Quantified Event Automata
Toward Efficient and Expressive Monitors

Giles Reger

in collaboration with

Howard Barringer, Yliès Falcone,
Klaus Havelund, David Rydeheard

August 29th, 2012
Outline

The Problem

Our Approach

Quantified Event Automata

Monitoring At Runtime
The Problem

Parametric Runtime Monitoring Problem
Checking at runtime whether a system satisfies a parametric property.

Requires

- An expressive formalism for describing parametric properties
- An efficient algorithm for checking these hold at runtime
Context

Previous approaches have focussed on

- **Efficiency**
  - JAVA MOP
  - TRACE MATCHES

- **Expressiveness**
  - EAGLE
  - RULE R
  - LOGSCOPE
  - TRACE CONTRACT
Context

Previous approaches have focussed on

- **Efficiency**
  - JAVA MOP
  - TRACEMATCHES

- **Expressiveness** *(our previous work)*
  - EAGLE
  - RULER
  - LOGSCOPE - used in the recent Mars rover mission
  - TRACE CONTRACT - used in two other NASA missions
Context

Previous approaches have focussed on

- Efficiency
  - JavaMOP
  - Tracematches

- Expressiveness *(our previous work)*
  - Eagle
  - Ruler
  - Logscope - used in the recent Mars rover mission
  - TraceContract - used in two other NASA missions

There is a need for expressive approaches.
Parametric Properties

- An event consists of a name and a list of data values
Parametric Properties

- An event consists of a name and a list of data values:
  
  open(log.txt)
Parametric Properties

- An *event* consists of a name and a list of data values
  
  \[ \text{open}(\text{log.txt}) \]

- A *trace* is a finite sequence of events
Parametric Properties

- An *event* consists of a name and a list of data values
  \[ \text{open(log.txt)} \]

- A *trace* is a finite sequence of events
  \[ \text{open(log.txt).open(out.csv).edit(log.txt).close(log.txt)} \]
Parametric Properties

- An event consists of a name and a list of data values
  \[ \text{open(\text{log.txt})} \]

- A trace is a finite sequence of events
  \[ \text{open(\text{log.txt}).open(\text{out.csv}).edit(\text{log.txt}).close(\text{log.txt})} \]

- A parametric property defines a (possibly infinite) set of traces
Parametric Properties

- An *event* consists of a name and a list of data values
  
  \[
  \text{open(\text{log.txt})}
  \]

- A *trace* is a finite sequence of events
  
  \[
  \text{open(\text{log.txt}).open(\text{out.csv}).edit(\text{log.txt}).close(\text{log.txt})}
  \]

- A *parametric property* defines a (possibly infinite) set of traces
  
  \[
  \{ \\
  \quad \text{open(\text{log.txt}).close(\text{log.txt})}, \\
  \quad \text{open(\text{log.txt}).edit(\text{log.txt}).save(\text{log.txt}).close(\text{log.txt})}, \\
  \quad \text{open(\text{log.txt}).open(\text{out.csv}).close(\text{log.txt}).close(\text{out.csv})}, \\
  \quad \ldots
  \}
  \]
Runtime Monitoring Setup

**Instrument** the system to observe a trace of relevant events.

```
property

monitor

observe

feedback

instrumentation

system
```

Verdict
Runtime Monitoring Setup

The monitor uses the given property . . .
Runtime Monitoring Setup

...to process each event...
Runtime Monitoring Setup

...possibly providing feedback to the system ...
...and finally computing a **verdict** - did the system pass?
<table>
<thead>
<tr>
<th>The Problem</th>
<th>Our Approach</th>
<th>Quantified Event Automata</th>
<th>Monitoring At Runtime</th>
</tr>
</thead>
</table>

**Outline**

**The Problem**

**Our Approach**

**Quantified Event Automata**

**Monitoring At Runtime**
Our Approach

- Describe a parametric property for a specific set of values with Event Automata (EA)
- Generalise these by replacing these values with quantified variables with Quantified Event Automata (QEA)
- QEA describe a family of EA - based on the domains of the quantified variables
Our Approach: Event Automata

- Describe a parametric property with **Event Automata**
- Alphabet of **symbolic events**
  - An event name and a list of data values or variables
- Transitions labelled with
  - symbolic events
  - guards
  - assignments
- Configurations contain **local state** (bindings)
- Automata model easy to manipulate at runtime
Specific File Usage Example

**Property : Specific File Usage**

The file “log.txt” must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.
Specific File Usage Example

Property: Specific File Usage

The file “log.txt” must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.
Specific File Usage Example

Property: Specific File Usage

The file “log.txt” must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
File Usage Example

Property : File Usage

Any file $f$ must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.
Property: File Usage

Any file $f$ must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.
Property: File Usage

Any file $f$ must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.

∀$f$

1. open($f$) → 2. edit($f$) → 3. close($f$)
2. close($f$) → 1. open($f$) → 2. edit($f$) → 3. save($f$)
3. save($f$) → 1. open($f$) → 2. edit($f$) → 3. close($f$)

F

edit($f$), save($f$), close($f$)
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- Given trace

\texttt{open(log.txt).open(out.csv).edit(log.txt).close(log.txt).close(out.csv)}

and alphabet

\{\texttt{open}(f), \texttt{edit}(f), \texttt{close}(f), \texttt{save}(f)\}

we get domain

\[ f \mapsto \{\text{log.txt, out.csv}\} \]
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant bindings
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events \( A \)
- Quantify over some of these variables used in \( A \)
- For a given trace \( \tau \)
- The domain of each variable is given by \( \tau \) and \( A \)
- This gives us a set of relevant bindings
- Here

\[
[f \mapsto \text{log.txt}], \ [f \mapsto \text{out.csv}]
\]
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant bindings
- Here
  \[
  [f \mapsto \text{log.txt}], \quad [f \mapsto \text{out.csv}]
  \]

- For each binding $\theta$
  - Let $E(\theta)$ be the Event Automaton instantiated with $\theta$
  - Let $\tau \downarrow_\theta$ be the trace projected with respect to $\theta$
  - Check if $\tau \downarrow_\theta$ is in the language of $E(\theta)$
For Each Binding

For Each Binding


[f \mapsto out.csv]
For Each Binding

\[ f \mapsto \text{out.csv} \]


1.

\[
\begin{array}{c}
\text{open(out.csv)} \\
\text{edit(out.csv)} \\
\text{close(out.csv)} \\
\text{save(out.csv)} \\
\text{open(out.csv)} \\
\text{close(out.csv)} \\
\text{edit(out.csv)} \\
\end{array}
\]

2.

3.

F

1.

\[
\begin{array}{c}
\text{edit(out.csv)}, \\
\text{save(out.csv)}, \\
\text{close(out.csv)} \\
\end{array}
\]
For Each Binding


\[ f \mapsto \text{out.csv} \quad \mapsto \]

\[ \begin{align*}
1 & \quad 2 & \quad 3 \\
\text{open(out.csv)} & \quad \text{edit(out.csv)} & \\
\text{close(out.csv)} & \quad \text{save(out.csv)} & \\
\text{edit(out.csv)}, \text{save(out.csv)}, \text{close(out.csv)} & & \text{close(out.csv)}
\end{align*} \]
For Each Binding

$$\text{open}(\text{log.txt}).\text{open}(\text{out.csv}).\text{edit}(\text{log.txt}).\text{close}(\text{log.txt}).\text{close}(\text{out.csv})$$

$$[f \leftrightarrow \text{out.csv}] \iff$$

\[1\]

\[2\]

\[3\]
For Each Binding


\[ f \mapsto \text{open(out.csv)} \]

For Each Binding


\[ f \mapsto \text{open(out.csv)} \]
For Each Binding

\[ f \mapsto \text{out.csv} \quad \mapsto \quad \text{open(out.csv)} \]
For Each Binding

open(log.txt).open(out.csv).edit(log.txt).\textbf{close(log.txt)}.close(out.csv)

$[f \mapsto \text{out.csv}] \mapsto \text{open(out.csv)}$
For Each Binding


\[ f \mapsto \text{out.csv} \quad \mapsto \quad \text{open(out.csv)}.\text{close(out.csv)} \]
For Each Binding

\[
\text{open}(\text{log.txt}).\text{open}(\text{out.csv}).\text{edit}(\text{log.txt}).\text{close}(\text{log.txt}).\text{close}(\text{out.csv})
\]

\[
[f \mapsto \text{out.csv}] \implies \text{open}(\text{out.csv}).\text{close}(\text{out.csv})
\]
For Each Binding

open(log.txt) . open(out.csv) . edit(log.txt) . close(log.txt) . close(out.csv)

\[ f \leftrightarrow \text{out.csv} \quad \mapsto \quad \text{open(out.csv)} . \text{close(out.csv)} \]
For Each Binding


\[ f \mapsto \text{out.csv} \quad \mapsto \quad \text{open(out.csv).close(out.csv)} \]
For Each Binding

\[
\text{open(log.txt).open(out.csv).edit(log.txt).close(log.txt).close(out.csv)}
\]

\[
[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(out.csv).close(out.csv)}
\]

\[
\text{open(out.csv)} \quad \text{edit(out.csv)}
\]

1 \quad 2 \quad 3

\[
\text{close(out.csv)} \quad \text{save(out.csv)}
\]

\[
\text{edit(out.csv), save(out.csv), close(out.csv)} \quad \text{close(out.csv)}
\]
For Each Binding

\[ f \mapsto \text{out.csv} \quad \mapsto \quad \text{open(out.csv).close(out.csv)} \quad \checkmark \]
\[ f \mapsto \text{log.txt} \]
For Each Binding


[f \mapsto \text{out.csv}] \implies \text{open(out.csv).close(out.csv)} \quad \checkmark

[f \mapsto \text{log.txt}]
For Each Binding

\[
\text{open(log.txt).open(out.csv).edit(log.txt).close(log.txt).close(out.csv)}
\]

\[
[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(out.csv).close(out.csv)} \quad \checkmark
\]

\[
[f \mapsto \text{log.txt}] \quad \mapsto
\]
For Each Binding

\[
\text{open(log.txt)} \cdot \text{open(out.csv)} \cdot \text{edit(log.txt)} \cdot \text{close(log.txt)} \cdot \text{close(out.csv)}
\]

\[
[f \mapsto \text{out.csv}] \implies \text{open(out.csv)} \cdot \text{close(out.csv)} \quad \checkmark
\]

\[
[f \mapsto \text{log.txt}] \implies \text{open(log.txt)}
\]
For Each Binding

\[
\text{open(} \log.txt \text{).open(} \text{out.csv} \text{).edit(} \log.txt \text{).close(} \log.txt \text{).close(} \text{out.csv} \text{)}
\]

\[
[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(} \text{out.csv} \text{).close(} \text{out.csv} \text{)} \quad \checkmark
\]

\[
[f \mapsto \log.txt] \quad \mapsto \quad \text{open(} \log.txt \text{)}
\]
For Each Binding


\[ f \mapsto \text{out.csv} \quad \mapsto \quad \text{open(out.csv).close(out.csv)} \quad \checkmark \]

\[ f \mapsto \text{log.txt} \quad \mapsto \quad \text{open(log.txt).edit(log.txt)} \]

---

**Diagram**

1. open(log.txt) → 2. edit(log.txt) → 3. close(log.txt) → F

- **1. open(log.txt)**
  - close(log.txt) → 2.
  - edit(log.txt) → 2.
  - save(log.txt) → 2.
  - open(log.txt) → 2.

- **2. edit(log.txt)**
  - save(log.txt) → 3.
  - open(log.txt) → 3.

- **3. close(log.txt)**
  - open(log.txt) → 1.
  - close(log.txt) → 1.

- **F**
  - edit(log.txt), save(log.txt), close(log.txt) → 1.
For Each Binding

\[
\text{open(log.txt).open(out.csv).edit(log.txt).} \text{close(log.txt).close(out.csv)}
\]

\[
[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(out.csv).close(out.csv)} \quad \checkmark
\]

\[
[f \mapsto \text{log.txt}] \quad \mapsto \quad \text{open(log.txt).edit(log.txt).} \text{close(log.txt)}
\]
For Each Binding


[f → out.csv]  ⇒  open(out.csv).close(out.csv)
[f → log.txt]  ⇒  open(log.txt).edit(log.txt).close(log.txt)
For Each Binding


\[ f \mapsto out.csv \quad \mapsto \quad open(out.csv).close(out.csv) \quad \checkmark \]

\[ f \mapsto log.txt \quad \mapsto \quad open(log.txt).edit(log.txt).close(log.txt) \]

![Diagram of event automata](image.png)
For Each Binding


\[ f \mapsto out.csv \] \implies open(out.csv).close(out.csv) \quad \checkmark

\[ f \mapsto log.txt \] \implies open(log.txt).edit(log.txt).close(log.txt)
For Each Binding

\[
\begin{align*}
\text{open}(\text{log.txt}).\text{open}(\text{out.csv}).\text{edit}(\text{log.txt}).\text{close}(\text{log.txt}).\text{close}(\text{out.csv}) \\
[f \mapsto \text{out.csv}] & \mapsto \text{open}(\text{out.csv}).\text{close}(\text{out.csv}) & \checkmark \\
[f \mapsto \text{log.txt}] & \mapsto \text{open}(\text{log.txt}).\text{edit}(\text{log.txt}).\text{close}(\text{log.txt})
\end{align*}
\]
For Each Binding


\[f \mapsto \text{out.csv}] \implies \text{open(out.csv).close(out.csv)} \quad \checkmark

\[f \mapsto \text{log.txt}] \implies \text{open(log.txt).edit(log.txt).close(log.txt)}
For Each Binding


[f \mapsto \text{out.csv}] \implies \text{open(out.csv).close(out.csv)} \quad \checkmark

[f \mapsto \text{log.txt}] \implies \text{open(log.txt).edit(log.txt).close(log.txt)} \quad \times
For Each Binding


\[ f \mapsto \text{out.csv} \implies \text{open(out.csv).close(out.csv)} \quad \checkmark \]
\[ f \mapsto \text{log.txt} \implies \text{open(log.txt).edit(log.txt).close(log.txt)} \quad \times \]
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- **Quantify** over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant **bindings**
- For each binding $\theta$
  - Let $E(\theta)$ be the Event Automaton instantiated with $\theta$
  - Let $\tau \downarrow_{\theta}$ be the trace projected with respect to $\theta$
  - Check if $\tau \downarrow_{\theta}$ is in the language of $E(\theta)$
- We then use these results to check the quantifications
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding $\theta$
  - Let $E(\theta)$ be the Event Automaton instantiated with $\theta$
  - Let $\tau \downarrow \theta$ be the trace projected with respect to $\theta$
  - Check if $\tau \downarrow \theta$ is in the language of $E(\theta)$
- We then use these results to check the quantifications
- In our example $\forall f$ means that we need $\tau \downarrow \theta$ in the language of $E(\theta)$ for all bindings $\theta$ that bind $f$
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- **Quantify** over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
  - The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding $\theta$
  - Let $E(\theta)$ be the Event Automaton instantiated with $\theta$
  - Let $\tau \downarrow \theta$ be the trace projected with respect to $\theta$
  - Check if $\tau \downarrow \theta$ is in the language of $E(\theta)$
- We then use these results to check the quantifications
- In our example $\forall f$ means that we need $\tau \downarrow \theta$ in the language of $E(\theta)$ for all bindings $\theta$ that bind $f$

$$
[f \mapsto \text{out.csv}] \quad \rightarrow \quad \text{open(out.csv).close(out.csv)} \quad \checkmark
$$
$$
[f \mapsto \text{log.txt}] \quad \rightarrow \quad \text{open(log.txt).edit(log.txt).close(log.txt)} \quad \times
$$
Our Approach: Quantified Event Automata

- Define an Event Automata over a set of symbolic events $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace $\tau$
- The domain of each variable is given by $\tau$ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding $\theta$
  - Let $E(\theta)$ be the Event Automaton instantiated with $\theta$
  - Let $\tau \downarrow \theta$ be the trace projected with respect to $\theta$
  - Check if $\tau \downarrow \theta$ is in the language of $E(\theta)$
- We then use these results to check the quantifications
- In our example $\forall f$ means that we need $\tau \downarrow \theta$ in the language of $E(\theta)$ for all bindings $\theta$ that bind $f$

[f $\mapsto$ out.csv] $\mapsto$ open(out.csv).close(out.csv) ✓

[f $\mapsto$ log.txt] $\mapsto$ open(log.txt).edit(log.txt).close(log.txt) ✗

The trace does not satisfy the property
Interpreting Quantifications

- If the quantification is all universal i.e. for $\forall x, \forall y$ we need $\tau \downarrow_\theta$ in the language of $E(\theta)$ for all bindings $\theta$ i.e. for all values in the domains of $x$ and $y$

- Existential quantification is treated as expected
  - Given $\forall x, \exists y$ we must find a binding $\theta = [x \mapsto v_x, y \mapsto v_y]$ for each value $v_x$ in the domain of $x$ such that $\tau \downarrow_\theta$ is in the language of $E(\theta)$
  - If all quantifications are existential we must find at least one binding $\theta$ such that $\tau \downarrow_\theta$ is in the language of $E(\theta)$

- Note that these bindings are given by the domains of the quantified variables, which are dependent on the trace
Outline

The Problem

Our Approach

Quantified Event Automata

Monitoring At Runtime
Quantified Event Automata

Definition (Event Automaton)

An Event Automaton $\langle Q, A, \delta, q_0, F \rangle$ is a tuple where

- $Q$ is a set of states,
- $A \subseteq SymbolicEvent$ is a alphabet of events,
- $\delta \subseteq (Q \times A \times Guard \times Assign \times Q)$ is a set of transitions,
- $q_0$ is an initial state, and
- $F \subseteq Q$ is a set of final states.

Definition (Quantified Event Automaton)

A QEA is a pair $\langle \Lambda, E \rangle$ where

- $\Lambda \in (\{\forall, \exists\} \times \text{variables}(E) \times Guard)^*$ is a list of quantified variables with guards, and
- $E$ is an Event Automaton.
Free Variables

• Some variables in the Event Automaton may not be quantified
• These are called free variables
• Free variables are (re)bound as the trace is processed
• Allowing us to capture changing data values
Property: Auction Bidding

Amounts bid for an item should be strictly increasing.

∀item

1

bid(item, max)

2

bid(item, new)

3

\[
\text{bid}(item, \text{new}) \quad \frac{\text{new} \leq \text{max}}{\text{max} := \text{new}}
\]

\[
\text{bid}(item, \text{new}) \quad \frac{\text{new} > \text{max}}{\text{max} := \text{new}}
\]
Property: Auction Bidding

Amounts bid for an item should be strictly increasing.

∀item

1 ➔ bid(item, max)

2 ➔ bid(item, new)

3 ➔ bid(item, max) \(\text{new} \leq \text{max}\)

\(\text{new} \geq \text{max}\) \(\text{max} := \text{new}\)
Bidding For A Hat

\[ \text{bid}(\text{hat}, 5). \text{bid}(\text{hat}, 10). \text{bid}(\text{hat}, 7) \]

\[
\begin{array}{c}
1 \quad \text{bid}(\text{hat}, \text{max}) \\
\quad \downarrow \\
2 \quad \text{bid}(\text{hat}, \text{new}) \quad \frac{\text{new} \leq \text{max}}{\text{new} > \text{max}} \\
\quad \uparrow \\
3 \quad \text{bid}(\text{hat}, \text{new}) \quad \frac{\text{max} := \text{new}}{}
\end{array}
\]
Bidding For A Hat

\[ \text{bid}(\text{hat}, 5).\text{bid}(\text{hat}, 10).\text{bid}(\text{hat}, 7) \]

\[ \langle 1, [ ] \rangle \]
Bidding For A Hat

\[ \text{bid}(\text{hat}, 5).\text{bid}(\text{hat}, 10).\text{bid}(\text{hat}, 7) \]

\[ \begin{array}{c}
1 \\
\downarrow \\
\text{bid}(\text{hat, max})
\end{array} \quad \begin{array}{c}
2 \\
\text{bid}(\text{hat, new})
\end{array} \quad \begin{array}{c}
3 \\
\text{bid}(\text{hat, new})
\end{array} \quad \begin{array}{c}
\text{new} \leq \text{max} \\
\text{new} > \text{max} \\
\text{max} \leftarrow \text{new}
\end{array} \]

\[ \langle 1, \{ \} \rangle \quad \text{bid}(\text{hat,5}) \quad \langle 2, \{ \text{max} \rightarrow 5 \} \rangle \]
Bidding For A Hat

\[ \text{bid}(\text{hat}, 5) \cdot \text{bid}(\text{hat}, 10) \cdot \text{bid}(\text{hat}, 7) \]

\[
\begin{array}{c}
1 \quad \text{bid}(\text{hat}, \text{max}) \quad 2 \quad \text{bid}(\text{hat}, \text{new}) \quad 3
\end{array}
\]

\[
\begin{align*}
\langle 1, [ ] \rangle & \xrightarrow{\text{bid}(\text{hat}, 5)} \langle 2, [\text{max} \mapsto 5] \rangle \\
\langle 2, [\text{new} \mapsto 10, \text{max} \mapsto 10] \rangle & \xleftarrow{\text{bid}(\text{hat}, 10)}
\end{align*}
\]
**Bidding For A Hat**

\[
\text{bid}(\text{hat}, 5). \text{bid}(\text{hat}, 10). \text{bid}(\text{hat}, 7)
\]

![Diagram showing bidding process](attachment:image.png)
Outline

The Problem

Our Approach

Quantified Event Automata

Monitoring At Runtime
The Problem

Our Approach

Quantified Event Automata

Monitoring At Runtime

Monitoring at Runtime (i.e. on the fly)

- The semantics for Quantified Event Automata are given in terms of a whole trace
- Required as we quantify over values in the whole trace
- This is inappropriate for monitoring at runtime
Monitoring at Runtime (i.e. on the fly)

- The semantics for Quantified Event Automata are given in terms of a whole trace
- Required as we quantify over values in the whole trace
- This is inappropriate for monitoring at runtime
- Solution: Develop a small-step semantics that processes the trace one event at a time
The semantics for Quantified Event Automata are given in terms of a whole trace.

Required as we quantify over values in the whole trace.

This is inappropriate for monitoring at runtime.

Solution: Develop a small-step semantics that processes the trace one event at a time.

Two semantics give equivalent verdicts at end of trace.
A Small Step Semantics

- Not all information received at once - therefore, need to build up partial bindings and partial projections
- Associate projections with bindings

\[ \theta_5 \rightarrow \theta_6 \]

- When adding a new binding use the largest (given by partial order on bindings) existing consistent binding

\[ \theta_5 \rightarrow \theta_6 \]

\[ \theta_1 \rightarrow \theta_2 \rightarrow \theta_3 \rightarrow \theta_4 \]

\[ [ ] \]
Property: Lock Ordering

Every distinct pair of locks should be taken and released in a consistent order.

∀l₁, ∀l₂ : l₁ ≠ l₂

lk(l₁)

ulk(l₁)

lk(l₂)

ulk(l₂)

lk(l₁)

lk(l₂)

ulk(l₁)

ulk(l₂)

lk = lock     ulk = unlock
Lock Ordering Example : Computing Projections

\[ l_k(A), l_k(B), u_lk(B), u_lk(A), l_k(B), l_k(A) \]
Lock Ordering Example: Computing Projections

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$ $l_2$</td>
<td></td>
<td>$l_1$ $l_2$</td>
<td></td>
</tr>
<tr>
<td>$\epsilon$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

lk(A)

\[ [l_1 \mapsto A, l_2 \mapsto A] \]

\[ [l_1 \mapsto A] \quad \quad \quad [l_2 \mapsto A] \]

\[ [\] \]

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 ) ( l_2 )</td>
<td>( \epsilon )</td>
<td>( l_1 ) ( l_2 )</td>
<td>( 1k(A) )</td>
</tr>
<tr>
<td>( A ) ( A )</td>
<td>( 1k(A) )</td>
<td>( A ) ( A )</td>
<td>( 1k(A) )</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

\[1k(A).1k(B)\]

<table>
<thead>
<tr>
<th>partial</th>
<th>projection</th>
<th>total</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>binding</td>
<td>binding</td>
<td>binding</td>
<td></td>
</tr>
<tr>
<td>[l_1 \mapsto A, l_2 \mapsto A]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[l_1 \mapsto A]</td>
<td>[l_2 \mapsto A]</td>
<td>[l_1 \mapsto B]</td>
<td>[l_2 \mapsto B]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>(A)</td>
<td>(A)</td>
<td>1k(A)</td>
</tr>
<tr>
<td>(B)</td>
<td>1k(B)</td>
<td>(B)</td>
<td>1k(B)</td>
</tr>
<tr>
<td>(l_1 \mapsto A)</td>
<td>1k(A)</td>
<td>(l_1 \mapsto B)</td>
<td>1k(A).1k(B)</td>
</tr>
<tr>
<td>(l_2 \mapsto A)</td>
<td>1k(A)</td>
<td>(l_2 \mapsto B)</td>
<td>1k(A).1k(B)</td>
</tr>
<tr>
<td>(l_1 \mapsto B)</td>
<td>1k(B)</td>
<td>(l_2 \mapsto A)</td>
<td>1k(B)</td>
</tr>
<tr>
<td>(l_1 \mapsto B)</td>
<td>1k(B)</td>
<td>(l_2 \mapsto B)</td>
<td>1k(B)</td>
</tr>
</tbody>
</table>
## Lock Ordering Example: Computing Projections

\[ \text{lk}(A).\text{lk}(B).\text{ulk}(B) \]

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ l_1 \mapsto A, l_2 \mapsto A ]</td>
<td>[ \epsilon ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto A, l_2 \mapsto B ]</td>
<td>[ \text{lk}(A) ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto B, l_2 \mapsto A ]</td>
<td>[ \text{lk}(A).\text{ulk}(B) ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto B, l_2 \mapsto B ]</td>
<td>[ \text{lk}(B).\text{ulk}(B) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ l_1 \mapsto A, l_2 \mapsto A ]</td>
<td>[ \text{lk}(A) ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto A, l_2 \mapsto B ]</td>
<td>[ \text{lk}(A).\text{lk}(B).\text{ulk}(B) ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto B, l_2 \mapsto A ]</td>
<td>[ \text{lk}(A).\text{lk}(B).\text{ulk}(B) ]</td>
</tr>
<tr>
<td>[ l_1 \mapsto B, l_2 \mapsto B ]</td>
<td>[ \text{lk}(B).\text{ulk}(B) ]</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

\[ \text{l}k(A).\text{l}k(B).\text{ul}k(B).\text{ul}k(A) \]

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \mapsto A, l_2 \mapsto A )</td>
<td>( \varepsilon )</td>
<td>( l_1 \mapsto A )</td>
<td>1k(A).ulk(A)</td>
</tr>
<tr>
<td>( l_1 \mapsto A, l_2 \mapsto B )</td>
<td>1k(A).ulk(A)</td>
<td>( l_1 \mapsto B )</td>
<td>1k(A).lk(B).ulk(B).ulk(A)</td>
</tr>
<tr>
<td>( l_1 \mapsto B, l_2 \mapsto A )</td>
<td>1k(A).ulk(A)</td>
<td>( l_2 \mapsto A )</td>
<td>1k(A).lk(B).ulk(B).ulk(A)</td>
</tr>
<tr>
<td>( l_1 \mapsto B, l_2 \mapsto B )</td>
<td>1k(B).ulk(B)</td>
<td>( l_2 \mapsto B )</td>
<td>1k(B).ulk(B)</td>
</tr>
<tr>
<td>( l_1 \mapsto A )</td>
<td>1k(A).ulk(A)</td>
<td>( l_2 \mapsto A )</td>
<td>1k(A).ulk(A)</td>
</tr>
<tr>
<td>( l_1 \mapsto B )</td>
<td>1k(B).ulk(B)</td>
<td>( l_2 \mapsto B )</td>
<td>1k(B).ulk(B)</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

\[ lk(A).lk(B).ulk(B).ulk(A).lk(B) \]

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \leftrightarrow A, l_2 \leftrightarrow A )</td>
<td>[ l_1 \mapsto \epsilon ]</td>
<td>( l_1 \leftrightarrow A, l_2 \leftrightarrow B )</td>
<td>( A \rightarrow A )</td>
</tr>
<tr>
<td>( l_1 \leftrightarrow A )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( l_1 \leftrightarrow B, l_2 \leftrightarrow A )</td>
<td>( A \rightarrow B )</td>
</tr>
<tr>
<td>( l_2 \leftrightarrow A )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( l_2 \leftrightarrow B )</td>
<td>( B \rightarrow A )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( l_1 \leftrightarrow B )</td>
<td>( B \rightarrow B )</td>
</tr>
<tr>
<td>( l_1 \mapsto \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( 1k(A).1k(B).ulk(B).ulk(A).1k(B) )</td>
<td>( 1k(A).ulk(B).1k(B) )</td>
</tr>
<tr>
<td>( l_1 \mapsto \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( 1k(A).1k(B).ulk(B).ulk(A).1k(B) )</td>
<td>( 1k(A).ulk(B).1k(B) )</td>
</tr>
<tr>
<td>( l_2 \mapsto \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( 1k(A).1k(B).ulk(B).ulk(A).1k(B) )</td>
<td>( 1k(A).ulk(B).1k(B) )</td>
</tr>
<tr>
<td>( l_2 \mapsto \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( 1k(A).1k(B).ulk(B).ulk(A).1k(B) )</td>
<td>( 1k(A).ulk(B).1k(B) )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>( 1k(A).ulk(A) )</td>
<td>( 1k(A).1k(B).ulk(B).ulk(A).1k(B) )</td>
<td>( 1k(A).ulk(B).1k(B) )</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

\[ \text{l}_k(A).\text{l}_k(B).\text{u}_k(B).\text{u}_k(A).\text{l}_k(B).\text{l}_k(A) \]

<table>
<thead>
<tr>
<th>partial binding</th>
<th>projection</th>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \mapsto A, l_2 \mapsto A )</td>
<td>([ l_1 \mapsto A ] )</td>
<td>( l_1 \mapsto A )</td>
<td>( \text{l}_k(A) \cdot \text{u}_k(A) \cdot \text{l}_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto A, l_2 \mapsto B )</td>
<td>( l_2 \mapsto A )</td>
<td>( A \cdot A )</td>
<td>( \text{l}_k(A) \cdot \text{l}_k(B) \cdot \text{u}_k(B) \cdot \text{u}_k(A) \cdot \text{l}_k(B) \cdot \text{l}_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto B, l_2 \mapsto A )</td>
<td>( l_2 \mapsto B )</td>
<td>( B \cdot A )</td>
<td>( \text{l}_k(A) \cdot \text{l}_k(B) \cdot \text{u}_k(B) \cdot \text{u}_k(A) \cdot \text{l}_k(B) \cdot \text{l}_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto B, l_2 \mapsto B )</td>
<td>( l_2 \mapsto B )</td>
<td>( B \cdot B )</td>
<td>( \text{l}_k(B) \cdot \text{u}_k(B) \cdot \text{l}_k(B) )</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing Projections

\[ l_k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) \]

![Diagram showing lock ordering example]

<table>
<thead>
<tr>
<th>Partial binding</th>
<th>Projection</th>
<th>Total binding</th>
<th>Projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \mapsto A, \ l_2 \mapsto A )</td>
<td>( A )</td>
<td>( l_1 \mapsto A, \ l_2 \mapsto B )</td>
<td>( 1k(A).u_lk(A).l_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto A, \ l_2 \mapsto B )</td>
<td>( 1k(A).u_lk(A).l_k(A) )</td>
<td>( l_1 \mapsto B, \ l_2 \mapsto A )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto B, \ l_2 \mapsto B )</td>
<td>( 1k(B).u_lk(B).l_k(B) )</td>
<td>( l_1 \mapsto B )</td>
<td>( 1k(B).u_lk(B).l_k(B) )</td>
</tr>
<tr>
<td>( l_1 \mapsto A )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
<td>( l_2 \mapsto A )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto A )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
<td>( l_2 \mapsto B )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
</tr>
<tr>
<td>( l_1 \mapsto A )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
<td>( l_2 \mapsto B )</td>
<td>( 1k(A).l_k(B).u_lk(B).u_lk(A).l_k(B).l_k(A) )</td>
</tr>
</tbody>
</table>
Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$  $l_2$</td>
<td></td>
</tr>
<tr>
<td>A  A</td>
<td>lk(A).ulk(A).lk(A)</td>
</tr>
<tr>
<td>A  B</td>
<td>lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)</td>
</tr>
<tr>
<td>B  A</td>
<td>lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)</td>
</tr>
<tr>
<td>B  B</td>
<td>lk(B).ulk(B).lk(B)</td>
</tr>
</tbody>
</table>

$\forall l_1, \forall l_2 : l_1 \neq l_2$
## Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

∀$l_1, \forall l_2 : l_1 \neq l_2$

![Diagram](image-url)
### Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

$\forall l_1, \forall l_2 : l_1 \neq l_2$
## Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

$\forall l_1, \forall l_2 : l_1 \neq l_2$
Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A A</td>
<td>$lk(A).ulk(A).lk(A)$</td>
</tr>
<tr>
<td>A B</td>
<td>$lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)$</td>
</tr>
<tr>
<td>B A</td>
<td>$lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)$</td>
</tr>
<tr>
<td>B B</td>
<td>$lk(B).ulk(B).lk(B)$</td>
</tr>
</tbody>
</table>

$\forall l_1, \forall l_2 : l_1 \neq l_2$
### Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 )</td>
<td>( l_2 )</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

\[ \forall l_1, \forall l_2 : l_1 \neq l_2 \]

The trace does not satisfy the property
# Lock Ordering Example: Computing a Verdict

## Total Binding vs. Projection

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

∀$l_1, \forall l_2 : l_1 \neq l_2$

**Strong Failure State**

---

The Problem

Our Approach

Quantified Event Automata

Monitoring At Runtime
Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

$\forall l_1, \forall l_2 : l_1 \neq l_2$

No extensions of this trace can satisfy the property
## Lock Ordering Example: Computing a Verdict

<table>
<thead>
<tr>
<th>total binding</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 ) ( l_2 )</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>lk(A).ulk(A).lk(A)</td>
</tr>
<tr>
<td>A</td>
<td>lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)</td>
</tr>
<tr>
<td>B</td>
<td>lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)</td>
</tr>
<tr>
<td>B</td>
<td>lk(B).ulk(B).lk(B)</td>
</tr>
</tbody>
</table>

\( \forall l_1, \forall l_2 : l_1 \neq l_2 \)

No extensions of this trace can satisfy the property

\[ = \text{StrongFailure} \]
Practicalities

- Storing trace projections directly would be inefficient
- Instead, store configurations directly

\[
\text{Configuration} = \text{State} \times \text{Binding}
\]

\[
\text{Binding} \rightarrow \mathcal{P}(\text{Configuration})
\]

- Compute language acceptance from states reached

- We have a prototype implementation in Scala

- Can take advantage of previous work in this area
  - Indexing schemes
  - Garbage collection
Future Work

We are currently working on

- Efficient Algorithms for Runtime Monitoring
- Specification Inference targeting Quantified Event Automata
Thank you for listening

Any questions?