_STATE ("**FINAL_STATE_AUTO**")	END_STATE ("**END_STATE_AUTO**") / ERROR_STATE ("**ERROR_STATE_AUTO**")
IN_PRE	IN_BEFORE
IN_POST	IN_AFTER
NOT_IN_PRE	NOT_IN_BEFORE
NOT_IN_POST	NOT_IN_AFTER

Figure 1: A motivating example, as previous bug found in YERITH−ERP−9.0.
$\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name).
$\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name).

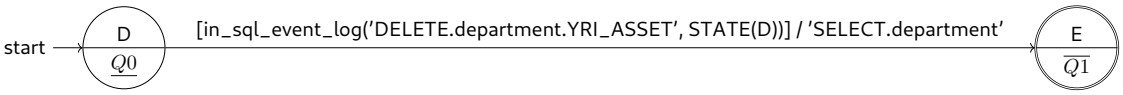


Figure 2: YERITH−ERP−9.0 administration section displaying departments (¬$\underline{Q0}$).

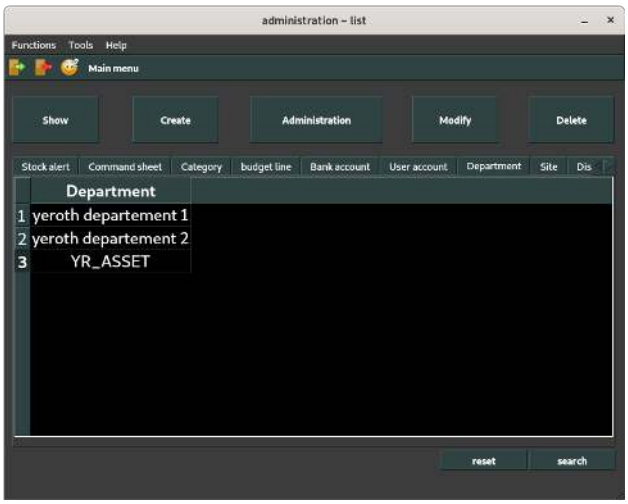


Figure 3: YERITH−ERP−9.0 stock asset window listing some assets ($\overline{Q1}$).

Figure 4: **A SAMPLE state diagram mealy machine file. KEYWORDS belonging both to 'engineering ("ERROR_STATE_AUTO")', and 'science (START_STATE)' can be intermingled in the same SDMM specification file.**

```
1. yr_sd_mealy_automaton_spec yr_missing_department_NO_DELETE
2. {
3.  START_STATE(d):NOT_IN_BEFORE(YRI_ASSET,department.department_name)
4.   ->[in_sql_event_log('DELETE.departement.YRI_ASSET',STATE(d))]/'SELECT.department'->
5.    ERROR_STATE(e):IN_AFTER(YRI_ASSET,stocks.department_name).
6. }
```

Figure 5: A SCREENSHOT OF YERITH_QVGE.

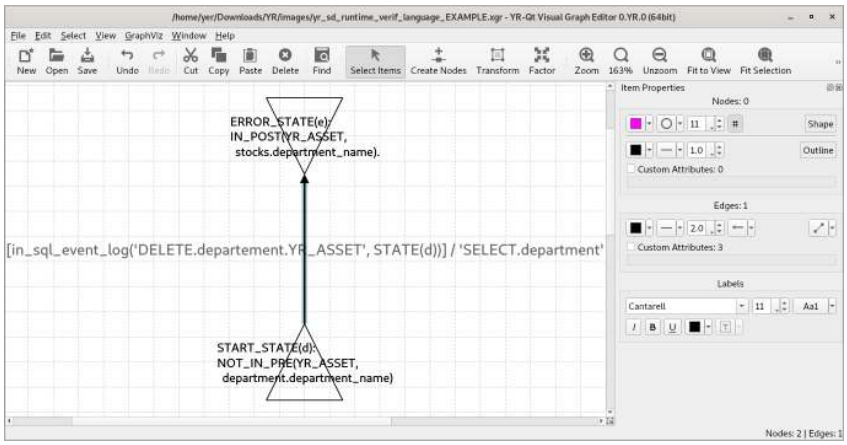


Figure 6: A SCREENSHOT OF YRI-DB-RUNTIME-VERIF SQL EVENT LOG.

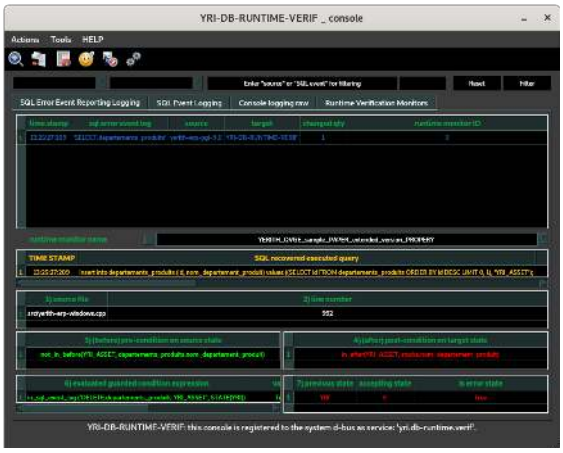
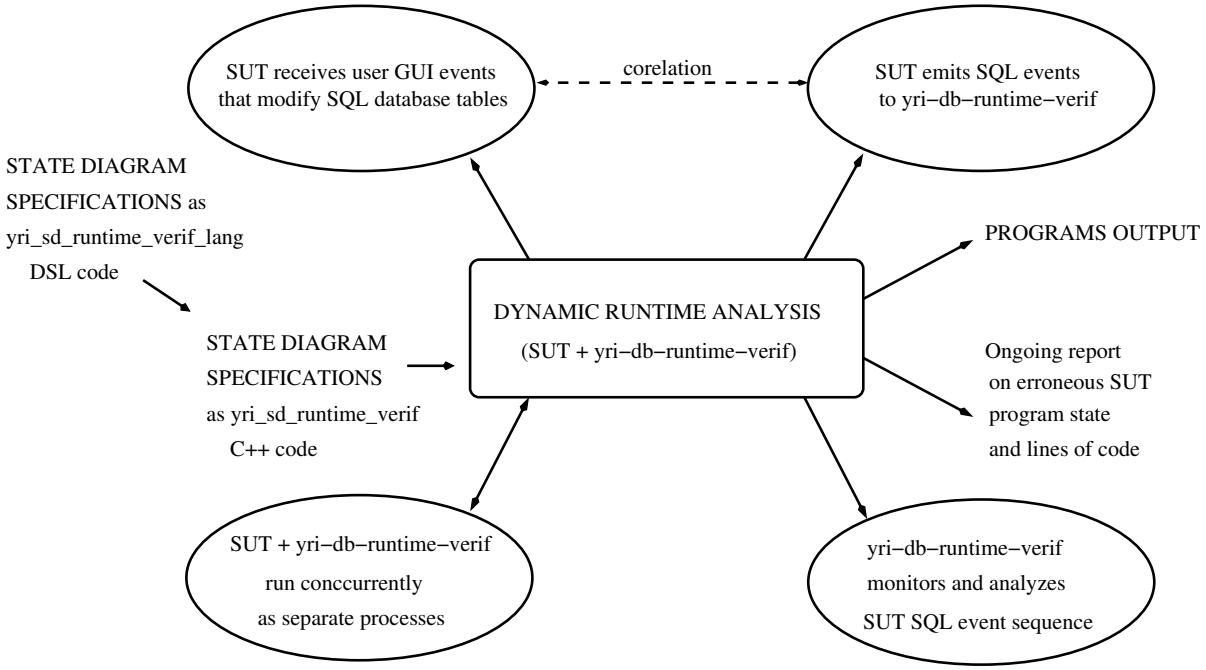


Figure 7: YRI-DB-RUNTIME-VERIF operation WORKFLOW (inspired from "Jon Eyolfson's RV'13 paper on TRACERORY" (https://hdl.handle.net/10012/6206)).



1 Motivation for "YRI_SD_RUNTIME_VERIF"

Specifying temporal usage rules for system safety is primordial for ensuring a proper quality assurance of a system that is aimed to be more productive and safer in its usage by users, and / ormaintainers.

THIS is why we created the C++ library YRI_SD_RUNTIME_VERIF. "**YRI_SD_RUNTIME_VERIF** " is an acronym for ***YERITH_State_Diangram_Runtime_Verification***.

"YRI_SD_RUNTIME_VERIF " has following usages and advantages:

1.) ***Formalism STATE DIAGRAM mealy machine*** for system specification: Mealy Machines are kind of state machines where output states depend only on input on a source state.

STATE DIAGRAM mealy machine as defined by myself is a finite state automaton that only has following characteristics:

a.) Each automaton / machine only has 1 start state S_0, and 1 final state S_f that is an accepting / error state.

An accepting or error state is a state that shows a defect status of the system under analysis (SUA / PUA / SUT)

b.) Each state S_i only has 1 outgoing state transition to an output state S_0.

c.) Each transition T, except a start transition T_0, could have a **pre-condition, and post-condition**; That both are called or named state-edge-condition (**pre-condition on T_1:** "T_1"; **post-condition on T_1:** "T_1").

2.) A C++ library that implements runtime monitoring and fail–state recovery as a static library : (https://www.github.com/yerithd/yri_sd_runtime_verif).

3.) A Free and OPEN SOURCE CODE SOFTWARE (Foss) implementation of a runtime monitor for using *state diagram mealy machine* specifications; BY means of a QT–dbus software communication stack with your own software : "**YRI-DB-RUNTIME-VERIF**" (https://www.github.com/yerithd/yri-db-runtime-verif).

YRI_SD_RUNTIME_VERIF's formal description of the state diagram formalism follows ***Mealy machine*** [Wik22] added with ***accepting states (final or erroneous states), and state diagram transition pre- and post-conditions*** : "*state diagram mealy machine*" ("SDMM").

Another excellent, detailed with proofs and theory presentation of mealy automata [PlH21] is available. In comparison to statechart [Har87], which is a ***visual formalism*** for states diagrams, YRI_SD_RUNTIME_VERIF doesn't support at time for instance the following features: *hierarchical states (composite state, submachine state), timing conditions.*

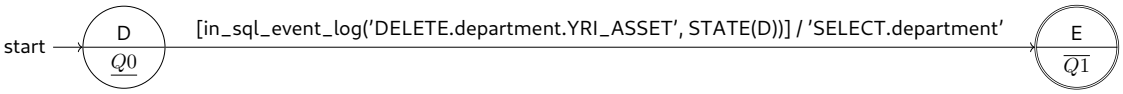
2 FORMAL description and definition of STATE DIAGRAM mealy machines

2.1 HAREL–statecharts

2.2 SDMM (State Diagram MEALY MACHINE)

3 HOW TO Setup C++ Library "YRI_SD_RUNTIME_VERIF" for Usage in A C++ PROGRAM SOURCE CODE

Figure 8: A motivating example, as previous bug found in YERITH–ERP–9.0.
$\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name).
$\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name).



4 METHODS of C++ Library "YRI_SD_RUNTIME_VERIF"

Table 2: YERITH_QVGE Design and Testing System Dependencies

PROJECT	Required Program / Library
1) YRI_SD_RUNTIME_VERIF_LANG	
2) YRI_SD_RUNTIME_VERIF_LANG_COMP	1)
3) YRI_SD_RUNTIME_VERIF_UNIT_TESTS	1)
4) YRI-DB-RUNTIME-VERIF	2)

Table 2 illustrates for each library project, which others it depends on.

5 Formal Scientific and Engineering Project Description

Detailed formal scientific and engineering contributions of design and testing system YERITH_QVGE can be found in ***JOURNAL ARTICLE "Runtime Verification Of SQL Correctness Properties with YRI-DB-RUNTIME-VERIF"*** [Nou23].

6 Conclusion

The graphical drawing tool YERITH_QVGE (Figure 5) costs only 2,500 EUROS. WE ONLY SUPPORT DEBIAN–LINUX (https://www.debian.org).

References

[Har87] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3), 1987.

[Nou23] Xavier Noundou. A Framework for Verifying SQL Correctness Temporal Properties of GUI Software at Runtime. https://zenodo.org/records/10976659, October 2023.

[PlH21] Jan Peleska and Wen ling Huang. Test automation; foundations and applications of model-based testing. https://www.informatik.uni-bremen.de/agbs/jp/papers/test-automation-huang-peleska.pdf, July 2021. Accessed last time on May 06, 2023 at 12:00.

[Wik22] Wikipedia.org. Mealy machine. https://en.wikipedia.org/wiki/Mealy_machine, December 2022. Accessed last time on Dec 15, 2022 at 12:00.

User's Guide for the Design and Testing System YERITH_QVGE (YRI_QVGE)

AUTHOR: Xavier Noumbissi Noundou [Pr. Prof. Dr.–Ing.]
Contact: YERITH.XAVIER@gmail.com

Contents

List of Figures 2

List of Tables 3

1 Introduction 6

2 YERITH_QVGE (YRI_QVGE) Short Overview 6

3 YERITH_QVGE (YRI_QVGE) Project Dependency 7

4 Advantages of YERITH_QVGE 7

5 State Diagram Mealy Machine (SDMM) 7
5.1 HOW TO READ A "SDMM" . . . 7
5.2 "SDMM" WITH MORE THAN 2 STATES . . . 7

6 YERITH_QVGE (YRI_QVGE) Workflow 7

7 Custom User Project (YRI–DB–RUNTIME–VERIF) 7

8 HOW TO START YRI-DB-RUNTIME-VERIF 8

9 SQL QUERY Recovery execution on demand 8
9.1 Automatic SQL Command Query Generation . . . 8
9.1.1 ERROR ACCEPTING STATE for sdmm 1. . . . 8
9.1.2 RECOVERY 1. . . . 9
9.1.3 RECOVERY 2 (Practical solution to be implemented in YRI-DB-RUNTIME-VERIF. . . . 9
9.1.4 Concrete RECOVERY 2 action. . . . 9

10 HOW TO USE a user interface 9

11 Formal Scientific and Engineering Project Description 9

12 Conclusion 9

List of Figures

1 A motivating example, as previous bug found in YERITH−ERP−9.0. $\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name). $\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name). . . . 4
2 YERITH−ERP−9.0 administration section displaying departments ($\neg\underline{Q0}$). . . . 4
3 YERITH−ERP−9.0 stock asset window listing some assets ($\overline{Q1}$). . . . 4
4 **A SAMPLE state diagram mealy machine file. KEYWORDS belonging both to 'engineering ("ERROR_STATE_AUTO")', and 'science (START_STATE)' can be intermingled in the same SDMM specification file.** . . . 4
5 A SCREENSHOT OF YERITH_QVGE. . . . 5
6 A SCREENSHOT OF YRI-DB-RUNTIME-VERIF SQL EVENT LOG. . . . 5
7 YRI-DB-RUNTIME-VERIF operation WORKFLOW (inspired from "Jon Eyolfson's RV'13 paper on TRACERORY" (https://hdl.handle.net/10012/6206)). . . . 5
8 SOFTWARE ARCHITECTURE OF YRI-DB-RUNTIME-VERIF. . . . 6
9 YERITH_QVGE software library dependencies. . . . 6
10 Workflow explanation. . . . 7
11 **SAMPLE sql recovery state diagram model in YERITH_QVGE** . . . 8
12 YERITH_QVGE user interface screenshot. . . . 9

List of Tables

1 STATE DIAGRAM MEALY MACHINE SPECIFICATION KEYWORDS IN YERITH_QVGE. **'AUTO'** KEYWORDS SPECIFIES ALSO SQL QUERY FOR GOING OUT AUTOMATICALLY FROM A FAIL (FORBIDDEN) STATE. ("SEE SECTION 9.") 4
2 YERITH_QVGE Design and Testing System Dependencies 7
3 YRI-DB-RUNTIME-VERIF Directories 7

Table 1: STATE DIAGRAM MEALY MACHINE SPECIFICATION KEYWORDS IN YERITH_QVGE. **'AUTO'** KEYWORDS SPECIFIES ALSO SQL QUERY FOR GOING OUT AUTOMATICALLY FROM A FAIL (FORBIDDEN) STATE. ("SEE SECTION 9.")

scientific keywords	engineering keywords
in_set_trace	in_sql_event_log
not_in_set_trace	not_in_sql_event_log
recovery_sql_query	recovery_sql_query
STATE	STATE
START_STATE	BEGIN_STATE
FINAL_STATE ("**FINAL_STATE_AUTO**")	END_STATE ("**END_STATE_AUTO**") / ERROR_STATE ("**ERROR_STATE_AUTO**")
IN_PRE	IN_BEFORE
IN_POST	IN_AFTER
NOT_IN_PRE	NOT_IN_BEFORE
NOT_IN_POST	NOT_IN_AFTER

Figure 1: A motivating example, as previous bug found in YERITH−ERP−9.0.
$\underline{Q0}$:= NOT_IN_BEFORE(YRI_ASSET, department.department_name).
$\overline{Q1}$:= IN_AFTER(YRI_ASSET, stocks.department_name).

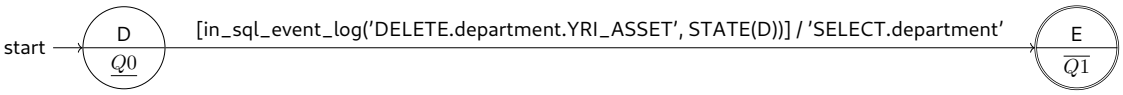


Figure 2: YERITH−ERP−9.0 administration section displaying departments ($\neg\underline{Q0}$).

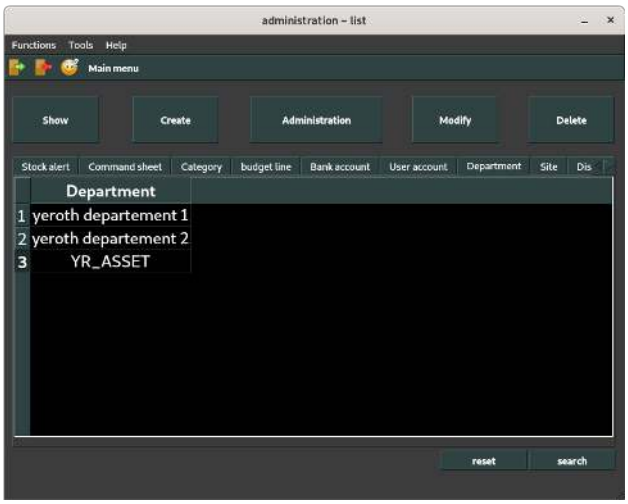


Figure 3: YERITH−ERP−9.0 stock asset window listing some assets ($\overline{Q1}$).

Figure 4: **A SAMPLE state diagram mealy machine file. KEYWORDS belonging both to 'engineering ("ERROR_STATE_AUTO")', and 'science (START_STATE)' can be intermingled in the same SDMM specification file.**

```
1. yr_sd_mealy_automaton_spec yr_missing_department_NO_DELETE
2. {
3.  START_STATE(d):NOT_IN_BEFORE(YRI_ASSET,department.department_name)
4.   ->[in_sql_event_log('DELETE.departement.YRI_ASSET',STATE(d))]/'SELECT.department'->
5.    ERROR_STATE(e):IN_AFTER(YRI_ASSET,stocks.department_name).
6. }
```

Figure 5: A SCREENSHOT OF YERITH_QVGE.

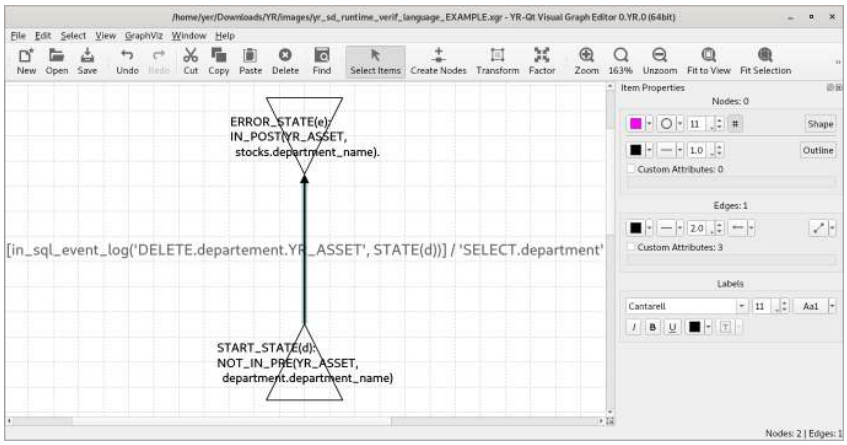


Figure 6: A SCREENSHOT OF YRI-DB-RUNTIME-VERIF SQL EVENT LOG.

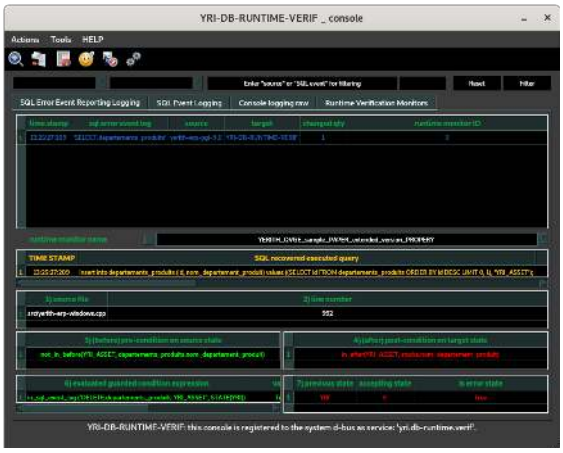
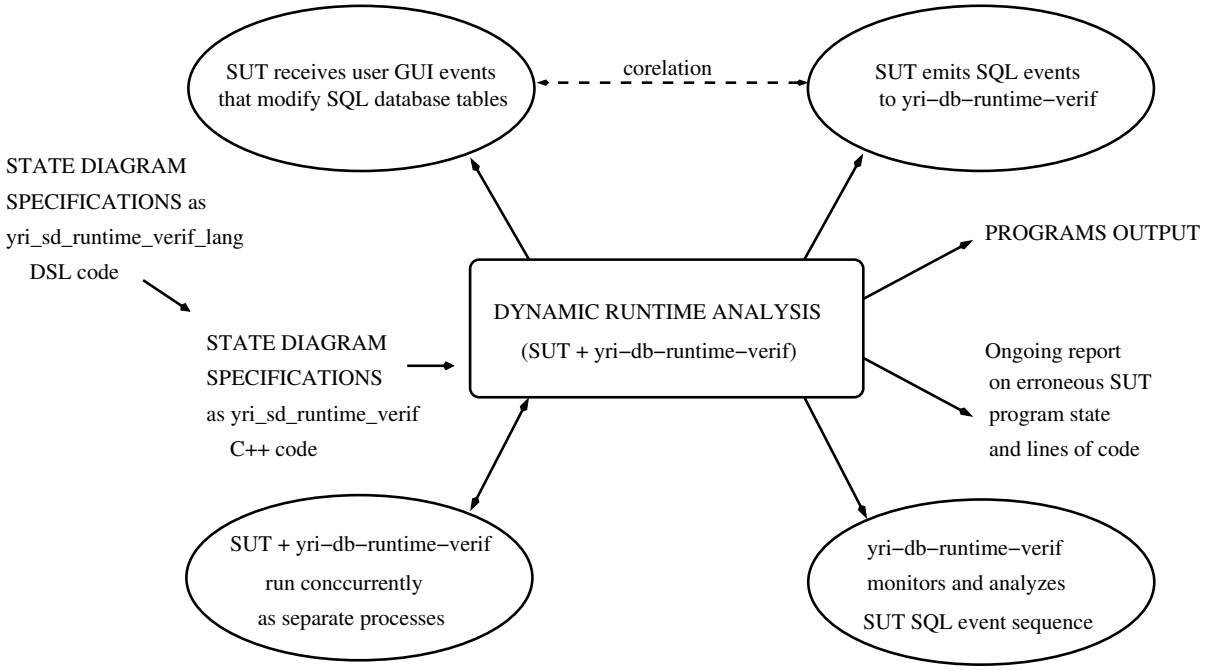
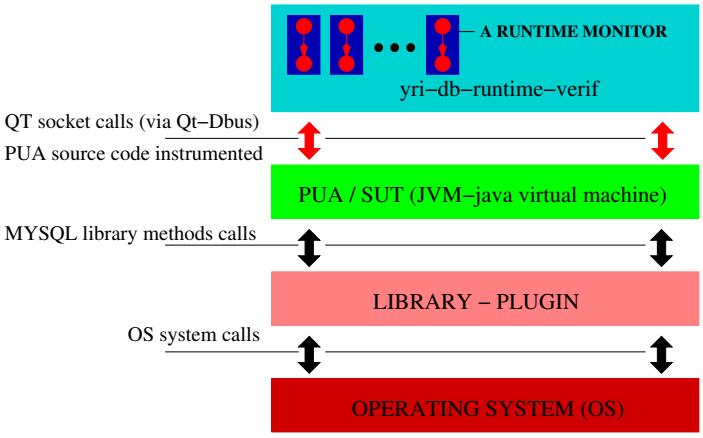


Figure 7: YRI-DB-RUNTIME-VERIF operation WORKFLOW (inspired from "Jon Eyolfson's RV'13 paper on TRACERORY" (https://hdl.handle.net/10012/6206)).



1 Introduction

Figure 8: SOFTWARE ARCHITECTURE OF YRI-DB-RUNTIME-VERIF.



This user's guide helps briefly and concisely how to create a binary executable of the runtime monitoring testing tool **YRI-DB-RUNTIME-VERIF** having user defined runtime monitors. The guide also specifies keywords allowed within runtime monitor specifications as State Diagram Mealy Machines.

YERITH_QVGE (YRI_QVGE) could be used for the following automatic generation, analysis, verification, and validation tasks:

1. Automatic generation of runtime monitoring module program to prove whether a test procedure, automated, or not, is correct with regards to a test and / or design STATE DIAGRAM MEALY MACHINE (formally described in [Nou23]).

   In effect, let the test execution be runtime monitored to watch whether accepting error states would be found.

   For instance, Junit testing environment could automatically integrate an automatically generated runtime monitor infrastructure for unit testing.

2. Automatic generation of runtime monitoring module program for any software that can emit DBus messages.

   "*Such runtime monitoring modules are for interest for special LTL model checking properties that cannot get a definite answer through use of a conventional model checker*".

3. Software design properties with SQL

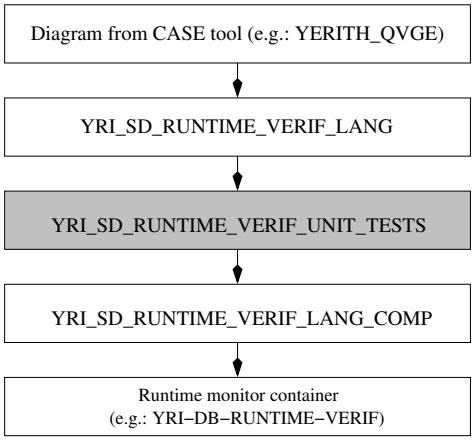
4. Software design properties including event sequences over different layers of software system architecture

5. Class diagram with sequence diagram.

[1] https://www.github.com/yerithd/yri_sd_runtime_verif
[2] https://www.github.com/yerithd/yri-db-runtime-verif
[3] Scientific: fail (forbidden) trace.
[4] Structure Query Language.

2 YERITH_QVGE (YRI_QVGE) Short Overview

Figure 9: YERITH_QVGE software library dependencies.



YERITH_QVGE is a CASE (Computer-Aided Software Engineering) design tool to generate "domain-specific language (DSL) YRI_SD_RUNTIME_VERIF_LANG [1]" files, to be inputted into the "compiler YRI_SD_RUNTIME_VERIF_LANG_COMP", so to generate C++ files for the "runtime verifier tester YRI-DB-RUNTIME-VERIF [2]" that allows for manual verification of SQL correctness properties of Graphical User Interface (GUI) software.

Figure 10 illustrates a workflow diagrammatically of the afore described process.

Figure 9 show a diagram of the afore described process; The step of the unit tests is colored in gray because it is only for developers of YERITH_QVGE intended.

YRI-DB-RUNTIME-VERIF inputs SQL correctness properties expressed using the formalism "*state diagram mealy machine* (YRI_SD_RUNTIME_VERIF_LANG)". Figure 8 illustrates a software system architecture of YRI-DB-RUNTIME-VERIF, together with the monitored program under analysis. The Free Open Source Code Software (FOSS) tool-chain of development testing is located as follows for free, EXCEPT for "YERITH_QVGE " that is a Closed Source Code Software (CSCS):

- COMPILER (i.e.: YRI_SD_RUNTIME_VERIF_LANG_COMP): https://www.github.com/yerithd/yri_sd_runtime_verif_lang
- RUNTIME VERIFIER TESTER (i.e.: YRI-DB-RUNTIME-VERIF): https://www.github.com/yerithd/yri-db-runtime-verif
- state diagram mealy machine UNIT TESTS CODE (i.e.: YRI_SD_RUNTIME_VERIF_UNIT_TESTS): https://www.github.com/yerithd/yri_sd_runtime_verif_UNIT_TESTS
- state diagram mealy machine (i.e.: YRI_SD_RUNTIME_VERIF_LANG): https://www.github.com/yerithd/yri_sd_runtime_verif

3 YERITH_QVGE (YRI_QVGE) Project Dependency

Table 2: YERITH_QVGE Design and Testing System Dependencies

PROJECT	Required Program / Library
1) YRI_SD_RUNTIME_VERIF_LANG	
2) YRI_SD_RUNTIME_VERIF_LANG_COMP	1)
3) YRI_SD_RUNTIME_VERIF_UNIT_TESTS	1)
4) YRI-DB-RUNTIME-VERIF	2)

Table 2 illustrates for each library project, which others it depends on.

4 Advantages of YERITH_QVGE

A sample state diagram mealy machine is shown in Figure 4.

WITH manual drawing of SQL CORRECTNESS PROPERTY MODEL, you are freed from manually writing "state diagram mealy machine text files" that could be tedious and lengthy. Also, editing state diagram mealy machine files manually could be more error-prone than letting a compiler (YRI_SD_RUNTIME_VERIF_LANG) do it for you.

5 State Diagram Mealy Machine (SDMM)

TABLE 1 depicts scientific keywords and their engineering counterpart that can be used in describing NOT DESIRABLE [3] SQL [4] call sequence state diagram mealy machine in YERITH_QVGE Design and Testing System.

A STATE DIAGRAM mealy machine specification is compiled into C++ code that describes a runtime monitor to be executed in the runtime monitoring tester YRI-DB-RUNTIME-VERIF. Figure 4 depicts a sample State Diagram Mealy Machine specification on a NOT DESIRABLE SQL call sequence.

5.1 HOW TO READ A "SDMM"

Figure 1 shows a finite automaton representation of the mealy machine description in Figure 4. It shall be read as follows:

- The program is in a start state D; state D is a start state since there is incoming "START" arrow into it.
- (**Pre-**) Condition $\overline{Q0}$: "department name 'YRI_ASSET' is not in table column 'department_name' of database table 'department'"; applies in state D.
- Whenever GUARD CONDITION : ***in_sql_event_log('DELETE.department.YRI_ASSET', STATE(d))***: "event 'DELETE.department.YRI_ASSET' appears in SQL event log (trace) leading to state D"; applies in state D, system under test (SUT) event 'SELECT.department' could occur.
- When SUT event 'SELECT.department' occurs, SUT is now in state E; state E is an error state because the node that represents it in Figure 1 has 2 circles on it.
- (**Post**-) Condition $\overline{Q1}$: "department name 'YRI_ASSET' is in table column 'department_name' of database table 'stocks'"; applies in state E.

  This shall not be the case since department 'YRI_ASSET' is no more defined in SUT database table 'department'.

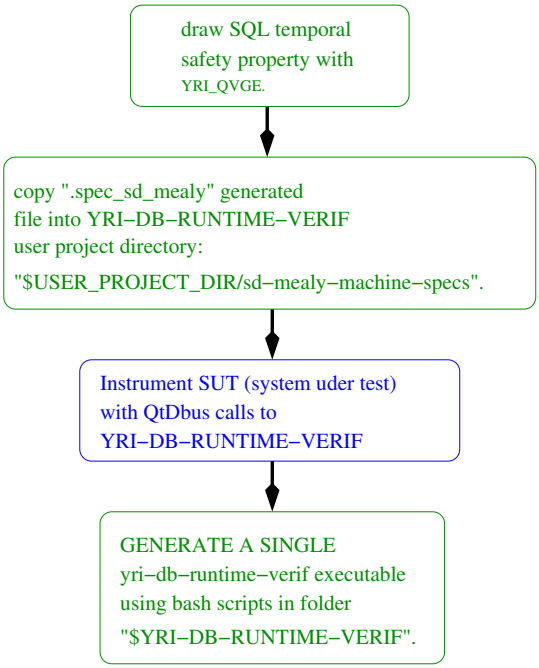
5.2 "SDMM" WITH MORE THAN 2 STATES

State Diagram Mealy Machines (SDMM) with more than 2 states have following characteristics, as detailed in scientific and engineering journal paper [Nou23] in preparation:

- Only the first transition has a pre-condition specification
- Each other transition only has a post-condition specification
- Since each state only has 1 outgoing state transition, the post-condition of the previous (incoming) state transition acts as the pre-condition of the next transition.

6 YERITH_QVGE (YRI_QVGE) Workflow

Figure 10: Workflow explanation.



The **"Design and Testing System"** YERITH_QVGE works with following workflow, as illustrated graphically in Figure 10, and in Figure 7:

1. Draw Structure Query Language (SQL) temporal safety property using drawing tool YERITH_QVGE;
2. copy the generated ".spec_sd_mealy" files into a user project directory in YRI-DB-RUNTIME-VERIF home development folder: "$YRI–DB–RUNTIME–VERIF";
3. follow the steps described in Section 7 so to gather a single executable that defines all specified runtime monitors.

7 Custom User Project (YRI–DB–RUNTIME–VERIF)

Table 3: YRI-DB-RUNTIME-VERIF Directories

Variable for illustration purposes	Meaning
$YRI–DB–RUNTIME–VERIF	root directory of YRI-DB-RUNTIME-VERIF
$YRI–DB–RUNTIME–VERIF/$USER_PROJECT	root directory of user project

Table 3 illustrates directories that will be used to describe a process to generate a single binary executable for a user's custom project with several runtime monitor specifications.

Figure 6 illustrates a screenshot of the Graphical User Interface (GUI) of YRI-DB-RUNTIME-VERIF. You can get a copy of YRI-DB-RUNTIME-VERIF using the following command:

git clone https://www.github.com/yerithd/yri-db-runtime-verif

Creating a binary executable for State Diagram Mealy Machine (SDMM) specifications consists of the following elements:

1. 'MariaDB' database connection configuration file: this file defines settings to connect to the system under test (SUT) application database; it is located in path: "$YRI-DB-RUNTIME-VERIF/YRI-DB-RUNTIME-VERIF-GUI-ELEMENTS-SETUP/yri-db-runtime-verif-database-connection.properties".

   A database connection to the SUT application database is required in order to check LTL property through the SDMM application library YRI_SD_RUNTIME_VERIF_LANG.

2. Property configuration file: this file defines environment variables necessary for building a binary executable for the user; it is located in path: "$YRI-DB-RUNTIME-VERIF/$USER_PROJECT/bin/configuration-properties.sh".

3. "$YRI-DB-RUNTIME-VERIF/$USER_PROJECT/sd-mealy-machine-specs": this directory contains user defined State Diagram Mealy Machine (SDMM) specifications to generate Corresponding runtime monitors within a single binary executable.

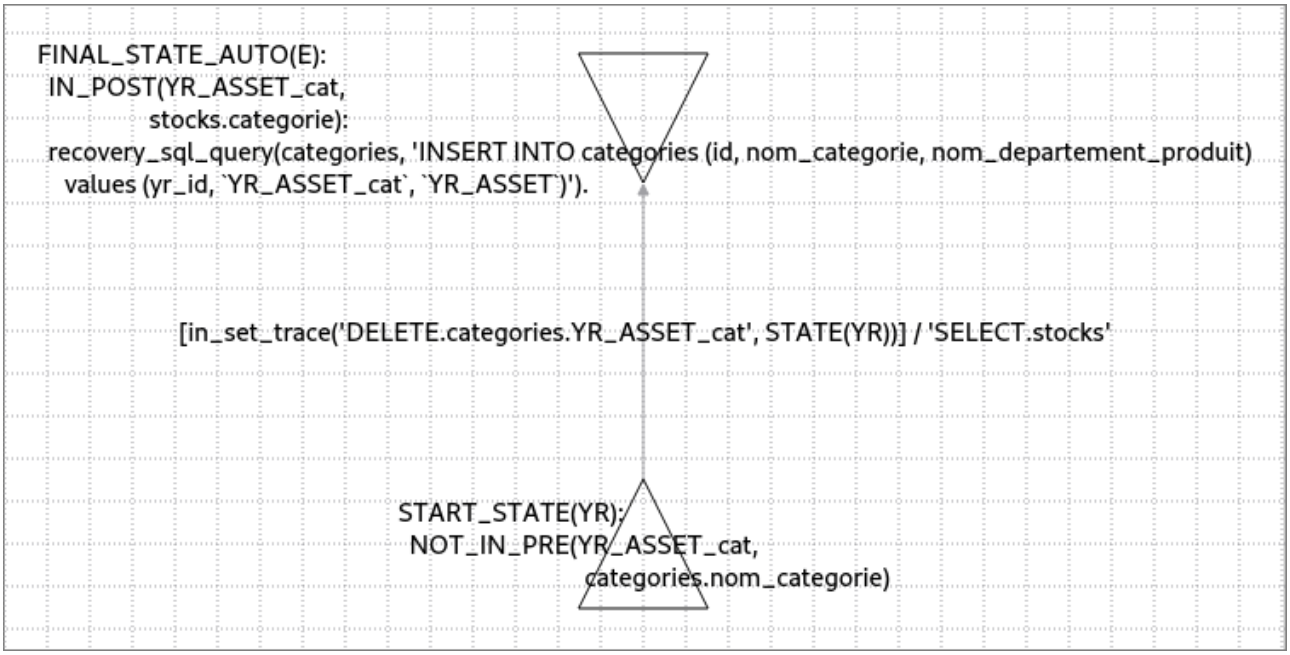
4. Generate an executable for a user defined runtime monitor:
   a) execute following command in directory "$YRI-DB-RUNTIME-VERIF":
   ```
   . ./YRI-create-executable-for-user-SDMM.sh -d $USER_PROJECT
   ```
   b) modify the LTL verification code part within the generated source code files.

   Then execute following command in directory "$YRI-DB-RUNTIME-VERIF":
   ```
   ./yr_db_runtime_verif_BUILD_DEBIAN_PACKAGE.sh
   ```
   c) uninstall YRI-DB-RUNTIME-VERIF with following command in directory "$YRI-DB-RUNTIME-VERIF":
   ```
   ./yr_DB_RUNTIME_VERIF_uninstall.sh
   ```
   d) re-install YRI-DB-RUNTIME-VERIF with following command in directory "$YRI-DB-RUNTIME-VERIF":
   ```
   ./yr_DB_RUNTIME_VERIF_INSTALL.SH
   ```

8 HOW TO START YRI-DB-RUNTIME-VERIF

- The "ELF-x64" binary executable, in the source development directory is located in full path: "$YRI-DB-RUNTIME-VERIF/bin".
- The DEBIAN-LINUX icon () of YRI-DB-RUNTIME-VERIF is located in "Applications" menu under section "Programming", and section "Accessories".
- The "ELF-x64" binary executable, after installation of the DEBIAN-LINUX package 'yri-db-runtime-verif.deb' is located in full path: "/opt/yri-db-runtime-verif/bin".

Figure 11: **SAMPLE sql recovery state diagram model in YERITH_QVGE**



9 SQL QUERY Recovery execution on demand

A user can specify which SQL command query to execute whenever a System Under Test (SUT) lands in an accepting error state. This is done using keywords ending with "AUTO", used for meaning "AUTO RECOVERY FROM FAIL STATE":

1. recovery_sql_query
2. END_STATE_AUTO
3. FINAL_STATE_AUTO
4. ERROR_STATE_AUTO.

The use of an "AUTO" keyword shall be accompanied with a use of keyword recovery_sql_query, that specifies a SQL command query to run when landing in this fail error accepting state.

9.1 Automatic SQL Command Query Generation

YERITH_QVGE implements an automatic SQL query generation strategy in case a user don't specify a SQL command query, since it could be leaved empty: Subsections 9.1.1, 9.1.2, 9.1.3, and 9.1.4 describe the strategy implemented.

9.1.1 ERROR ACCEPTING STATE for sdmm 1.

$$\frac{\texttt{not in\_before(YX, YY)} \qquad \texttt{ACTION(V)}}{\texttt{in\_after(DD, YR)}}$$

9.1.2 RECOVERY 1.

$$\frac{\texttt{in\_after}\,(\texttt{DD},\,\texttt{YR}) \qquad \texttt{ACTION}\,(\texttt{RECOVERY\_ND})}{\texttt{not in\_after}\,(\texttt{DD},\,\texttt{YR})}$$

9.1.3 RECOVERY 2 (Practical solution to be implemented in YRI-DB-RUNTIME-VERIF.

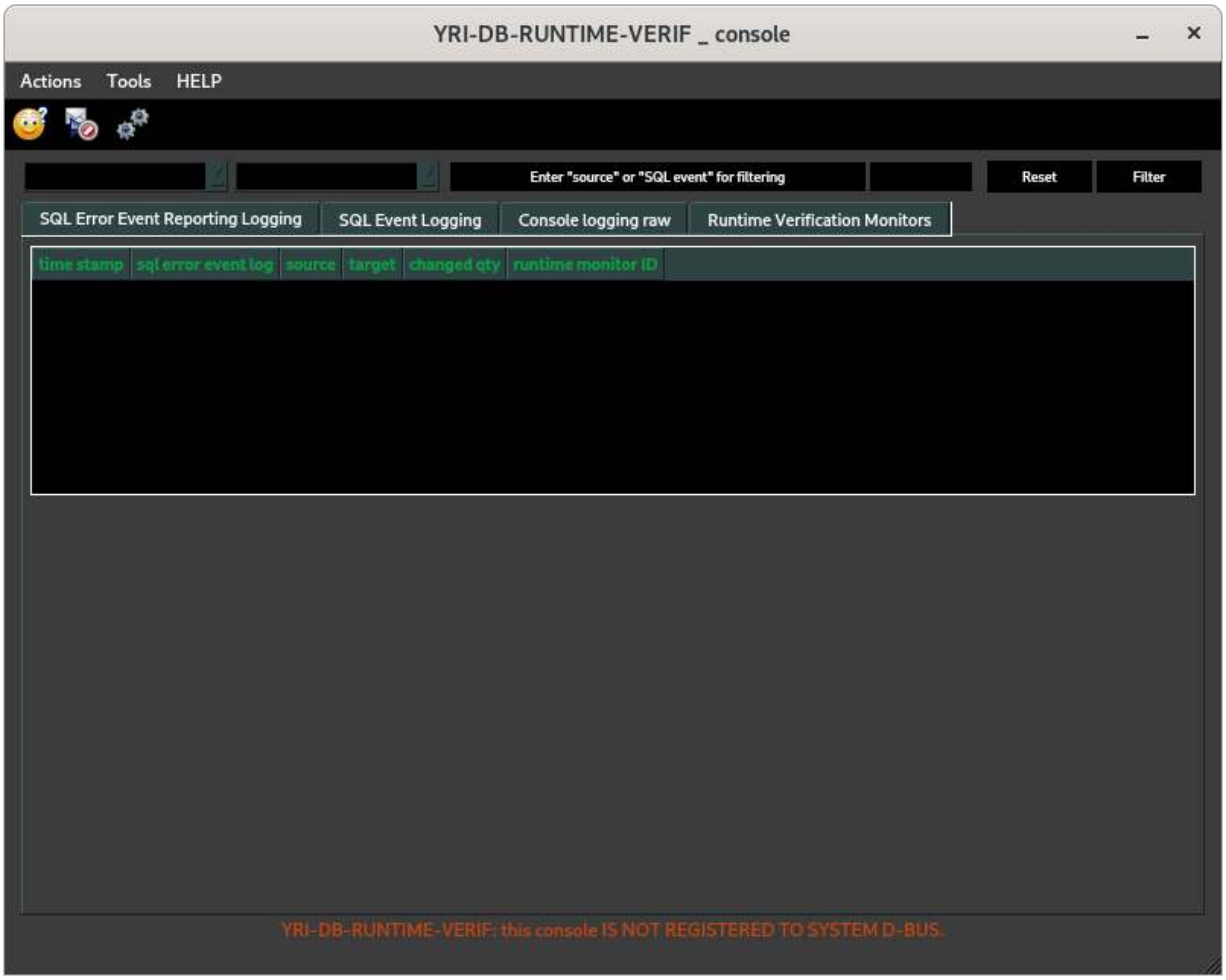
$$\frac{\texttt{in\_after}\,(\texttt{DD},\,\texttt{YR}) \qquad \texttt{ACTION}\,(\texttt{RECOVERY\_D})}{\texttt{in\_after}\,(\texttt{YX},\,\texttt{YY})}$$

9.1.4 Concrete RECOVERY 2 action.

$$\frac{\texttt{in\_after}\,(\texttt{YX},\,\texttt{YY}) \qquad \texttt{insert\_RECOVERY}\,(\texttt{YX},\,\texttt{YY})}{\texttt{in\_before}\,(\texttt{YX},\,\texttt{YY})}\ \bullet$$

10 HOW TO USE a user interface

Figure 12: YERITH_QVGE user interface screenshot.



11 Formal Scientific and Engineering Project Description

Detailed formal scientific and engineering contributions of design and testing system YERITH_QVGE can be found in ***JOURNAL ARTICLE "Runtime Verification Of SQL Correctness Properties with YRI-DB-RUNTIME-VERIF"*** [Nou23].

12 Conclusion

The graphical drawing tool YERITH_QVGE (Figure 5) costs only 2,500 EUROS. WE ONLY SUPPORT DEBIAN–LINUX (https://www.debian.org).

References

[Nou23] Xavier Noundou. A Framework for Verifying SQL Correctness Temporal Properties of GUI Software at Runtime. https://zenodo.org/records/10976659, October 2023.