What is a Trace? A Runtime Verification Perspective

Giles Reger[1] Klaus Havelund[2]

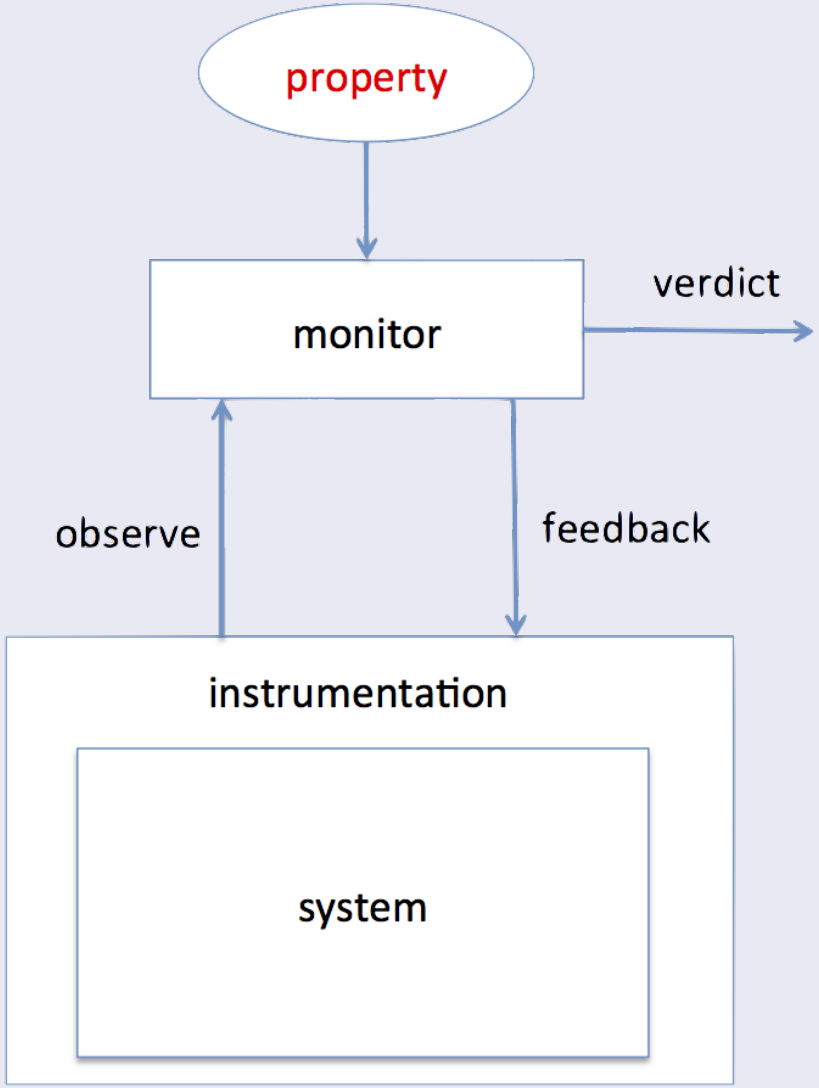
[1]University of Manchester, Manchester, UK

[2]Jet Propulsion Laboratory, California Inst. of Technology, USA

ISoLa 2016 – Corfu, October 12, 2016

Runtime Verification Perspective

The Picture



Where's the Trace?

Specification

- Traces as models i.e. structures that satisfy properties
- Different levels of semantics, commonly trace semantics
- May not be (fully) formalised
- Different kinds of models (data, time)

Instrumentation

- Defined as a set of instrumentation points
- Instrumentation can be separate and automatic (e.g. AOP)
- Instrumentation can be manual
- Domain dependent (Java, C, Erlang, Python, Hardware)

Log File

- May need to map information in log file to events
- Might contain a lot of other information
- May be spread between many files

The Logical Divide

Two Camps Divided

- Specification Language
- Formal and Abstract
- Neat and Tidy

- Instrumentation
- Log Files
- Hacky and Concrete
- Often Messy

Two extremes (nobody is wrong, nobody is right)

- Design a nice abstract specification language, implement a monitoring algorithm and add some instrumentation
- Start with the instrumentation, design a monitoring algorithm that does something sensible, realise we have a specification language

Why do we care?

Tool Interoperability

- Can tools operate on the same log files?
- Can tools interface with the same instrumentation?
- If not, can we translate log files or update instrumentation?
- How do we compare tools?

Tool Applicability

- Traces exist in other places, can we apply readily apply RV there?
- What do we need in a trace for RV?

Theoretical Relationships

- Question of relationship between specification languages needs formal and common notions of trace...

The Remainder of this Talk

Three Questions

- Status: What kinds of traces do we have?
- Contents: What goes in a trace?
- Format: How should we record log files?

Traces as Models

Dimensions

- Finite (complete or prefix)
- Quantitative or qualitative notion of time
- Single or multiple events per time point
- Data carrying events or propositional
- Declared or universal alphabet
- Time as data or inbuilt with pointwise or continious semantics
- Non-event based structures i.e. interpreted functions

Formal Structures

- Alphabet of event names Σ, trace is a finite sequence over Σ
- Parametric events include data parameters i.e. $a(2,3)$, leads to parametric traces, also similar notion of data words
- Timed words (pair events with time), signal function ($\mathbb{R}_+ \to 2^\Sigma$)
- Structured data (spatio-temporal logics, structured events (XML))

Instrumentation Methods

For Java

- AOP: AspectJ, Disl
- JVMTI (Agents)
- Reflection (e.g. JUnitRV)
- Java-MaC

For C/C++

- AOP: RMOR, AspectC++, InterAspect
- Rewriting (E-ASCL, RiTHM)

For Other Software

- Dtrace
- Erlang tracing, more recently AOP

For Hardware

- Bus sniffing (e.g. BusMOP) is inherently event-triggered
- Directly access registers/signals as circuit, sample-based

Instrumentation Discussion

Is Instrumentation part of RV?

- Where does the monitor end and instrumentation start?
- Clearly, instrumentation is a research activity but should it affect the design of monitoring algorithms?

Suggestion: Monitor Interface

- Introduce a standard interface between monitor and instrumentation
- The interface defines the trace
- Advantage: introduces layer of abstraction that separates concerns, allows for better re-usability of tools/benchmarks
- Disadvantages: difficult to perform optimisations such as inlining and distribution of monitoring, assumes monitor is Outline

Other Sources of Traces

Who else talks about traces?

- Web servers
- Databases
- System logging

Can we view those traces as our traces?

- The problem of dealing with traces not recorded for RV
- Often these traces are incomplete, how do we deal with this?
- Traces may come from part way through a continuous run, how do we deal with bootstrapping i.e. we don't know the past

What goes in a trace?

What goes in an event?

- Event name (usually, necessary?)
- Time-stamp (optionally)
- Data parameters (optionally)
  - Ordered or named? e.g. $(2, 5)$ vs $[x = 2, y = 5]$
  - Typed? Do they support operations?
  - Structured? Do we know the structure?
  - Defining equality between data values

How are they organised?

- A (partial) ordering between events

What else do we need?

- Potentially need to give other information such as the alphabet or domains of quantification separately
- Sampling rate or data about uncertainty
- Contextual information such as garbage collection

Tool Format : BeepBeep

Example

```
<trace>
<message>
<timestamp>1464984222599</timestamp>
<characters>
<character>
  <id>0</id>
  <status>FALLER</status>
  <position>
    <x>50.166668</x>
    <y>38.025</y>
  </position>
  <velocity>
    <x>0.16666667</x>
    <y>0.025000002</y>
  </velocity>
</character>
...
```

Highlights

- XML with some predefined tags
- Specification language defined over XML events

Tool Format : OCLR-Check

Example

```
<?xml version="1.0" encoding="ASCII"?>
<trace:Trace
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:trace="http://www.svv.lu/offline/trace/Trace">
  <traceElements index="1" event="//@events.0">
    <timestamp value="1"/>
  </traceElements>
  <traceElements index="2" event="//@events.1">
    <timestamp value="2"/>
  </traceElements>
  ...
</trace>
```

Highlights

- Traces are propositional
- Comes with a Schema

Tool Format : MonPoly

Example

```
@946686924
mgr_S (Alice,Kevin)
@946688989
mgr_S (Bob,Lucy)
@946690031
mgr_S (Charlie,Mary)
@946691392
mgr_S (Dave,Neil)
@946693111
mgr_S (Eve,Otto)
publish (Thomas,16)
approve (Eve,500)
@946693131
mgr_S (Felix,Peter)
publish (Ruby,149)
approve (Charlie,159)
```

Highlights

- Timestamp declarations separate time points
- Events within a time point are unordered
- Parameters given as ordered list

Competition format: CSV

Example

```
command, x, y
move, 3, 4
draw, 0, 4
move, 0, 0
draw, 3, 4
```

Highlights

- Single line per event
- Optionally use a header to define column names
- Very efficient parsing

Disadvantages

- How do we represent variable data values
- Cannot easily represent structured data or metadata

Competition format: CSV

Example

```
event, map, collection, iterator
updateMap, 6750210, ,
createColl,6750210, 2081191879,
createIter, , 2081191879, 910091170
useIter, , , 910091170
updateMap, 1183888521, ,
```

Highlights

- Single line per event
- Optionally use a header to define column names
- Very efficient parsing

Disadvantages

- How do we represent variable data values
- Cannot easily represent structured data or metadata

Competition format: CSV

Example

```
updateMap, 6750210
createColl, 6750210, 2081191879
createIter, 2081191879, 910091170
useIter, 910091170
updateMap, 6750210
```

Highlights

- Single line per event
- Optionally use a header to define column names
- Very efficient parsing

Disadvantages

- How do we represent variable data values
- Cannot easily represent structured data or metadata

Competition format: CSV

Example

```
updateMap, map, 6750210
createColl, map, 6750210, collection, 2081191879
createIter, collection, 2081191879, iterator, 910091170
useIter, iterator, 910091170
updateMap, map, 6750210
```

Highlights

- Single line per event
- Optionally use a header to define column names
- Very efficient parsing

Disadvantages

- How do we represent variable data values
- Cannot easily represent structured data or metadata

Competition format: XML

Example

```
<log>
     <event >
       <name>createColl</name>
       <field>
         <name>map</name>
         <value>6750210</value>
       </field>
      <field>
         <name>collection</name>
         <value>2081191879</value>
       </field>
     </event>
</log>
```

Highlights

- Lots of structure via tags
- Can include metadata in tags

Competition format: XML

Example

```
<log>
     <event timestamp="1462810918">
       <name>createColl</name>
       <field>
         <name>map</name>
         <value>6750210</value>
       </field>
      <field>
         <name>collection</name>
         <value>2081191879</value>
       </field>
     </event>
</log>
```

Highlights

- Lots of structure via tags
- Can include metadata in tags

Competition format: XML

Example

```
<log>
     <event>
       <name>createColl</name>
       <value>6750210</value>
       <value>2081191879</value>
    </event>
   </log>
```

Highlights

- Lots of structure via tags
- Can include metadata in tags
- Disadvantage: very verbose, even when compacted
- Can validate against schema

Competition format: JSON

Example

```
[
   {
     "createColl" : {
       "map" : "6750210",
       "collection" : "2081191879"
      }
   }
 ]
```

Highlights

- Stores attribute-value pairs and arrays
- More concise than XML
- Can use arrays to model positional arguments

Competition format: JSON

Example

```
[
     {"updateMap" : ["6750210"]},
     {"createColl" : ["6750210", "2081191879"]},
     {"createIter" : ["2081191879", "910091170"]},
     {"useIter" : ["910091170"]},
     {"updateMap" : ["6750210"]}
   ]
```

Highlights

- Stores attribute-value pairs and arrays
- More concise than XML
- Can use arrays to model positional arguments

What format should we use?

From three to... three

- Probably JSON
  - I prefer the condensed version but don't know the implications for structured data
- But CSV covers most use cases and is easy to work with
  - I would prefer the no-header, unnamed version with separate alphabet information
- I'm not a fan of XML but other people are
- So probably all three!!

Another disadvantage of CSV

In the 1st competition MonPoly had to translate their multiple events per time-point format using time-point (tp) and time-stamp (ts) fields...!

```
event, tp, ts, c, t, a
trans, 0, 32, 1797, 14581, 176
trans, 1, 32, 4187, 23430, 2144
trans, 2, 32, 1662, 46471, 2486
```

Summary

A trace is a trace

- Like Brexit is Brexit, a trace is a trace!
- Practically, we want similar notions for interoperability
- Theoretically, we want similar notions to compare languages

More Challenges

- Concurrent and Distributed Systems
- Rolling logs
- Uncertainty