Working Group 2

Standardization, benchmarks, tool interoperability

Chair: Giles Reger

September 21, 2015

Outline

Aim

Clear the landscape of formalisms and tools proposed and built for runtime verification, to design common input formats and to establish a large class of benchmarks and challenges.

Aim

Clear the landscape of formalisms and tools proposed and built for runtime verification, to design common input formats and to establish a large class of benchmarks and challenges.

- Collected a list of tools
- If yours isn't on it, add it please
- Next step, organise/categorise the list
- Very diverse

Aim

Clear the landscape of formalisms and tools proposed and built for runtime verification, to design common input formats and to establish a large class of benchmarks and challenges.

- Task A for traces
- Task B for online monitoring interfaces
- Lots of challenges (see diversity of tools)

Aim

Clear the landscape of formalisms and tools proposed and built for runtime verification, to design common input formats and to establish a large class of benchmarks and challenges.

- Some coming from Competition already
- Requires standards

Aim

Clear the landscape of formalisms and tools proposed and built for runtime verification, to design common input formats and to establish a large class of benchmarks and challenges.

- Including the annual Competition

Contents

- Competition
  - WG should encourage/support
  - I am one of the organisers this year
- Tasks
  - Set up to informally discuss some sub-goals
  - Each has intended output
  - Two started, some progress, third will start soon

CRV15

- Second competition
- New chairs
- Same structure, scoring etc

- Results coming on Wednesday

CRV15 Benchmarks

- Domains
  - Java API
  - Resource management
  - Concurrency
  - Security
  - Programming Patterns
  - Online Systems
  - Communication Systems
  - SQL usage
  - Exam systems
  - (Abstract with letters)
- Languages
  - State machines
  - Temporal logic
  - Rule systems
  - Code annotations
- Input formats
  - Traces - updated definitions for CSV, XML and JSON formats
    - Only CSV used.
  - Program instrumentation - AspectJ, manual, own code rewriting

CRV15 Tools

- Offline monitoring
  - MarQ
  - RVMonitor
  - LogFire
- Online monitoring of Java programs
  - MarQ
  - Mufin
- Online monitoring of C programs
  - E-ACSL
  - RVMonitor

- Others entered but did not make it to the evaluation stage
- Some submitted benchmarks but did not compete

CRV15 other observations

- Many specifications combined parts of a specification language and ad-hoc programming
  - Is the distinction between specification language and programming important/useful?
- Lots of initial interest but low numbers in final evaluation
  - Structure of competition requires lots of work from participants
  - How should the competition evolve?
- Nobody (including myself) fully followed the instructions!

Task A

- Goals
  - Propose a few trace formats that RV related tools are encouraged to support
  - (Future) develop common parsers
- Main Challenges
  - Keeping things flexible whilst efficiently parsable

Existing Formats: Competition

- XML, JSON and CSV
- Vary slightly between 2014 and 2015 (updated to follow standards)
- Each capture *events* which are abstractly

```
NAME{
  field1 : value1,
  ...
  fieldn : valuen
}
```

- In CSV field names are difficult to capture, use header
- Mostly CSV used
- Parameter values seen as strings to be interpreted by monitor

Existing Formats: BeepBeep

- Events are free-form XML structures
- Same sub-structure (with the same parameter names) can be repeated an arbitrary number of times with different values
- For example event `pingus` have many `pingu` elements

```
<pingus>
  <pingu><id>0</id><x>1</x><y>10</y></pingu>
  <pingu><id>9</id><x>34</x><y>6</y></pingu>
  ...
</pingus>
```

Existing Formats: MonPoly

- Encode a sequence of timestamped databases
- For example

```
@10 login(Alice) login(Bob)
@10 login(Charlie)
```

- login(Alice) and login(Bob) unordered but both before login(Charlie). All three happen at timestamp 10.

Existing Formats: OCLR-Check

- Natively supports XML, can translate CSV into this
- Traces are sequence of trace elements
- A trace element contains two associations:
  - an event (e.g., concrete event) with an event name;
  - a timestamp with an integer value

Existing Formats: MarQ

- Supports competition formats
- Has internal notion of event
- Define *translator* objects to transform between. These can
  - Ignore events entirely
  - Drop parameters
  - Reorder parameters
  - Parse parameter values (integer, boolean, double currently)
  - Intern parameter values (making them equal under $==$)

Defining Traces and Events

- A trace is a collection of events and an ordering
- Typically the ordering is total and the collection is a sequence
- Would like to support partial orders

- An event is a record of the analysed systems action or state
  - Concrete in the trace, abstract in the specification
  - Has a name (?)
  - Has a (possibly empty) collection of *parameters*

- Separate abstract definitions from concrete structures

Some Questions...

- To what extent should parameters be structured?
  - Should we support (for example) lists of 'values' as part of the trace definition
  - What about datatypes such as integers or strings?
- Should time be treated specially?
  - To enforce ordering by time
  - To allow easy filtering by time
- Should events with the same name have the same signature?
  - Should an event have a name?
  - Do parameters need a notion of type?
  - Do parameters need names?
  - Parameter list versus parameter 'map'
- How to define equality between parameter values?
  - Semantic vs reference equality in Java
  - Temporal equality i.e. values only equal within a time period. Can be used to encode lifetime of object e.g. garbage collection

Some Questions...

- To what extent should parameters be structured?
  - Should we support (for example) lists of 'values' as part of the trace definition
  - What about datatypes such as integers or strings?
- Should time be treated specially?
  - To enforce ordering by time
  - To allow easy filtering by time
- Should events with the same name have the same signature?
  - Should an event have a name?
  - Do parameters need a notion of type?
  - Do parameters need names?
  - Parameter list versus parameter 'map'
- How to define equality between parameter values?
  - Semantic vs reference equality in Java
  - Temporal equality i.e. values only equal within a time period. Can be used to encode lifetime of object e.g. garbage collection

Some Questions...

- To what extent should parameters be structured?
  - Should we support (for example) lists of 'values' as part of the trace definition
  - What about datatypes such as integers or strings?
- Should time be treated specially?
  - To enforce ordering by time
  - To allow easy filtering by time
- Should events with the same name have the same signature?
  - Should an event have a name?
  - Do parameters need a notion of type?
  - Do parameters need names?
  - Parameter list versus parameter 'map'
- How to define equality between parameter values?
  - Semantic vs reference equality in Java
  - Temporal equality i.e. values only equal within a time period. Can be used to encode lifetime of object e.g. garbage collection

Some Questions...

- To what extent should parameters be structured?
  - Should we support (for example) lists of 'values' as part of the trace definition
  - What about datatypes such as integers or strings?
- Should time be treated specially?
  - To enforce ordering by time
  - To allow easy filtering by time
- Should events with the same name have the same signature?
  - Should an event have a name?
  - Do parameters need a notion of type?
  - Do parameters need names?
  - Parameter list versus parameter 'map'
- How to define equality between parameter values?
  - Semantic vs reference equality in Java
  - Temporal equality i.e. values only equal within a time period. Can be used to encode lifetime of object e.g. garbage collection

Some Questions...

- To what extent should parameters be structured?
  - Should we support (for example) lists of 'values' as part of the trace definition
  - What about datatypes such as integers or strings?
- Should time be treated specially?
  - To enforce ordering by time
  - To allow easy filtering by time
- Should events with the same name have the same signature?
  - Should an event have a name?
  - Do parameters need a notion of type?
  - Do parameters need names?
  - Parameter list versus parameter 'map'
- How to define equality between parameter values?
  - Semantic vs reference equality in Java
  - Temporal equality i.e. values only equal within a time period. Can be used to encode lifetime of object e.g. garbage collection

Some Decisions... probably

- Prefer JSON as a principal format due to its flexibility and support of structured values
  - But strong support for CSV due to compactness
- Support for metadata seems useful
  - Secondary file? In JSON could be header record?
  - Things such as ordering, time units, equality...
- Want to support online tools by parsing and replaying using Monitor Interface...

Task B

- Goals
  - To consider the question where does monitoring end and instrumentation begin?
  - To explore the notion of a Monitoring Interface (MI) that is able to sit between a monitoring technique and an instrumentation technique
  - Define a set of MI Standards for a variety of appropriate RV settings
  - Propose concrete instantiations of the MI Standards
  - Encourage the adoption of the MI Standard in tools

- Main Challenges
  - Separating monitoring and instrumentation... should we?
  - Very diverse area... target languages, levels of abstraction, specification dependence,...

Task B

- Goals
  - To consider the question where does monitoring end and instrumentation begin?
  - To explore the notion of a Monitoring Interface (MI) that is able to sit between a monitoring technique and an instrumentation technique
  - Define a set of MI Standards for a variety of appropriate RV settings
  - Propose concrete instantiations of the MI Standards
  - Encourage the adoption of the MI Standard in tools
- Main Challenges
  - Separating monitoring and instrumentation... should we?
  - Very diverse area... target languages, levels of abstraction, specification dependence,...

The Monitor Interface Idea

- Support a Many-to-Many setup with may benchmarks using a single interface and many monitoring tools supporting that interface
- Will improve tool reuse and tool evaluation (including competition)
- Scope still very large (previous question key)

Current Status

- **JavaMOP**. Automatically generates AspectJ for monitoring Java programs, the events form part of the specification as pointcut definitions. Also has method for monitoring C programs (?)
- **tracematches.** Is an extension of the AspectJ language and is therefore tightly coupled with instrumentation technique.
- **MarQ**. Is an outline tool. Commonly used with ad-hoc AspectJ. Currently some ad-hoc C code exists monitoring C/C++ code. The interface consists of the methods monitor(int event) and monitor(int event, Object[] args).
- **RuleR.** Outline tool (used with AspectJ). Interface with methods dispatch(String event) and dispatch(String event, Object[] args).

Current Status

- **Larva.** An outline tool which generates AspectJ from the specification. This in turn interacts with the monitoring classes by communicating events which occur.
- **E-ACSL.** Inline monitoring for C programs. Take as input a C program and a set of E-ACSL formal specifications and generate an instrumented C program which reports the invalid properties (just the first one, by default). There is currently no explicit notion of events.
- **JUnit-RV**
- **Mufin**
- **LogFire**

A Proposal for Java

- An initial proposal that has already led to much discussion...

```
public interface MonitorInterface<E,P,V>{
    public V event(E name, P... parameters);
    public V garbage(Object objects);

    public void continue();

    public V getCurrentVerdict();

    public boolean providesContinue();
    public boolean supportsParametricEvents();
}
```

Discussion Points

- Many similar to Task A

- Should time be captured in the interface or handled by the monitor?
- How do we encode notions of parameter equality/signature?
- How do we capture multiple property checking?
- Should parameter objects be immutable? (probably not)
- How to handle multiple violations of specification?

Other Proposals

- Well-defined parsable output
  - For verdicts
  - Also statistics
  - What about *explanations*?
- Support for pretty-printing and debugging
- Support for asynchronous monitoring
- Support for Save/Restore of monitor state
- Support for Initialise/Reset of monitor state (see continue above)

Stuff to do

- Finish these discussions
- Write everything down
- Code basic interfaces and examples, try and get people to use them

- Discussions have been very Java-focussed
- Need to consider other languages, such as C
- Here the divide between monitor and instrumentation may be even trickier

Task C

- Goals
  - Develop a common set of specifications that can be used to compare, evaluation and explore RV techniques
  - Design a categorisation to describe such specifications; towards a general taxonomy
- Non-Goals
  - Fix a general specification language
  - Force a format for inputs (expect usage of standards where possible)
- Main Challenges
  - Ensuring that specifications written in different languages coincide
  - The engineering effort required to setup and maintain

The Idea

- Motivations
  - Many papers use similar specifications to introduce their languages
  - Comparison between specifications in different languages is rarely done
  - Evaluation (such as the competition) requires lots of effort
- Have an *online resource* where
  - Specifications can be browsed
  - Organised around a categorisation of specifications
  - Repository can be easily downloaded
- Can be a Wiki format but requires uploading + storage of files

Proposed Specification Format

- One Informal Description
- List of references where specification is used in literature
- Category (from categorisation to be defined)

- Many
  - Formal specification in a RV specification language
  - Short explanatory trace exemplifying behaviour
  - Benchmark object
    - i.e. trace/program that should have certain status
    - Defining format for these objects out of scope of this task.
    - But should use formats from Tasks A and B where possible

- Notes on relation to other specifications
  - i.e. 'is variant of A' or 'is extension of B to cover timeouts'

Planning

- Upcoming status report on Task A and Task B
- Then will continue discussions, hope to develop final proposals by year end
- Ideally these will be accompanied with some parsers/examples etc

- Collect additional interest for Task C and begin discussions

- Planning for next year's competition should start