References

[1] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[2] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.

[3] Edward Chang, Zohar Manna, and Amir Pnueli. *Characterization of temporal property classes*, pages 474–486. Automata, Languages and Programming. Springer, 1992.

[4] Ylies Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012.

References

[1] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[2] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.

[3] Edward Chang, Zohar Manna, and Amir Pnueli. *Characterization of temporal property classes*, pages 474–486. Automata, Languages and Programming. Springer, 1992.

[4] Ylies Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012.

RiTHM development status report

14th April, 2015

Abstract

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

1 Current Results and Details about on-going items

1.1 Runtime verification in case of Missing Events

Important sections for the paper to be submitted are proposed as per below

1.1.1 Introduction

- LTL as industry standard for verification; LTL on finite paths
- Berief description of problem of *runtime verification* in the event of loss of trace, and practical scenarios; Considerations w.r.t. *reactive systems*, *instrumentation overhead*, *monitoring overhead*.
- Brief description of previous work on *monitoring* in case of loss of trace; Consideration of contributions by Smolka et al. and related papers (to be surveyed)
- Benefits and limitations of previous work.
- Description of our approach of problem solving; Extending the definition of monitorability provided by Falcone et al. in [**?**] in the event of loss of *trace*; Extending definition of *soundess* by building upon the current definition which describes *soundess* as absence of *false negatives*; Description of *persistent loss of trace* and *transient loss of trace*
- Brief description of the decision procedure to identify *monitorable* formulae from *LTL hierarchy* in [**?**] when there is a *transient loss of trace*; Details of another decision procedure to monitor LTL formulae at runtime depending upon the state of the monitor.

- Brief description of *proof* of correctness of monitoring for both decision procedures
- Benefits of our approaches w.r.t. *deadlines in reactive systems*, *reduced instrumentation overhead*, *reduced monitoring overhead*.

1.1.2 Background

- Definition of *transient loss of trace*and *persistent loss of trace*in context of *Reactive Systems*; Practical scenarios and examples. (Survey to be done for citing examples in real world)
- Formal Definitions of *transient loss of trace*and *persistent loss of trace*in context of *Runtime Verification*; Builiding up on the notion of *finite/persistent* loss of stream of alphabets.
- Overview of *Runtime Verification*
- Details of LTL - definition, semantics; Overview of LTL hierarchy by Pnueli et al. in [**?**]
- Background on LTL on finite paths - 3-valued semantics by Leucker et al. in [**?**]; 4-valued semantics by Bauer et al. in [**?**]; Method to generate monitors described in [**?**].
- Definition of monitorability by Falcone et al. in [**?**] which builds up on classes of LTL - *Prefix Properties*, *Recurrence - büchi* and *Persistence - co-büchi* properties. Highlighting the monitoring definition of *Recurrence - büchi* and *Persistence - co-büchi* properties by Falcone et al.
- Extension of 4-valued semantics by Schneider et al. in [**?**].

1.1.3 Related Work

- Contributions by Smolka et al. in [**?**] and the benefits and limitations of their approach. Emphasis on the limitations of the method to arrive at probabilities.
- Our approach and the *stringent* notion of *soundness* for our decision procedures as compared to the one proposed in [**?**] by Smolka et al.
- Survey to be done on other papers.

1.1.4 Our Approach

- We describe the first part of the problem as "Which LTL formulae are *monitorable* in the event of *transient loss of trace* and subsequent *guaranteed* recovery of the system from the loss?"

- Description of the decision procedure to generate LTL_4 monitor using the method described in [**?**]
- Describe our algorithm which asserts whether a LTL specification is *monitorable* in the event of *transient loss of trace*; We build up the algorithm on the notion of absence of states in the FSM which are labelled as $\top$ or $\bot$ and all transitions to states in the FSM which are marked as $\top_p$ and $\bot_p$ are unique for different possible conjunctions of predicate symbols or their negations i.e. each alphabet of a finite / infinite word causes transition to a unique state for the given FSM.
- Practical examples of such specifications - *Recurrence* and *Persistence* class.
- Definition of *Soundness* in the event of *transient loss of trace*; We build up the definition on the notion that if the system recovers from a *transient loss of trace*, the truth value of the monitor never allows *false-negatives*.
- Proof of correctness of the decision procedure w.r.t new defintion of *Soundness*; We build up the proof from the details top of classes of LTL hierarchy and our criteria w.r.t. the restrictions on the LTL_4 monitor. Proof includes consideration of infinite path.
- Description of second problem. A superset of the class of specifications which can be monitored in the event of *transient loss of trace*, is described. The formulae in this superset could be monitored at runtime depending on the state of LTL_4 monitor.
- Proof of correctness for the second decision procedure and examples of the associated specifications

1.1.5 Empirical Evaluation

- Examples of monitoring incomplete data w.r.t the second decision procedure.
- Details of examples where the instrumentation is performed by DIME.

1.2 RiTHM development work

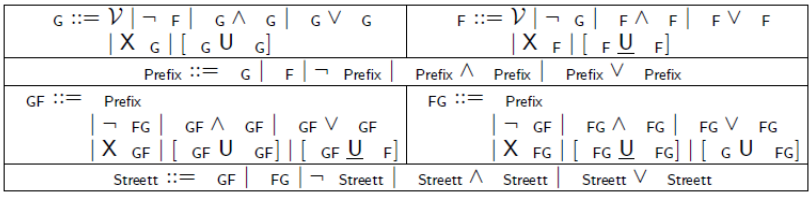
- MTL parser has been developed, the monitor for MTL is being developed. **Update**: Including both Past-time and future-time MTL variants.
- For developing predicate specification language, work is done on analysis of script engines which can be used. Beanshell `http://www.beanshell.org/intro.html` 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

2.1 Runtime verification in case of Missing Events

- **Missing Events** correspond to the case when Runtime Monitor does not receive a trace from the program which is being monitored.
- Events could be missed for a *finite* amount of time or there could be *persistent*. This loss of a stream of events could be formalized as either a *finite* path within the program when the events could not be monitored or a possible *infinite* path within the program when the events could not be monitored.
- In case of loss of trace for *finite* time, the monitor cannot keep track of a finite path which is being executed in the program and hence it cannot provide verdict of certain type of *LTL* properties.
- On the other hand, a *persistent* loss of program trace implies that the monitor cannot validate any type of *LTL* property.



- In above, *LTL* hierarchy, there are 6 classes of specifications and the class *Street* is the most expressive one and for most of the LTL formulae, it is not difficult to find an equivalent class which belongs to the class *Street*.

- Among the classes in *LTL* hierarchy, in the event of loss of trace for a *finite* path within the program, it is not possible to preserve soundness while monitoring *Safety* (P_G), *Liveness* (P_F) and *Prefix* which is boolean closure of *Safety* and *Liveness*

- This happens because a *Liveness* property could be satisfied by a *finite* path of a *reactive* system and a *Safety* property could be violated by *finite* path. This *finite* path could be the path whose trace was lost. Hence, the verdict given by a *LTL* monitor might not show the actual status.

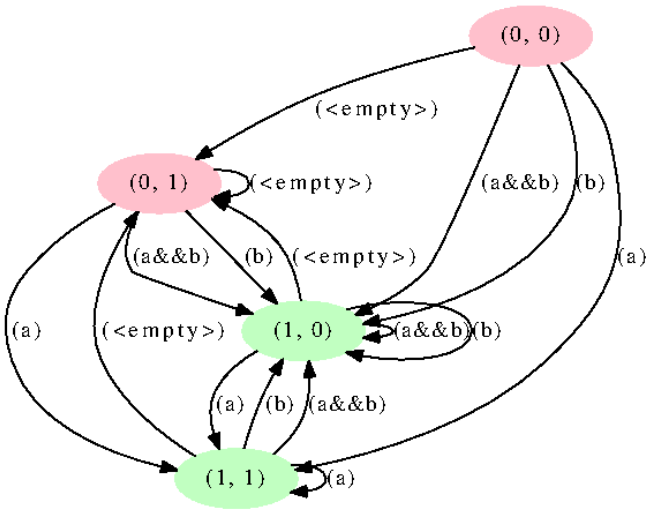
- On the other hand, the properties in the class *Recurrence* (P_{GF}) and *Persistence* (P_{FG}) could be monitored by tolerating a loss of trace for a *finite* path in the program provided certain criteria are fulfilled by states of *LTL* monitor.

- The formulas which belong to the class *Recurrence* (P_{GF}) have Büchi acceptance condition where one of the final states if the Büchi automaton should visited by the run of infinite word infinitely often. Hence, a *finite* loss of program trace would not affect the monitoring provided the monitor (which is constructed from Büchi automaton) visits one of the final states infinitely often.

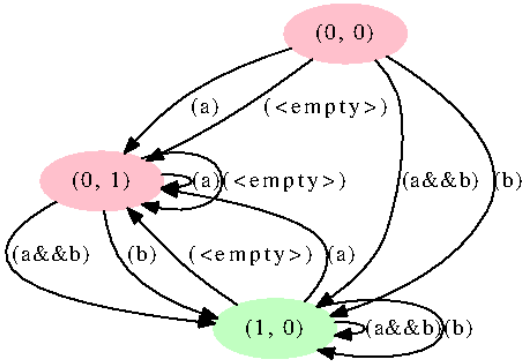
- The formulas which belong to the class *Persistence* (P_{FG}) have Persistence acceptance condition where one of the final states if the Persistence automaton should be continuously visited by the run of infinite word. Hence, a *finite* loss of program trace would not affect the monitoring provided the monitor eventually visits one of the final states in a continuous manner.

- *Strett* class, which is boolean closure of *Persistence* and *Recurrence* classes exhibits a combination of acceptance conditions of *Persistence* and *Recurrence*.

- For example., below LTL_4 monitor belongs to the *Persistence* property $\Diamond\Box a$ U b. Here, provided the monitor is in one of the states among (0,1), (1,0), (1,1), a finite loss of a stream of events could be tolerated without violating the *asymptotic* evaluation the truth value of the formula.



- Below LTL_4 monitor belongs to the *Recurrence* property $\Box\Diamond a$ U b. Here, provided the monitor is in one of the states among (0,1), (1,0), a finite loss of a stream of events could be tolerated without violating the *asymptotic* evaluation the truth value of the formula.



- Future directions can include the proof of asymptotic correctness of the valuation of truth-value in the event of *finite* loss of trace for *Recurrence* (P_{GF}), *Persistence* (P_{FG}) and *Strett* classes.

2.2 RiTHM development and CRV'15 conference updates

- **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) $\longrightarrow \diamondsuit$ pthread_running(tid)
  - $\forall$ tid: pthread_mutex_lock(tid, "ex_mutex") $\longrightarrow \diamondsuit$ pthread_mutex_unlock(tid, "ex_mutex")
  - $\forall$ tid: pthread_create(tid) $\longrightarrow \diamondsuit$ 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')) $\longrightarrow \diamondsuit^{-1}$ pthread_mutex_init(10000, 'ex_mutex')))
  - $\forall$ tid: (pthread_exit(tid)) $\longrightarrow \diamondsuit^{-1}$ 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 $\longrightarrow \diamondsuit$ thrunning) (Satisfied by trace)
  - $\forall > 90\%$ pid, $\forall$ tid : ($\diamondsuit$ threply) - Vioated by trace
  - $\forall$ pid, $\forall$ tid : ($\square$ (thready $\longrightarrow \diamondsuit$ thrunning)) - Satisfied by trace
  - $\exists = 1$ pid, $\exists = 2$ tid: $\neg$ ($\square$ ($\neg$ thdestroy)) - Vioated by trace
  - $\forall > 50\%$ pid, $\forall > 50\%$ tid: ($\diamondsuit$ (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.