RiTHM development status report

7th April, 2015

Abstract

Details about the current progress of **RiTHM** development and journal paper

1 Previous Results

- MTL parser has been developed, the monitor for MTL is being developed. **Update**: Including both Past-time and future-time MTL variants.
- For developing predicate specification language, work is done on analysis of script engines which can be used. Beanshell `http://www.beanshell.org/intro.html` is under consideration along with JavaScript engines which can be embedded into java code, and the predicate definitions could be specified using the languages of these engines.
- IronForge data, and running **RiTHM** on the properties of the data. Work in Progress.
- Integration of DIME + **RiTHM** - Papers worked on for analyzing the previous work. Work done by Smolka et al. focuses on using Markov Chains for providing probabilistic estimates on the satisfaction of specifications. On similar lines, Probabilistic Timed Automata could be used for specifying models of systems which exhibit the characteristics of incomplete-data for verification along with the requirement of hard real-time deadlines.
- 'Lessons Learnt' section for journal paper is being worked upon.
- **RiTHM** 's monitor using specifications in the format of regular expressions is being enhanced so that monitoring is trace-length independent. The coding is in progress and we now use `http://www.brics.dk/automaton/doc/index.html` to create automaton from regular expression and input trace ti this automaton event by event.

2 Previous Results

- **RiTHM** For CRV'15 competition which will be held with RV'15 conference, benchmarks are submitted for 'C' program monitoring track and Offline Monitoring track. The details of benchmarks which are submitted can be found at
  - For 'C' program monitoring track - `https://forge.imag.fr/plugins/mediawiki/wiki/crv15/index.php/C_track`
  - For Offline monitoring track `https://forge.imag.fr/plugins/mediawiki/wiki/crv15/index.php/Offline_track`
- The 'C' program benchmarks are designed to monitor a 'C' program which launches 0.1 million POSIX threads, and various properties have been specified using First Order Linear Temporal Logic (Past as well as Future time)
- 'C' program which is being monitored can be found at `https://github.com/yogirjoshi/CRVBenchMark`
- The specifications for 'C' program monitoring track are as per below
  - $\forall$ tid: pthread_create(tid) $\longrightarrow \Diamond$ pthread_running(tid)
  - $\forall$ tid: pthread_mutex_lock(tid, "ex_mutex") $\longrightarrow \Diamond$ pthread_mutex_unlock(tid, "ex_mutex")
  - $\forall$ tid: pthread_create(tid) $\longrightarrow \Diamond$ pthread_join(tid)
  - $\forall$ tid: (pthread_mutex_lock(tid, 'ex_mutex') $\vee$ pthread_mutex_destroy(tid, 'ex_mutex') $\vee$ pthread_mutex_unlock(tid, 'ex_mutex')) $\longrightarrow \Diamond^{-1}$ pthread_mutex_init(10000, 'ex_mutex')))
  - $\forall$ tid: (pthread_exit(tid)) $\longrightarrow \Diamond^{-1}$ pthread_mutex_unlock(tid, 'ex_mutex'))
- For offline monitoring track, QNX trace-logger files have been used. The trace file is at `https://github.com/yogirjoshi/datatools/blob/master/CRV1.tar.gz`. It contains 0.1 million events on which the specifications will be validated.
- Specifications are defined on various events of QNX threads. The specifications are as per below
  - $\forall$ pid, $\forall$ tid : (thcreate $\longrightarrow \Diamond$ thrunning) (Satisfied by trace)
  - $\forall > 90\%$ pid, $\forall$ tid : ($\Diamond$ threply) - Vioated by trace
  - $\forall$ pid, $\forall$ tid : ($\Box$ (thready $\longrightarrow \Diamond$ thrunning)) - Satisfied by trace
  - $\exists = 1$ pid, $\exists = 2$ tid: $\neg$ ($\Box$ ($\neg$ thdestroy)) - Vioated by trace
  - $\forall > 50\%$ pid, $\forall > 50\%$ tid: ($\Diamond$ (thsem $\vee$ thmutex)) - Vioated by trace

- Verbose LTL parser is implemented which uses verbose representation of LTL operators.
  The code is at `https://github.com/yogirjoshi/parsertools.git`.

- Verbose LTL parser is integrated with existing LTL monitor. Some new APIs added to Parser interface for rewriting the specifications into interchangable formats. The code is at `https://github.com/yogirjoshi/monitortools.git`

- **RiTHM** plugin loader is implemented so that different monitors, parsers and data-importers can be plugged in and used.
  Below example starts **RiTHM** instance to monitor LTL specifications using four valued semantics, and it uses CSV data

  ```
  java rithm.driver.RiTHMBrewer
  -specFile=/home/y2joshi/InputFiles/specsQnx
  -dataFile=/home/y2joshi/Input1.csv
  -outputFile=/home/y2joshi/InputFiles/output3.html
  -monitorClass=LTL4
  -traceParserClass=CSV
  -specParserClass=LTL
  ```

  Similarly, **RiTHM** 's another instance can be started to monitor using Verbose LTL (using 4-valued semantics), and it uses trace data in XML format

  ```
  java rithm.driver.RiTHMBrewer
  -specFile=/home/y2joshi/InputFiles/specsQnx
  -dataFile=/home/y2joshi/Input1.XML
  -outputFile=/home/y2joshi/InputFiles/output3.html
  -monitorClass=LTL4
  -traceParserClass=CSV
  -specParserClass=VLTL
  ```

- **RiTHM** can import data in CSV format and the CSV data-importer is intergrated with **RiTHM** framework

- **RiTHM** source has been refactored to use maven for project source code and build management. **RiTHM** source has been enhanced to drop some legacy APIs to make the design more scalable

- API key feature for **RiTHM** is being implemented. The API key will allow access management for **RiTHM** when used in server mode.