CS 6156 Runtime Verification

Events, Traces, Properties

Owolabi Legunsen

Concepts discussed in this class

- RV checks traces of system events against properties that are specified in some language

- But, what do the terms in blue mean?

- These terms occur a lot in the RV literature

Let’s discuss…

- What is an event?

- What is a trace?

- What is a property?

Let’s discuss…

- What is an event?

something that we can observe during system exec

most atomic "thing"

- What is a trace?

One path through a CFG

a sequence of events that happened during exec

- What is a property?

certain sequences of events

a way to describe a set of traces

What is an Event?

- A mathematical (formal languages) view
  - An event as a symbol *e* in an alphabet Σ, where Σ is a finite set of such symbols

- A logical view
  - An event as an atomic predicate in a logical formula

- A practical view
  - An event as a state/step during system execution

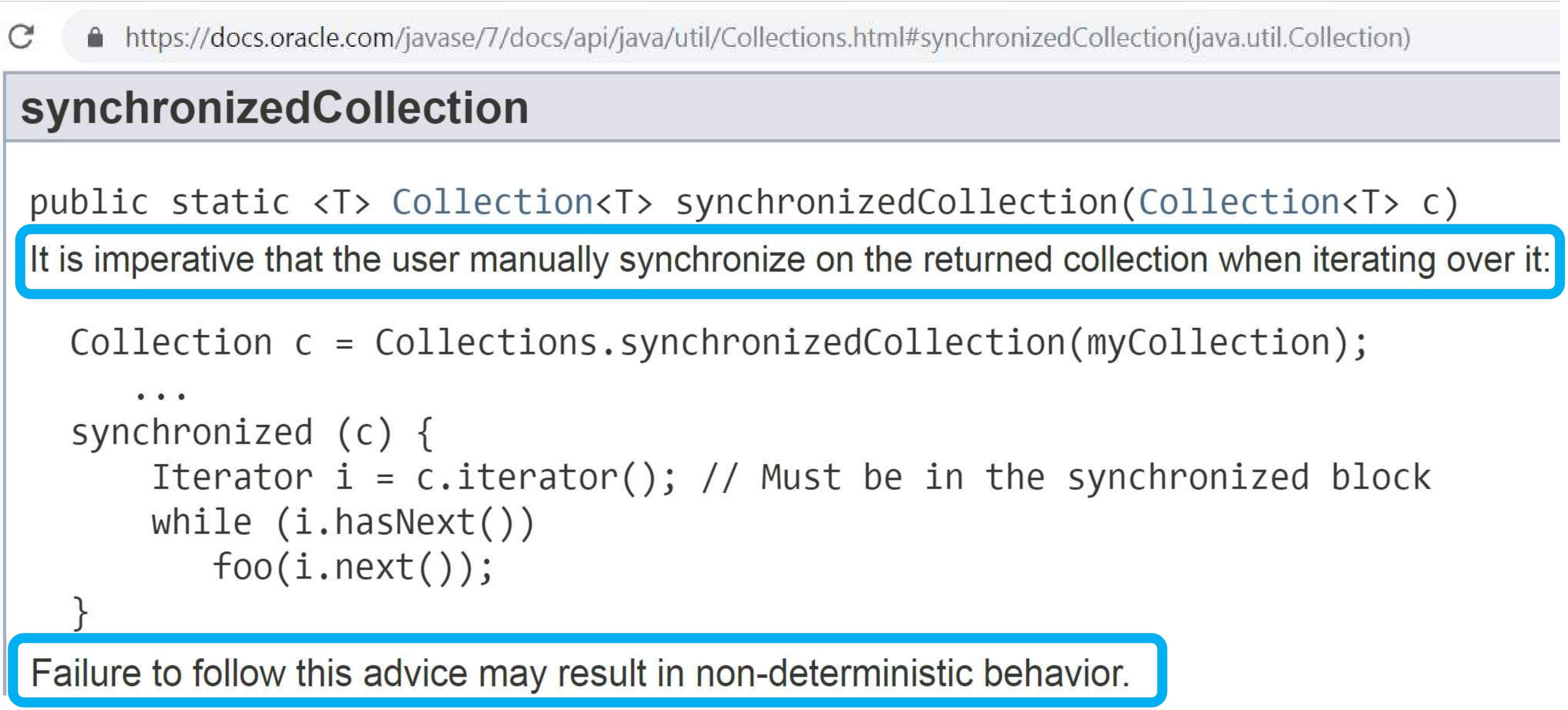
When/how you’ll see these views

- View of events as symbols is common when defining concepts or proving theorems in RV
- View of events as atomic predicates is often used when specifying properties
- View of events as execution state/steps is required for defining what to observe in system executions

↓

Instrumentation

Example: CSC spec



https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedCollection(java.util.Collection)

synchronizedCollection

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```

It is imperative that the user manually synchronize on the returned collection when iterating over it:

```
  Collection c = Collections.synchronizedCollection(myCollection);
     ...
  synchronized (c) {
      Iterator i = c.iterator(); // Must be in the synchronized block
      while (i.hasNext())
         foo(i.next());
  }
```

Failure to follow this advice may result in non-deterministic behavior.

What events (execution states/steps) do we care about?

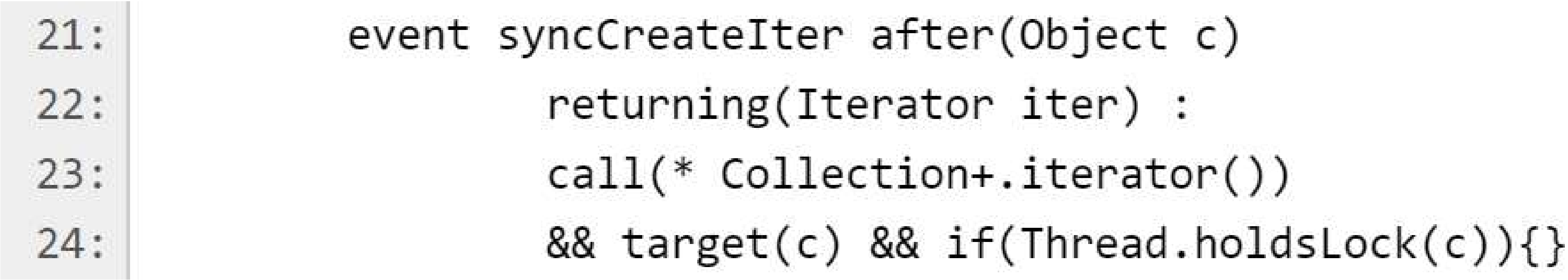
Example: events in the CSC spec

Demo

What view(s) of events are in CSC?

Events as execution states/steps

- Examples: method calls, field/variable access, lock acquisition/release
- One often must define the conditions under which to observe the execution step



```
        event syncCreateIter after(Object c)
                returning(Iterator iter) :
                call(* Collection+.iterator())
                && target(c) && if(Thread.holdsLock(c)){}
```

- Events can carry data, or they can be parametric

What view of events is this? (1)

- A property is a logical formula over a set of events[1]

[1] Legunsen et al., Techniques for Evolution-Aware Runtime Verification, ICST 2019

What view of events is this? (2)

An RV tool instruments the program based on the properties so that executing the instrumented program generates events and creates monitors that listen to events and check properties?[1]

[1] Legunsen et al., Techniques for Evolution-Aware Runtime Verification, ICST 2019

What view of events is this? (3)

- A bad prefix is a finite sequence of events which cannot be the prefix of any accepting trace.[2]

[2] d’Amorim et al., Efficient Monitoring of ω-Languages, CAV 2005

Takeaway message on events

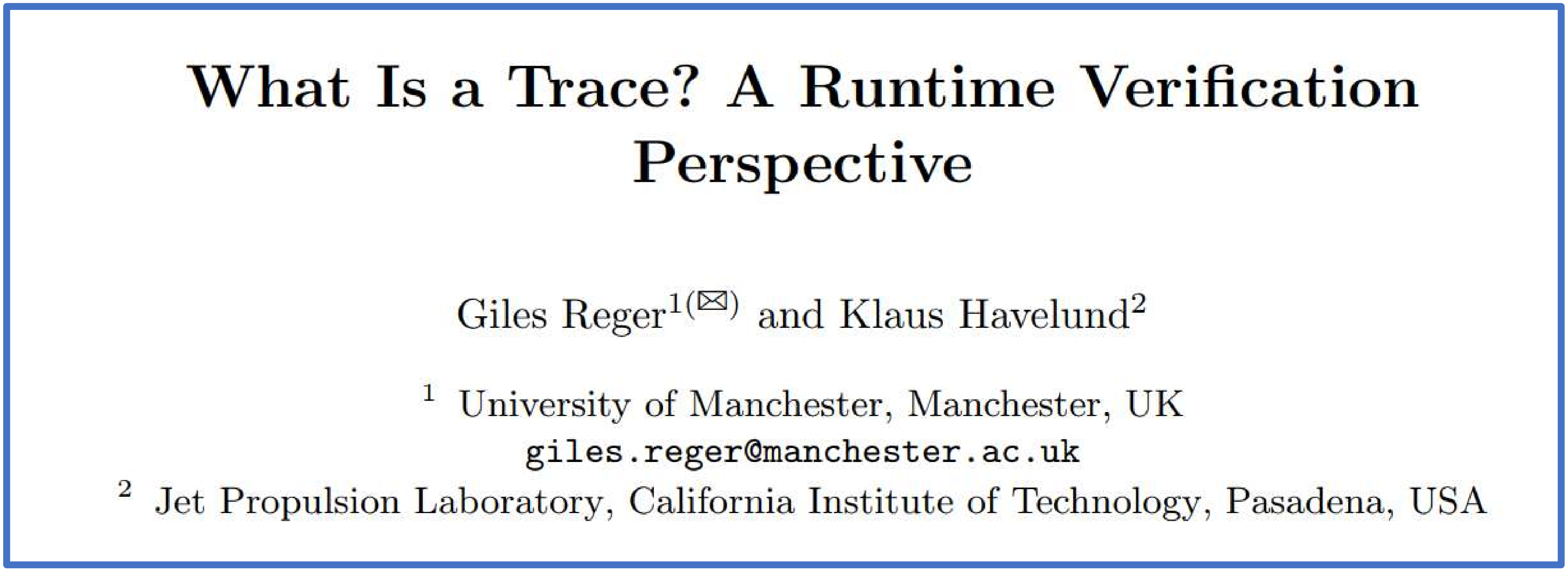
- Events are fundamental in RV theory and practice
- But RV literature will often mix the different views of events
- So, when you read papers on RV, be careful to distinguish these views

Any questions about events?

?

What is a trace?

There are many notions/views of traces in RV, e.g.,



What Is a Trace? A Runtime Verification Perspective

Giles Reger[1(✉)] and Klaus Havelund[2]

[1] University of Manchester, Manchester, UK
giles.reger@manchester.ac.uk

[2] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA

http://havelund.com/Publications/traces-isola-2016.pdf

What is a trace? Some views..

- A trace is a sequence of events
  - In practice: sequences are finite
  - In theory: we reason about infinite sequences (why?)

- If events are symbols in an alphabet Σ, traces are strings (or words) in Σ *
  - So, we can talk about (in)finite prefixes/suffixes of traces

What is a trace? (A definition)

Let Σ be a set of events. A Σ-**trace** (or simply a **trace** when Σ is understood or not important) is any finite sequence of events in Σ, that is, an element in Σ^*. If event $e \in \Sigma$ appears in trace $w \in \Sigma^*$ then we write $e \in w$.[3]

[3] Rosu and Chen, Semantics and Algorithms for Parametric Monitoring, LMCS 2012

Example 1: events and traces

Consider a resource (e.g., a synchronization object) that can be acquired and released during the lifetime of a procedure (i.e., between when the procedure begins and when it ends).

1. *What events do we care about?*

2. *What is an example trace over events in 1?*

“Good” and “bad” traces

- From example 1, are these good or bad traces:
  - begin acquire release end
  - begin acquire acquire release end
  - begin acquire acquire release release end

- Properties formalize notion of “good” or “bad” traces

- Intuition: traces validate or violate a property depending on how the property is specified

What is a property?

- A property is a set of traces
  - may include “good” traces and exclude “bad” traces
  - or, it may exclude “good” traces and include “bad” traces

- Alternately, a property is a language of acceptable or unacceptable traces (a subset of Σ^*).

- In practice, can you think of why set/language inclusion/exclusion may be insufficient for RV?

Are these definitions sufficient?

- If “good” properties in example 1 are those in which an acquired resource is released before the procedure ends. Are these “good” or “bad” traces?
  - begin acquire release acquire end
  - begin acquire acquire

- Partial traces may be in “don’t know” category
  - future events may lead to including/excluding the trace
- We need to build on the idea of partitioning traces into categories

Properties: another definition

A monitor realizes P

An Σ-**property** P (or simply a **property**) is a function $P : \Sigma^* \to C$ partitioning the set of traces into (verdict) categories C.

Partial or total function?

- This definition is a better basis for monitoring
  - C can be any set, e.g., {validating, violating, don't-know}
  - C is chosen depending on the specification language and the property being specified

Properties partition sets of traces (1)

- Let regular expressions (RE) be the spec language and choose C = {match, fail, dont-know}

- Then an RE, E, specifies property P_E, defined as:
  - **$P_E(w)$ = match** iff w is in the language of E
  - **$P_E(w)$ = fail** iff $\nexists\, w' \in \Sigma^*$ s.t. ww' is in the language of E
  - **$P_E(w)$ = dont-know** otherwise

- This is the semantics of monitoring RE in JavaMOP

Examples: CSC-related traces

- CSC specifies “bad” traces as a regular expression:
  - (sync asyncCreateIter) | (sync syncCreateIter accessIter)

- One matching trace:
  - sync asyncCreateIter accessIter accessIter accessIter accessIter

- Another matching trace:
  - sync syncCreateIter accessIter accessIter accessIter accessIter accessIter

Properties: other things to know

- Can all interesting system behavior be defined as "sets of traces"?
  - No. Hyperproperties[5] are "sets of sets of traces".

- Properties are sometimes called "trace properties"
  - In contrast with "state properties", which are defined in terms of program values at a point in an execution
  - xUnit Assertions are examples of "state properties"

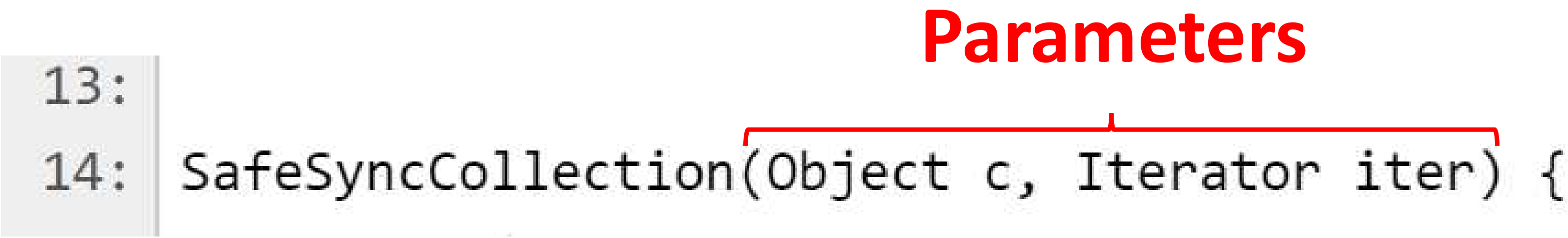
[5] Clarkson and Schneider, Hyperproperties, CSF 2008

Questions about traces/properties?

?

Recall: events can be parametric

- Events in real programs occur on different “objects”



Parameters

```

SafeSyncCollection(Object c, Iterator iter) {
```

- RV tools must be able to handle parametricity to correctly partition traces at runtime
  - Let’s look at an example

Acquire/release revisited

- Property: procedures must release acquired resources

- Spec: (begin(ϵ|(acquire(acquire|release)*release))end)*
  - Consecutive “acquire” or “release” events have the effect of acquiring or releasing the resource exactly once

- Categorize as a match, fail, or don’t-know (JavaMOP):

begin acquire acquire acquire release end begin acquire release end

Acquire/release revisited

- Same trace, but two different resources (r_1 and r_2):

begin‹› acquire‹r_1› acquire‹r_2› acquire‹r_1› release‹r_1› end‹›
begin‹› acquire‹r_2› release‹r_2› end‹›

- Categorize this parametric trace (JavaMOP)
  - Your answer:
  - Reason:

Monitoring a parametric trace (1)

- Intuition: split into two trace slices, one per resource

begin‹› acquire‹r_1› acquire‹r_2› acquire‹r_1› release‹r_1› end‹›
begin‹› acquire‹r_2› release‹r_2› end‹›

⬇

begin‹› acquire‹r_1› acquire‹r_1› release‹r_1› end‹› begin‹› end‹›

&

begin‹› acquire‹r_2› end‹› begin‹› acquire‹r_2› release‹r_2› end‹›

Monitoring a parametric trace (2)

- Then, check the trace slices non-parametrically:

begin acquire acquire release end begin end

begin acquire end begin acquire release end

Parametric trace slicing

- Essential for monitoring real software

- Future discussion: definitions and algorithms for efficient trace slicing

- Defining parametric trace slicing and parametric monitoring needs definitions of
  - parametric events
  - parametric traces
  - parametric properties

What we discussed

- What is an event?
- What is a trace?
- What is a property?
- What are parametric events, traces, and properties?
- Intro to parametric trace slicing (to be continued…)

Any questions about events, traces, and parameters?

?