CS 6156

Safety Properties and Monitoring

Owolabi Legunsen

Fall 2020
Part 2 on “theory of RV”
(we’ll switch to RV practice next week)
A conversation in lecture 1

• **Owolabi**: In theory, RV can force the system to always be correct

• **Student 1**: but... doesn’t that depend on how “correctness” is specified, i.e., bad things will never happen, or good things will eventually happen?

• **Owolabi**: 😊
Another conversation in lecture 2

• **Owolabi:** Partial traces may be in “don’t know” category. So, notions of set or language membership should be extended for RV.

• **Students 2 & 3:** Wait... are you relaxing the notion of a safety property?

• **Owolabi:** 😊
The goals in this lecture

• Formalize the intuition of “correctness” from the previous classes and reading

• Provide a framework for answering similar questions in the future

• Such formalization and framework are important for a deeper understanding of RV
What we’ll do in this lecture

• A synopsis of the paper in reading 2

Scientific Annals of Computer Science vol. 22 (2), 2012, pp. 327–365
doi: 10.7561/SACS.2012.2.327

On Safety Properties and Their Monitoring

Grigore ROȘU¹

• Goal, give you the intuition you’ll need to read it on your own (if interested)
What kinds of correctness properties have you heard about?

• Safety properties
• Liveness properties
• Hyperproperties
• Fairness
Give examples of these properties?

race freedom ⇒ safety

eventual consistency ⇒ liveness

fairness ⇒ ?

equivalence ⇒ hyperproperty
Q1: Which of these kinds of correctness properties can RV check?

• Your answer: Safety

• Why?

finite no of events
Intuition: what is a safety property?

• Your answers:

bad thing will not happen
Recall: Properties as sets of traces

• In practice, traces are always finite

• In theory, traces can be infinite, e.g., the ideal reactive system

• Traces are strings over $\Sigma^*$, so we can talk about their prefixes
Def 1: safety property

• A safety property is a prefix-closed set of “good” finite traces. Let the set of all such finite-trace prefix-closed properties be $\text{Safety}^*$

• Recall: $L$ is prefix-closed if for all prefixes $u$ of $w$, $w \in L \rightarrow u \in L$. Let $P \in \text{Safety}^*$
  • If $\neg P(w)$, then $\nexists u$ s. t. $P(wu)$, where $w, u \in \Sigma^*$
  • Equivalently, if $P(wu)$ then $P(w)$
Implication of Def 1

• Once a bad event occurs, the resulting trace can never belong to \textit{Safety}* in the future

• So, as soon as a “bad” event is concatenated with a trace that is in \textit{Safety}*, RV can report a violation and stop checking
Illustrating Def 1 (1)

• Safety property: a one-time-access key issued to a client can be activated, then used at most once, then closed

• Prefix-closed set: \{ \epsilon, activate, activate close, activate use, activate use close \}

• Any trace that is not in this prefix-closed set is a violation of the safety property
Illustrating Def 1 (2)

• Safety property: a one-time-access key issued to a client can be activated, then used multiple times, then closed

• The prefix-closed set now has infinitely many finite traces: \(\{\epsilon\} \cup \{activate\}\cdot\{use^n \mid n \in \mathbb{N}\}\cdot\{\epsilon, close\}\)

• RV can still detect traces that are not in this set, e.g., \(\{activate activate, activate use close use, \ldots\}\)
Is prefix closedness sufficient?

- **Safety** \(^*\) contains the safety properties \(\{}\) and all prefix-closed *finite* set of traces

- Any reactive system will eventually violate such safety properties even if no “bad” event occurs (reactive systems should run “forever”)

- So, we need more than prefix-closedness
Def 2: persistent safety properties

• We need prefix closedness, but we also want (reactive) systems to be able to progress safely

• \textbf{PersistentSafety}^\star \text{ is the set of all safety properties that allow a system in a safe state to continue execution onto the next safe state}

\[
\text{PersistentSafety}^\star = \{ P \in \text{Safety}^\star \mid P(w) \rightarrow \exists a \text{ s.t. } P(wa) \}
\]
More on PersistentSafety*

- **PersistentSafety*** gives a means of thinking about infinite behaviors in terms of finite traces

- \( \forall P \in \text{Safety}^* \ \exists P^\circ \in \text{PersistentSafety}^* \) s.t. \( P^\circ \) is the largest persistent safety property in \( P \)

- \( |\text{Safety}^*| = |\text{PersistentSafety}^*| = c \)

- See paper for more details and proofs
Any questions so far?
Zoom break

• 3 minutes
Problems withDefs 1 & 2?

• Another view of safety: a “bad” infinite trace must have a finite “bad” prefix

• Safety* and PersistentSafety* seem not to say anything about “bad” infinite traces

• Is there a relationship between this view, Safety*, and PersistentSafety*?
Def 3: safety properties on $\infty$ traces

• Let $\textbf{Safety}^\omega$ be the set of infinite trace properties $Q \in \mathcal{P}(\Sigma^\omega)$ s.t. if $u \notin Q$ then there is a finite trace $w \in \text{prefixes}(u)$ s.t. $wv \notin \text{for any } v \in \Sigma^\omega$.

• Probably the most common definition of safety\(^1\)

• $\textbf{Safety}^\omega$ and $\textbf{Safety}^\star$ agree (see proof in the paper):
  $|\textbf{PersistentSafety}^\star| = |\textbf{Safety}^\omega| = c$

\(^1\)Alpern and Schneider, Defining Liveness, IPL 1985
Notice a common theme?

- **Safety**: the sequence of past events in a “good” trace must be in the property

- **PersistentSafety**: to proceed to a new safe state, the sequence of past events must have been safe

- **Safety**: an infinite trace becomes “bad” after a finite sequence of past events
“Always past” characterization

• A safety property as an arbitrary (not necessarily prefix-closed) property on finite traces s.t. all finite prefixes of “good” traces are in the property

• Bijection to Safety* and Safetyω (proof in paper)
  • any safety property can be expressed as “always past”

• Connects very nicely with past-time LTL
  • one reason why LTL is a popular spec language in RV
We saw an example before...

• Property: keys must be authenticated before use
• LTL spec: $\forall k. \Box (\text{use} \rightarrow \Diamond \text{authenticate})$
• “always (b implies eventually in the past a)”
• $\Box(b \rightarrow \Diamond a)$ compactly represents this set:

$\{wsw's' \mid w, w' \in \Sigma^*, s, s' \in \Sigma, a(s) \text{ and } b(s') \text{ hold}\} \cup \{ws \mid w \in \Sigma^*, s \in \Sigma, b(s) \text{ does not hold}\}$

Why not always $\text{ptLTL}$?
There are more notions of safety

• The paper discusses at least two other notions that we omit in this lecture

• They all refer to the same set of safety properties, even though they are expressed differently
Why go through all the math?

• **1**: “something bad will not happen”

• **2**: “always in the past, something bad did not happen”

• Math showed a bijection between 1 & 2

• RV can check properties expressed as 2, but not 1
Revisiting lecture 2 conversation

• **Owolabi:** Partial traces may be in “don’t know” category. So, notions of set or language membership should be extended for RV.

• **Students 2 & 3:** Wait... are you relaxing the notion of a safety property?

• **Owolabi:** No, we are expressing safety properties in a checkable way that has a bijection to other notions of safety properties
Monitoring safety properties

• Checking safety properties as sets of traces is hard
  • Those sets can contain infinitely many traces
  • Analyzing those sets can be inconvenient

• We need to specify safety properties in formalisms that are easier to represent and reason about
Recall definition from lecture 2

• A $\Sigma$-property is a function $P : \Sigma^* \rightarrow C$ partitioning the set of traces into (verdict) categories $C$

• RV operationalizes $P$ through a monitor
Def 4: What is a monitor?

• A Σ-monitor is a triple $\mathcal{M} = (S, s_0, M : S \times \Sigma \rightarrow S)$, where $S$ is a set of events, $s_0$ is the initial event and $M$ is a deterministic partial transition function.

• Notes:
  • No final state, allows checking reactive systems
  • $\mathcal{M}$ is driven by events generated by the observed system
  • Each event drives the monitor from one state to another
  • If $M$ is undefined for the current state and current event, $\mathcal{M}$ declares a violation
Why is Def 4 important?

• A property is monitorable if it can be specified as a monitor

• All safety properties can be specified by their monitor (see paper for proof)
  • But transition function M may be undecidable

• Synthesizing monitors from compact specifications of safety properties is critical in RV
The complexity of monitoring (1)

• Let $P$ be a safety property

• The complexity of monitoring $P$ is the complexity of checking if $w \in \text{prefixes}(P)$, where $w \in \Sigma^*$

• Problem: assumes that we can always store $w$, and ignores complexity due to online monitoring
The complexity of monitoring (2)

• Let P be a safety property

• The complexity of monitoring P is a function of the size of a finite specification or representation of P

• Problems:
  • P may have different sizes in different spec languages
  • Spec of P may take more space than needed to monitor P (“every $2^n$-th event is $a$” as FSM with $2^n$ states)
The complexity of monitoring (3)

• Let P be a safety property

• Complexity of monitoring P is the functional complexity of M in a “best” $\mathcal{M} = (S, s_0, M: S \times \Sigma \rightarrow S)$

• Good: complexity of processing each event is important

• Bad: ignores the accumulating cost of M with time

Take home
Monitoring is arbitrarily hard

• Proof is in the paper

• Implication 1: P is monitorable does not always imply that monitoring P is feasible

• Implication 2: One needs carefully choose P and to design efficient monitor synthesis algorithms
Review

• Formalizations of notions of safety properties and their consensus

• “Always past” characterization allows us to express safety properties in ways that we can check

• Monitoring safety properties is arbitrarily hard
Next class

• Instrumentation (how to observe events)
  • There will likely be live coding in class

• Reading(s) will be released soon
Next week...

<table>
<thead>
<tr>
<th>Milestone</th>
<th>When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discuss some concrete project topics in class</td>
<td>By 9/17</td>
</tr>
<tr>
<td>Meet Owolabi to discuss your project proposal*</td>
<td>Before 10/5</td>
</tr>
<tr>
<td>Project proposal is due (up to 1 page)</td>
<td>10/6</td>
</tr>
<tr>
<td>Meet Owolabi to discuss project progress*</td>
<td>Before 10/26</td>
</tr>
<tr>
<td>Project progress report 1 is due (up to 2 pages)</td>
<td>10/27</td>
</tr>
<tr>
<td>Meet Owolabi to discuss project progress*</td>
<td>Before 11/18</td>
</tr>
<tr>
<td>Project progress report is due (up to 2 pages)</td>
<td>11/19</td>
</tr>
<tr>
<td>Present final project in class</td>
<td>TBD</td>
</tr>
<tr>
<td>Final project report is due</td>
<td>12/17</td>
</tr>
</tbody>
</table>

* These meetings are mandatory
Also next week...

- Assign paper presentations
  - Modalities will be shared on Piazza