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

  JavaMOP
  tracematches

- Expressiveness

  Eagle
  RuleR
  Logscope
  TraceContract

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

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

  open(log.txt)

- A *trace* is a finite sequence of events

Parametric Properties

- An *event* consists of a name and a list of data values

  `open`(log.txt)

- A *trace* is a finite sequence of events

  `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

  open(log.txt)

- A *trace* is a finite sequence of events

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

  open(log.txt)

- A *trace* is a finite sequence of events

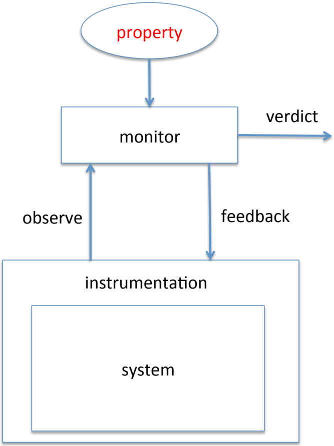
  open(log.txt).open(out.csv).edit(log.txt).close(log.txt)

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

{
open(log.txt).close(log.txt),
open(log.txt).edit(log.txt), save(log.txt), close(log.txt),
open(log.txt).open(out.csv).close(log.txt).close(out.csv),
...
}

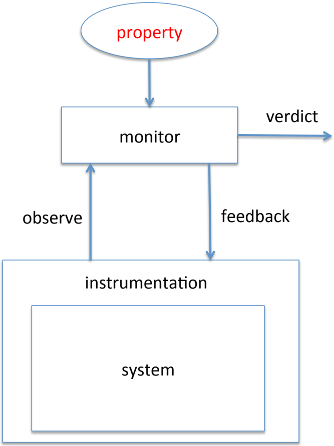
Runtime Monitoring Setup

Instrument the system to observe a trace of relevant events



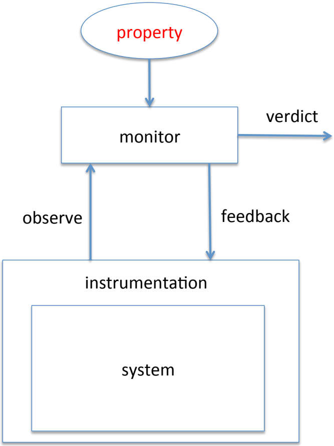
Runtime Monitoring Setup

The monitor uses the given property . . .



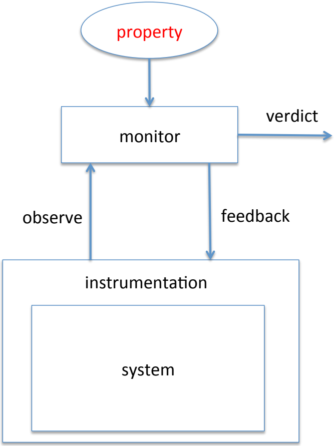
Runtime Monitoring Setup

. . . to process each event . . .



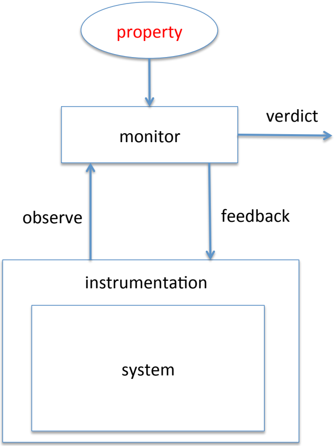
Runtime Monitoring Setup

. . . possibly providing feedback to the system . . .



Runtime Monitoring Setup

. . . and finally computing a verdict - did the system pass?



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

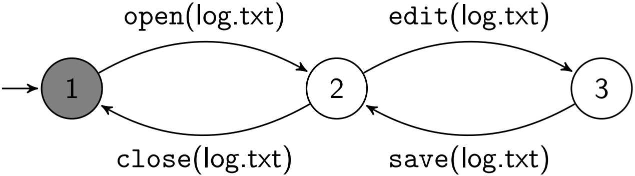
Property : Specific File Usage

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

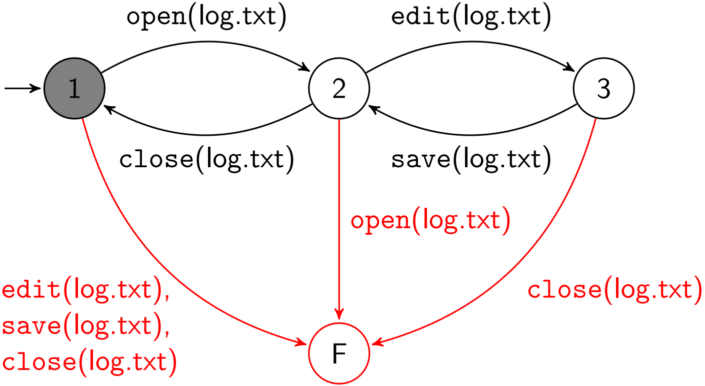
The file “log.txt” must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.



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

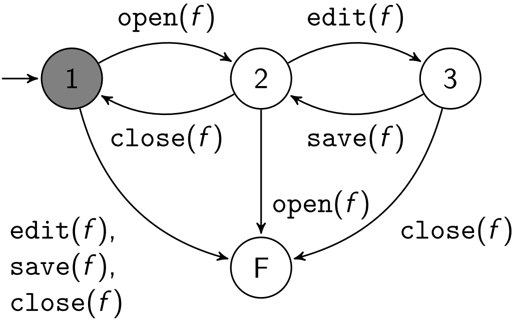
Property : File Usage

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.

File Usage Example

Property : File Usage

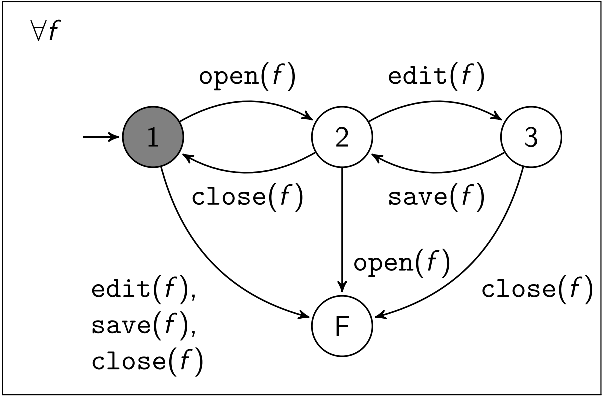
Any file f must be opened before it is used, if opened must eventually be closed and if edited must be saved before being closed.



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.



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 τ

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 τ
- The domain of each variable is given by τ 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 τ
- The domain of each variable is given by τ and $\mathcal{A}$
- Given trace

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

and alphabet

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

we get domain

$$[f \mapsto \{\mathsf{log.txt}, \mathsf{out.csv}\}]$$

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 τ
- The domain of each variable is given by τ 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 $\mathcal{A}$
- Quantify over some of these variables used in $\mathcal{A}$
- For a given trace τ
- The domain of each variable is given by τ and $\mathcal{A}$
- This gives us a set of relevant bindings
- Here

$$[f \mapsto \mathsf{log.txt}], \quad [f \mapsto \mathsf{out.csv}]$$

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 τ
- The domain of each variable is given by τ 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 θ
  - Let $\mathrm{E}(\theta)$ be the Event Automaton instantiated with θ
  - Let $\tau\downarrow_\theta$ be the trace projected with respect to θ
  - Check if $\tau\downarrow_\theta$ is in the language of $\mathrm{E}(\theta)$

For Each Binding

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

For Each Binding

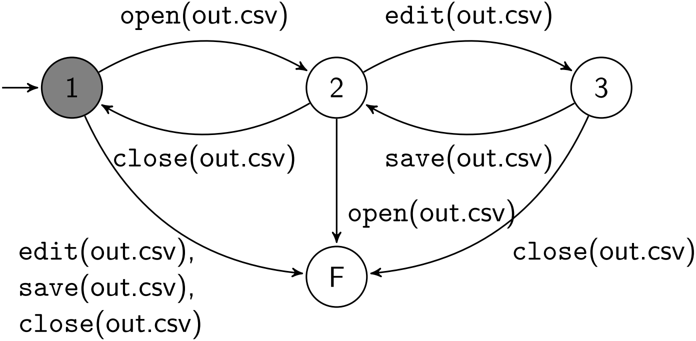
$\texttt{open}(\mathsf{log.txt}).\texttt{open}(\mathsf{out.csv}).\texttt{edit}(\mathsf{log.txt}).\texttt{close}(\mathsf{log.txt}).\texttt{close}(\mathsf{out.csv})$

$[f \mapsto \mathsf{out.csv}]$

For Each Binding

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

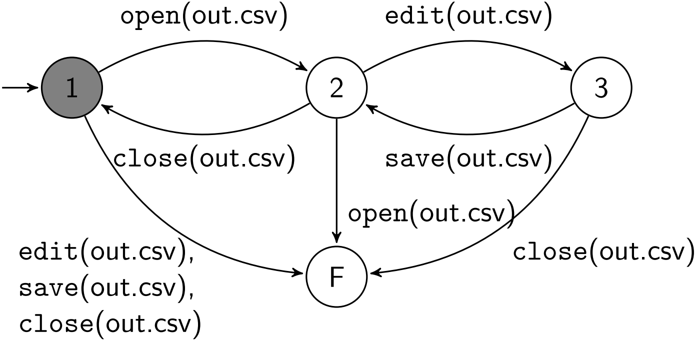
$[f \mapsto \text{out.csv}]$



For Each Binding

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

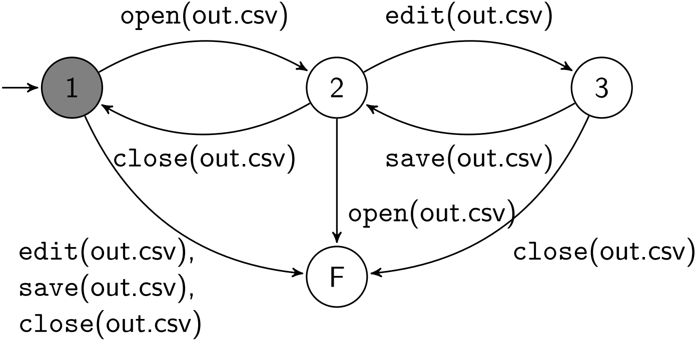
$[f \mapsto \text{out.csv}] \quad \mapsto$



For Each Binding

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

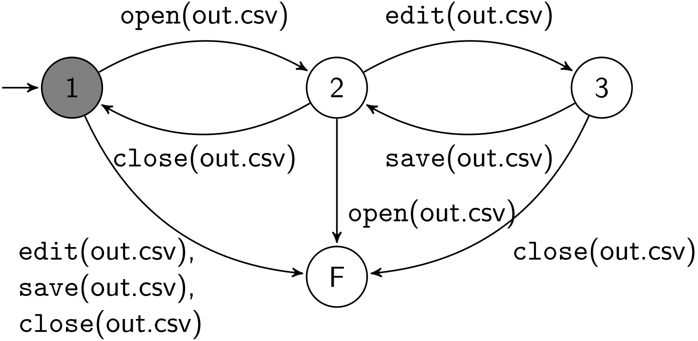
$[f \mapsto \text{out.csv}] \quad \mapsto$



For Each Binding

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

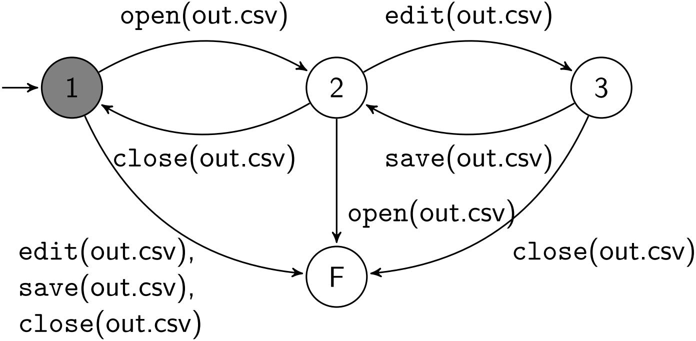
$[f \mapsto \text{out.csv}] \quad \mapsto \quad$ open(out.csv)



For Each Binding

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

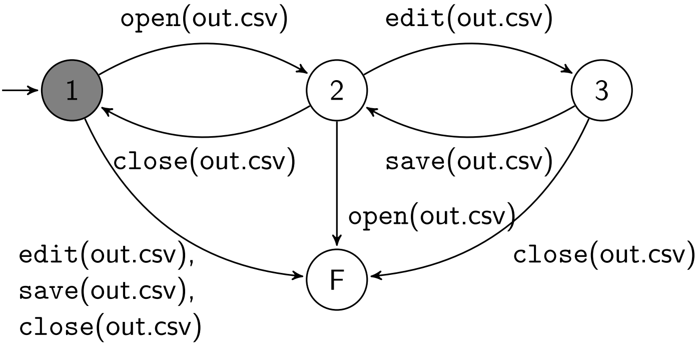
$[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(out.csv)}$



For Each Binding

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

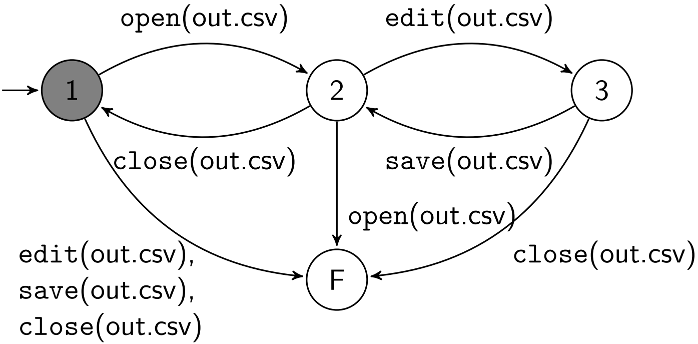
$[f \mapsto \text{out.csv}] \quad \mapsto \quad \text{open(out.csv)}$



For Each Binding

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

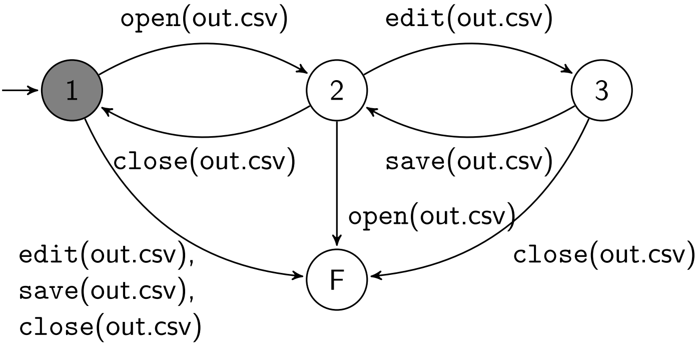
$[f \mapsto \text{out.csv}] \quad \mapsto \quad$ open(out.csv).close(out.csv)



For Each Binding

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

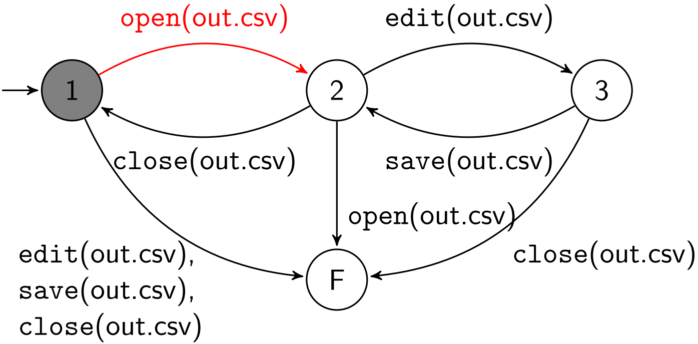
$[f \mapsto \text{out.csv}] \quad \mapsto \quad$ open(out.csv).close(out.csv)



For Each Binding

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

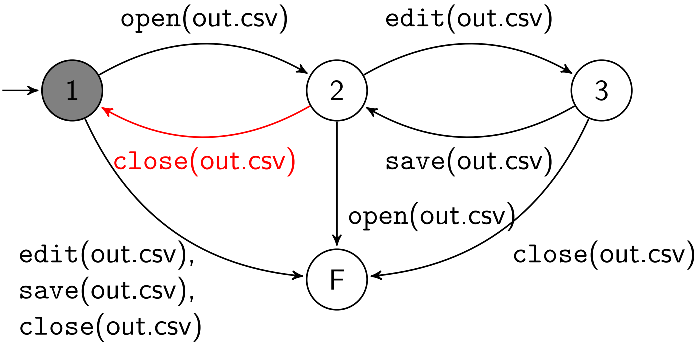
[f ↦ out.csv] ↦ open(out.csv).close(out.csv)



For Each Binding

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

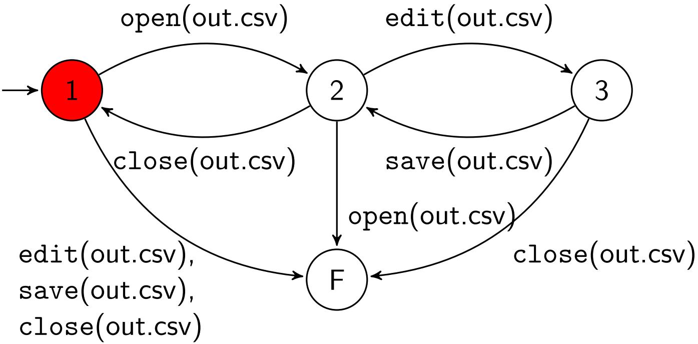
$[f \mapsto \text{out.csv}]$ ↦ open(out.csv).close(out.csv)



For Each Binding

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

$[f \mapsto \text{out.csv}]$ ↦ open(out.csv).close(out.csv) ✓



For Each Binding

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

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

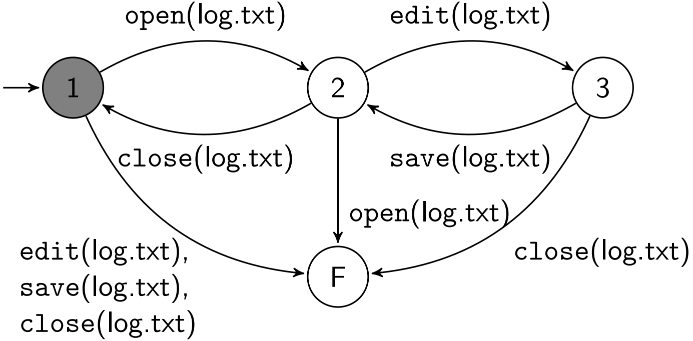
$[f \mapsto \text{log.txt}]$

For Each Binding

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

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

$[f \mapsto \text{log.txt}]$

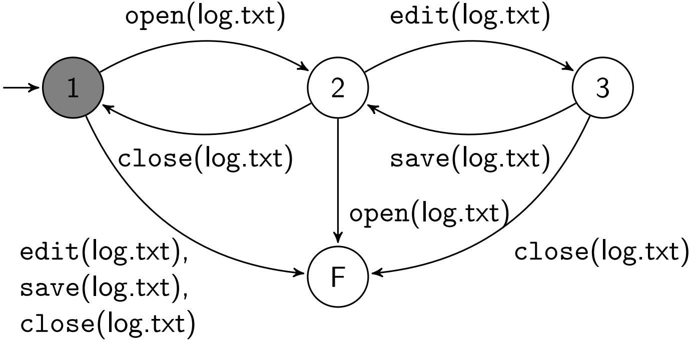


For Each Binding

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

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

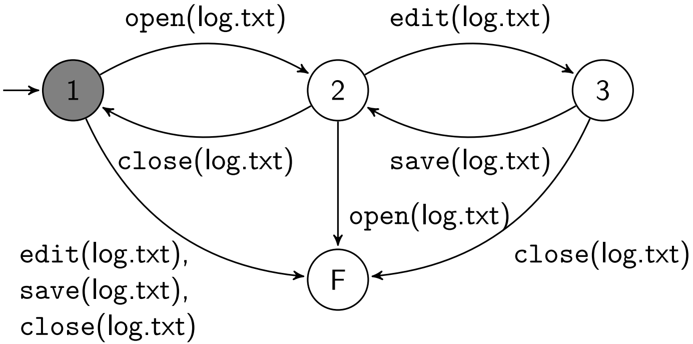
$[f \mapsto \text{log.txt}] \quad \mapsto$



For Each Binding

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

$[f \mapsto \text{out.csv}]$	$\mapsto$	open(out.csv).close(out.csv)	✓
$[f \mapsto \text{log.txt}]$	$\mapsto$	open(log.txt)	

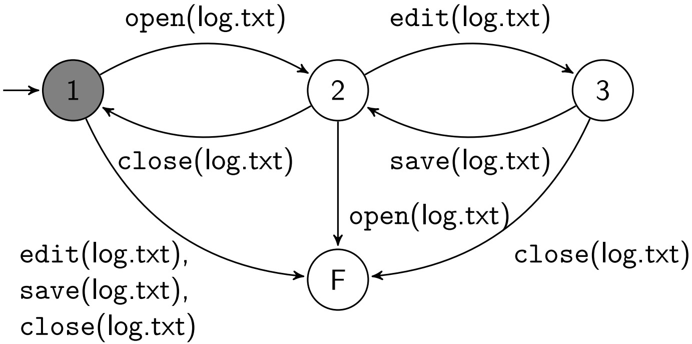


For Each Binding

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

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

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

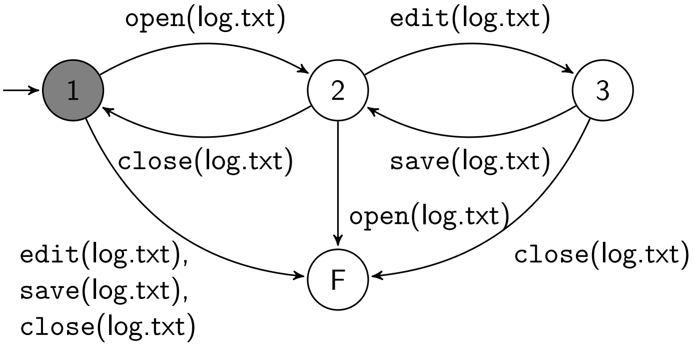


For Each Binding

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

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

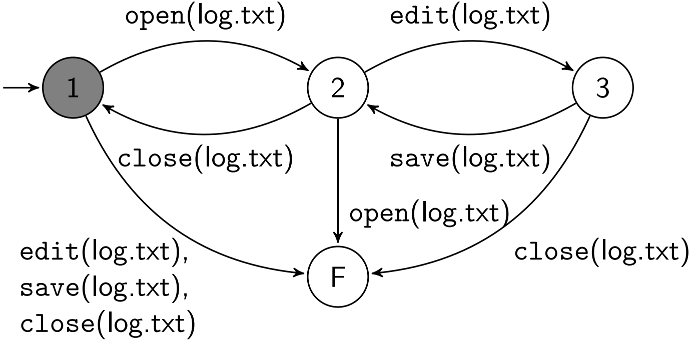
$[f \mapsto \text{log.txt}] \mapsto$ open(log.txt).edit(log.txt)



For Each Binding

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

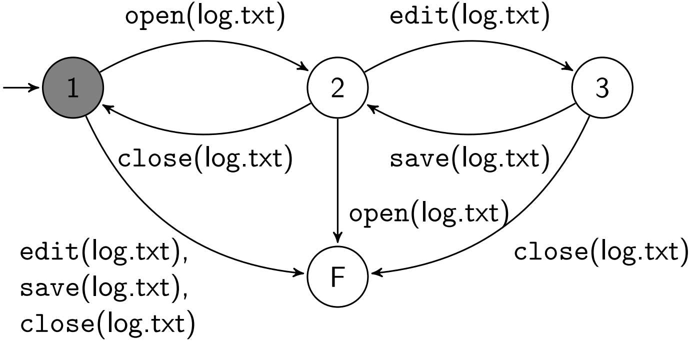
$[f \mapsto$ out.csv$]$	$\mapsto$	open(out.csv).close(out.csv)	✓
$[f \mapsto$ log.txt$]$	$\mapsto$	open(log.txt).edit(log.txt).close(log.txt)	



For Each Binding

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

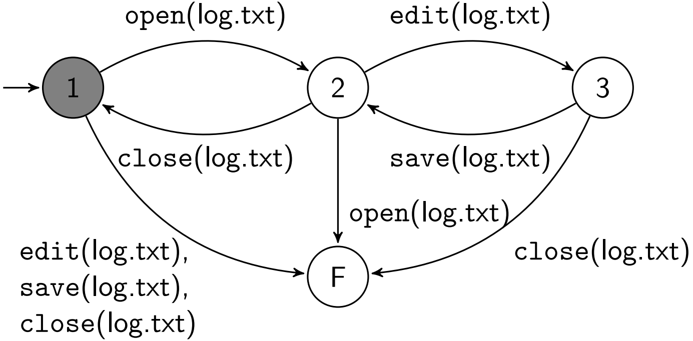
$[f \mapsto$ out.csv$]$	$\mapsto$	open(out.csv).close(out.csv)	✓
$[f \mapsto$ log.txt$]$	$\mapsto$	open(log.txt).edit(log.txt).close(log.txt)	



For Each Binding

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

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

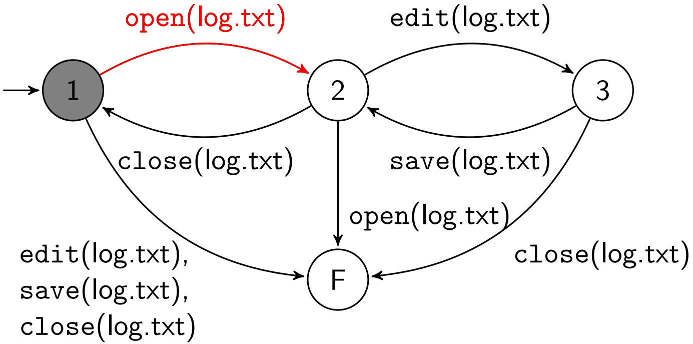


For Each Binding

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

[f ↦ out.csv] ↦ open(out.csv).close(out.csv) ✓

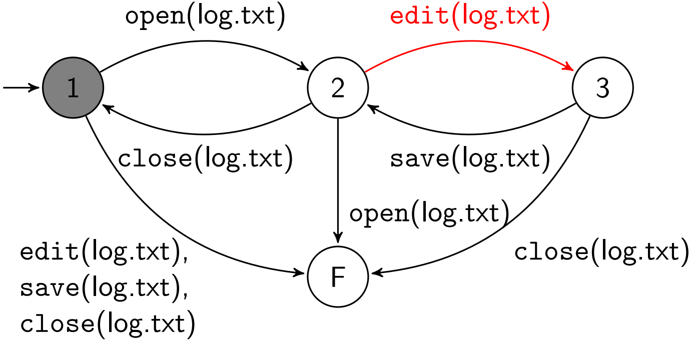
[f ↦ log.txt] ↦ open(log.txt).edit(log.txt).close(log.txt)



For Each Binding

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

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

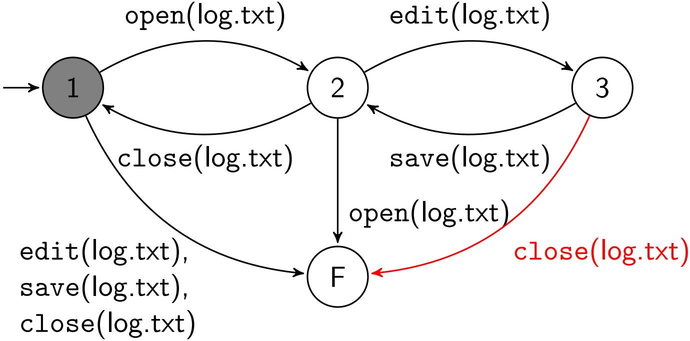


For Each Binding

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

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

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

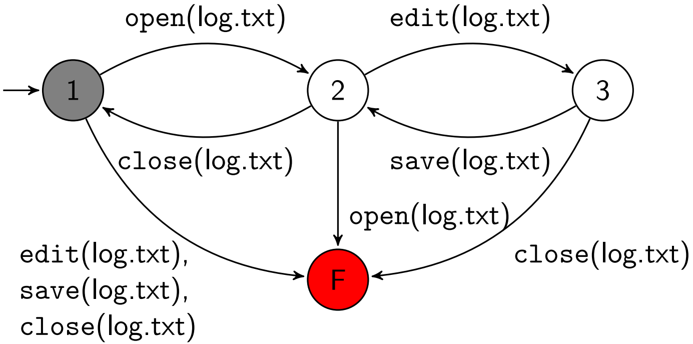


For Each Binding

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

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

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



For Each Binding

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

$$\begin{array}{lcll}
[f \mapsto \mathsf{out.csv}] & \mapsto & \texttt{open}(\mathsf{out.csv}).\texttt{close}(\mathsf{out.csv}) & \checkmark \\
[f \mapsto \mathsf{log.txt}] & \mapsto & \texttt{open}(\mathsf{log.txt}).\texttt{edit}(\mathsf{log.txt}).\texttt{close}(\mathsf{log.txt}) & \text{✗}
\end{array}$$

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 τ
- The domain of each variable is given by τ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding θ
  - Let E(θ) be the Event Automaton instantiated with θ
  - Let $\tau \downarrow_\theta$ be the trace projected with respect to θ
  - Check if $\tau \downarrow_\theta$ is in the language of E(θ)
- 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 τ
- The domain of each variable is given by τ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding θ
  - Let $\mathsf{E}(\theta)$ be the Event Automaton instantiated with θ
  - Let $\tau\downarrow_\theta$ be the trace projected with respect to θ
  - Check if $\tau\downarrow_\theta$ is in the language of $\mathsf{E}(\theta)$
- 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 $\mathsf{E}(\theta)$ for all bindings θ 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 τ
- The domain of each variable is given by τ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding θ
  - Let E(θ) be the Event Automaton instantiated with θ
  - Let $\tau \downarrow_\theta$ be the trace projected with respect to θ
  - Check if $\tau \downarrow_\theta$ is in the language of E(θ)
- 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(θ) for all bindings θ 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)	✗

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 τ
- The domain of each variable is given by τ and $\mathcal{A}$
- This gives us a set of relevant bindings
- For each binding θ
    - Let E(θ) be the Event Automaton instantiated with θ
    - Let $\tau \downarrow_\theta$ be the trace projected with respect to θ
    - Check if $\tau \downarrow_\theta$ is in the language of E(θ)
- 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(θ) for all bindings θ 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 $\mathsf{E}(\theta)$ for all bindings θ 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 $\mathsf{E}(\theta)$
    - If all quantifications are existential we must find at least **one** binding θ such that $\tau \downarrow_\theta$ is in the language of $\mathsf{E}(\theta)$

- 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 $\langle Q, \mathcal{A}, \delta, q_0, F \rangle$ is a tuple where

- Q is a set of states,
- $\mathcal{A} \subseteq \mathit{SymbolicEvent}$ is a alphabet of events,
- $\delta \subseteq (Q \times \mathcal{A} \times \mathit{Guard} \times \mathit{Assign} \times Q)$ 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 $\langle \Lambda, \mathrm{E} \rangle$ where

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

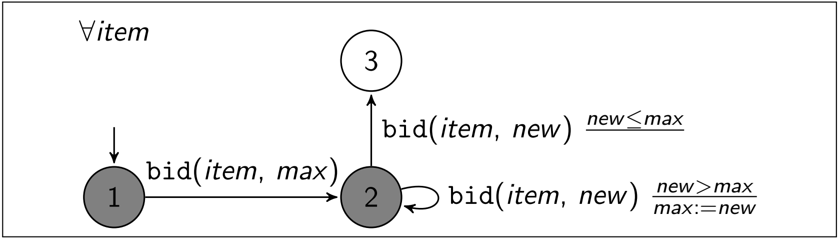
Free Variables

- 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

Auction Bidding Example

Property : Auction Bidding

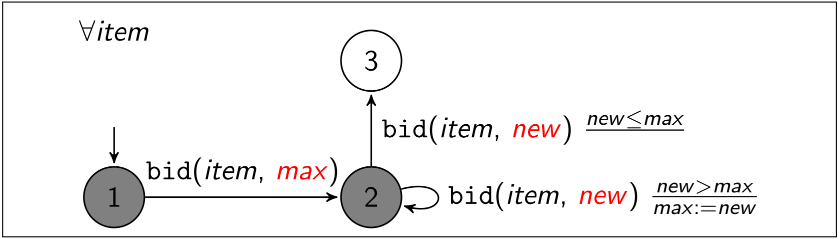
Amounts bid for an item should be strictly increasing.


$\forall item$
3
1
2
$\texttt{bid}(item, max)$
$\texttt{bid}(item, new)$ $\frac{new \leq max}{}$
$\texttt{bid}(item, new)$ $\frac{new > max}{max := new}$

Auction Bidding Example

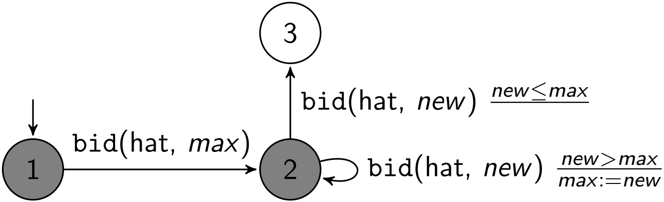
Property : Auction Bidding

Amounts bid for an item should be strictly increasing.


$\forall item$
3
1
2
$\texttt{bid}(item, max)$
$\texttt{bid}(item, new)$ $\frac{new \leq max}{}$
$\texttt{bid}(item, new)$ $\frac{new > max}{max := new}$

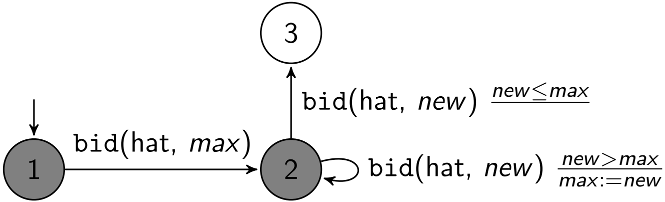
Bidding For A Hat

$\texttt{bid}(\text{hat}, 5).\texttt{bid}(\text{hat}, 10).\texttt{bid}(\text{hat}, 7)$


3
1
2
$\texttt{bid}(\text{hat},\ \mathit{max})$
$\texttt{bid}(\text{hat},\ \mathit{new})\ \ \frac{\mathit{new}\leq\mathit{max}}{}$
$\texttt{bid}(\text{hat},\ \mathit{new})\ \ \frac{\mathit{new}>\mathit{max}}{\mathit{max}:=\mathit{new}}$

Bidding For A Hat

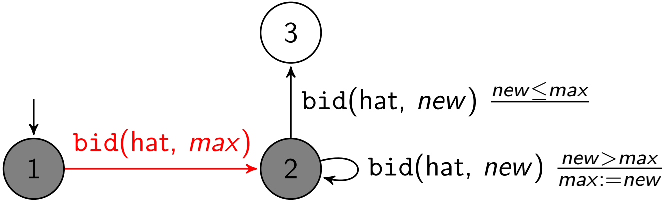
$\texttt{bid}(\mathsf{hat}, 5).\texttt{bid}(\mathsf{hat}, 10).\texttt{bid}(\mathsf{hat}, 7)$


3
bid(hat, *new*) $\frac{new \leq max}{}$
1
bid(hat, *max*)
2
bid(hat, *new*) $\frac{new > max}{max := new}$

$\langle 1, [\ ] \rangle$

Bidding For A Hat

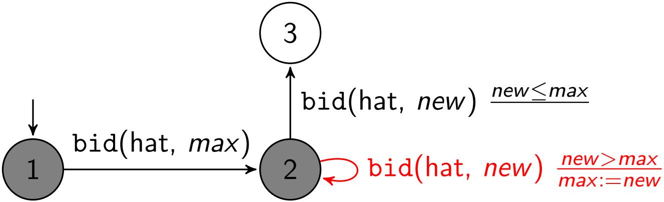
$\texttt{bid}(\text{hat}, 5).\texttt{bid}(\text{hat}, 10).\texttt{bid}(\text{hat}, 7)$



$$\langle 1, [\,]\rangle \quad \xrightarrow{\texttt{bid}(\text{hat},5)} \quad \langle 2, [\mathit{max} \mapsto 5]\rangle$$

Bidding For A Hat

$\texttt{bid}(\mathrm{hat}, 5).\texttt{bid}(\mathrm{hat}, 10).\texttt{bid}(\mathrm{hat}, 7)$

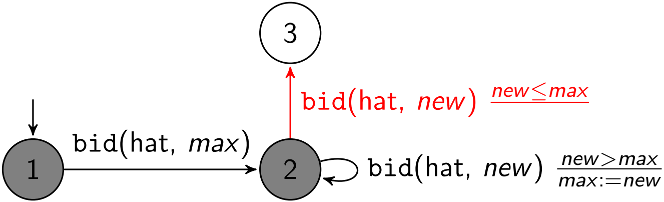


$\langle 1, [\,]\rangle \xrightarrow{\texttt{bid}(\mathrm{hat},5)} \langle 2, [max \mapsto 5]\rangle$

$\xrightarrow{\texttt{bid}(\mathrm{hat},10)} \langle 2, [new \mapsto 10, max \mapsto 10]\rangle$

Bidding For A Hat

$\texttt{bid}(\mathsf{hat}, 5).\texttt{bid}(\mathsf{hat}, 10).\texttt{bid}(\mathsf{hat}, 7)$


3
bid(hat, *new*) $\frac{new \leq max}{}$
1
bid(hat, *max*)
2
bid(hat, *new*) $\frac{new > max}{max := new}$

$\langle 1, [\ ]\rangle \xrightarrow{\texttt{bid}(\mathsf{hat},5)} \langle 2, [max \mapsto 5]\rangle$

$\xrightarrow{\texttt{bid}(\mathsf{hat},10)} \langle 2, [new \mapsto 10, max \mapsto 10]\rangle$

$\xrightarrow{\texttt{bid}(\mathsf{hat},7)} \langle 3, [new \mapsto 7, max \mapsto 10]\rangle$

Outline

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

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

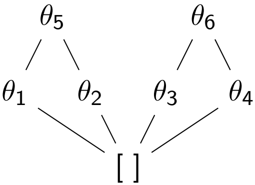
- The semantics for Quantified Event Automata are given in terms of a whole trace
- Required as we quantify over values in the whole trace
- This is inappropriate for monitoring at runtime
- Solution: Develop a small-step semantics that processes the trace one event at a time
- Two semantics give equivalent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

$$\mathit{Binding} \rightharpoonup \mathit{Trace}$$

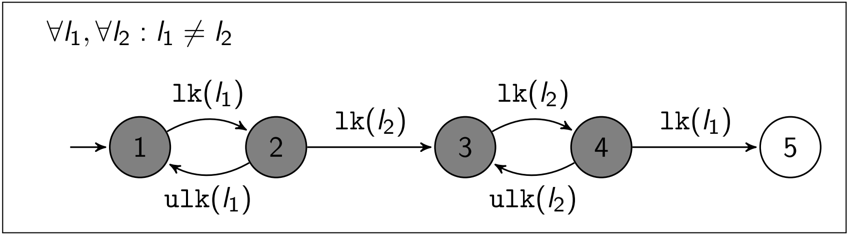
- When adding a new binding use the largest (given by partial order on bindings) existing consistent binding



Lock Ordering Example

Property : Lock Ordering

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



$\mathtt{lk} = \mathtt{lock}$ $\quad$ $\mathtt{ulk} = \mathtt{unlock}$

Lock Ordering Example : Computing Projections

$$\texttt{lk}(A).\texttt{lk}(B).\texttt{ulk}(B).\texttt{ulk}(A).\texttt{lk}(B).\texttt{lk}(A)$$

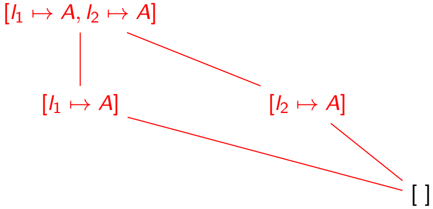
Lock Ordering Example : Computing Projections

[ ]

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ			

Lock Ordering Example : Computing Projections

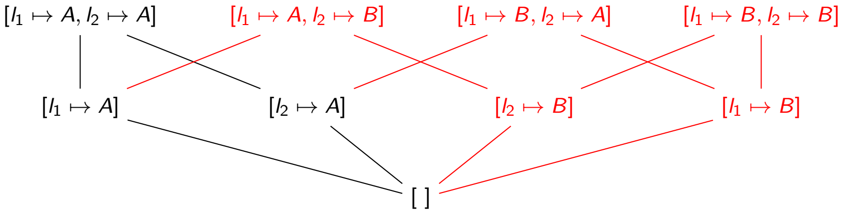
$\mathtt{lk}(A)$



partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	$\mathtt{lk}(\mathrm{A})$
A		$\mathtt{lk}(\mathrm{A})$			
	A	$\mathtt{lk}(\mathrm{A})$			

Lock Ordering Example : Computing Projections

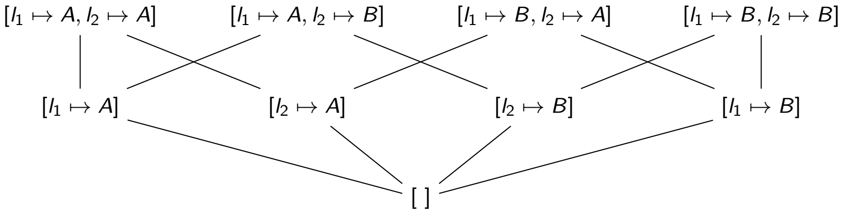
$\texttt{lk}(A).\texttt{lk}(B)$


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

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	lk(A)
A		lk(A)	A	B	lk(A).lk(B)
	A	lk(A)	B	A	lk(A).lk(B)
B		lk(B)	B	B	lk(B)
	B	lk(B)			

Lock Ordering Example : Computing Projections

$$\mathtt{lk}(A).\mathtt{lk}(B).\mathtt{ulk}(B)$$



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

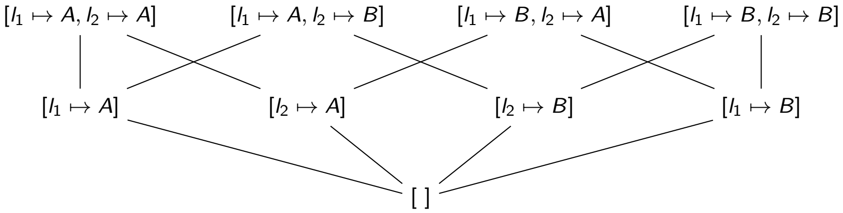
$[l_1 \mapsto A]$ $[l_2 \mapsto A]$ $[l_2 \mapsto B]$ $[l_1 \mapsto B]$

$[\,]$

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	$\mathtt{lk}(\mathrm{A})$
A		$\mathtt{lk}(\mathrm{A})$	A	B	$\mathtt{lk}(\mathrm{A}).\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B})$
	A	$\mathtt{lk}(\mathrm{A})$	B	A	$\mathtt{lk}(\mathrm{A}).\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B})$
B		$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B})$	B	B	$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B})$
	B	$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B})$			

Lock Ordering Example : Computing Projections

$$\mathtt{lk}(A).\mathtt{lk}(B).\mathtt{ulk}(B).\mathtt{ulk}(A)$$



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

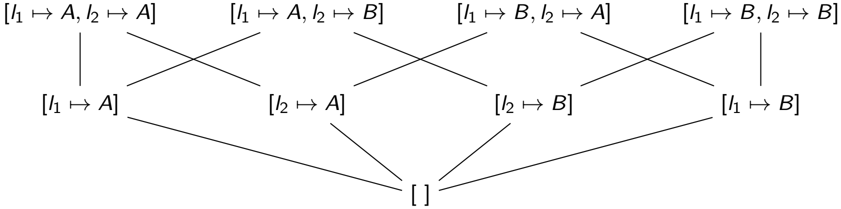
$[l_1 \mapsto A]$ $[l_2 \mapsto A]$ $[l_2 \mapsto B]$ $[l_1 \mapsto B]$

$[\,]$

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	lk(A).ulk(A)
A		lk(A).ulk(A)	A	B	lk(A).lk(B).ulk(B).ulk(A)
	A	lk(A).ulk(A)	B	A	lk(A).lk(B).ulk(B).ulk(A)
B		lk(B).ulk(B)	B	B	lk(B).ulk(B)
	B	lk(B).ulk(B)			

Lock Ordering Example : Computing Projections

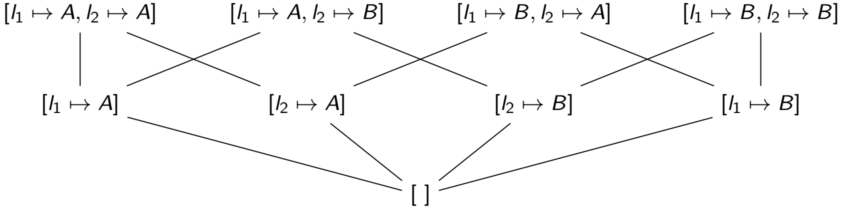
$$\mathtt{lk}(A).\mathtt{lk}(B).\mathtt{ulk}(B).\mathtt{ulk}(A).\mathtt{lk}(B)$$


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

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	$\mathtt{lk}(\mathrm{A}).\mathtt{ulk}(\mathrm{A})$
A		$\mathtt{lk}(\mathrm{A}).\mathtt{ulk}(\mathrm{A})$	A	B	$\mathtt{lk}(\mathrm{A}).\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B}).\mathtt{ulk}(\mathrm{A}).\mathtt{lk}(\mathrm{B})$
	A	$\mathtt{lk}(\mathrm{A}).\mathtt{ulk}(\mathrm{A})$	B	A	$\mathtt{lk}(\mathrm{A}).\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B}).\mathtt{ulk}(\mathrm{A}).\mathtt{lk}(\mathrm{B})$
B		$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B}).\mathtt{lk}(\mathrm{B})$	B	B	$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B}).\mathtt{lk}(\mathrm{B})$
	B	$\mathtt{lk}(\mathrm{B}).\mathtt{ulk}(\mathrm{B}).\mathtt{lk}(\mathrm{B})$			

Lock Ordering Example : Computing Projections

$$\mathtt{lk}(A).\mathtt{lk}(B).\mathtt{ulk}(B).\mathtt{ulk}(A).\mathtt{lk}(B).\mathtt{lk}(A)$$



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

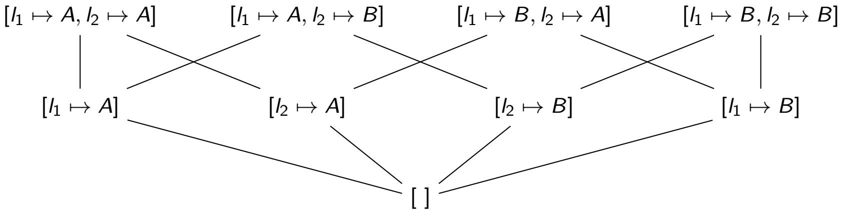
$[l_1 \mapsto A]$ $[l_2 \mapsto A]$ $[l_2 \mapsto B]$ $[l_1 \mapsto B]$

$[\,]$

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	lk(A).ulk(A).lk(A)
A		lk(A).ulk(A).lk(A)	A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
	A	lk(A).ulk(A).lk(A)	B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B		lk(B).ulk(B).lk(B)	B	B	lk(B).ulk(B).lk(B)
	B	lk(B).ulk(B).lk(B)			

Lock Ordering Example : Computing Projections

$$\texttt{lk}(A).\texttt{lk}(B).\texttt{ulk}(B).\texttt{ulk}(A).\texttt{lk}(B).\texttt{lk}(A)$$

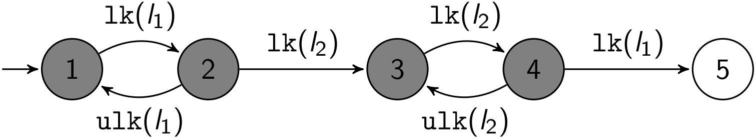

$[l_1 \mapsto A, l_2 \mapsto A]$
$[l_1 \mapsto A, l_2 \mapsto B]$
$[l_1 \mapsto B, l_2 \mapsto A]$
$[l_1 \mapsto B, l_2 \mapsto B]$
$[l_1 \mapsto A]$
$[l_2 \mapsto A]$
$[l_2 \mapsto B]$
$[l_1 \mapsto B]$
$[\,]$

partial binding		projection	total binding		projection
l_1	l_2		l_1	l_2	
		ϵ	A	A	lk(A).ulk(A).lk(A)
A		lk(A).ulk(A).lk(A)	A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
	A	lk(A).ulk(A).lk(A)	B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B		lk(B).ulk(B).lk(B)	B	B	lk(B).ulk(B).lk(B)
	B	lk(B).ulk(B).lk(B)			

Lock Ordering Example : Computing a Verdict

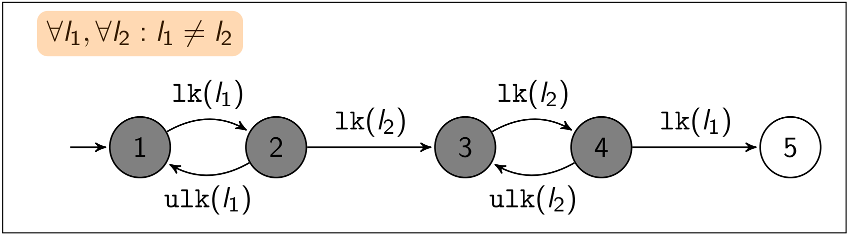
total binding		projection
l_1	l_2	
A	A	lk(A).ulk(A).lk(A)
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B	B	lk(B).ulk(B).lk(B)

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



Lock Ordering Example : Computing a Verdict

total binding		projection
l_1	l_2	
A	A	lk(A).ulk(A).lk(A)
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B	B	lk(B).ulk(B).lk(B)


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

Lock Ordering Example : Computing a Verdict

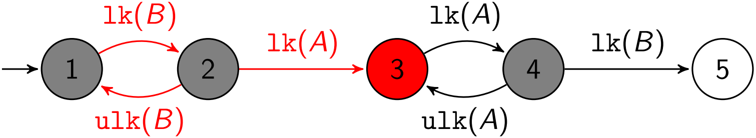
total binding		projection
l_1	l_2	
~~A~~	~~A~~	~~lk(A).ulk(A).lk(A)~~
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)
~~B~~	~~B~~	~~lk(B).ulk(B).lk(B)~~

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

Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~lk(A).ulk(A).lk(A)~~	
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✓
~~B~~	~~B~~	~~lk(B).ulk(B).lk(B)~~	

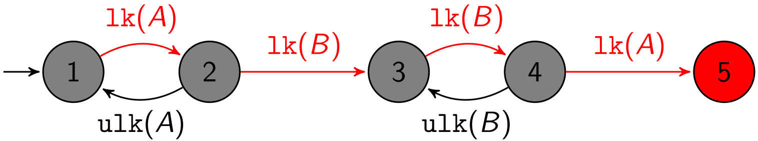
$\forall l_1, \forall l_2 : l_1 \neq l_2$



Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~lk(A).ulk(A).lk(A)~~	
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✗
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✓
~~B~~	~~B~~	~~lk(B).ulk(B).lk(B)~~	

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



Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~lk(A).ulk(A).lk(A)~~	
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✗
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✓
~~B~~	~~B~~	~~lk(B).ulk(B).lk(B)~~	

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

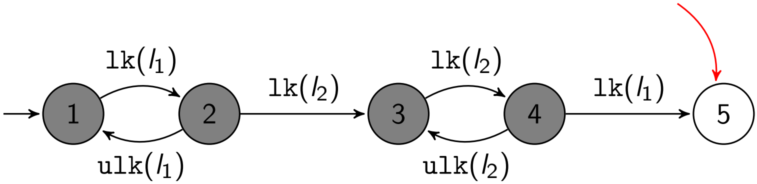
The trace does not satisfy the property

Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~`lk`(A).`ulk`(A).`lk`(A)~~	
A	B	`lk`(A).`lk`(B).`ulk`(B).`ulk`(A).`lk`(B).`lk`(A)	✗
B	A	`lk`(A).`lk`(B).`ulk`(B).`ulk`(A).`lk`(B).`lk`(A)	✓
~~B~~	~~B~~	~~`lk`(B).`ulk`(B).`lk`(B)~~	

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

Strong Failure State



Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~lk(A).ulk(A).lk(A)~~	
A	B	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✗
B	A	lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)	✓
~~B~~	~~B~~	~~lk(B).ulk(B).lk(B)~~	

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

No extensions of this trace can satisfy the property

Lock Ordering Example : Computing a Verdict

total binding		projection	
l_1	l_2		
~~A~~	~~A~~	~~`lk(A).ulk(A).lk(A)`~~	
A	B	`lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)`	✗
B	A	`lk(A).lk(B).ulk(B).ulk(A).lk(B).lk(A)`	✓
~~B~~	~~B~~	~~`lk(B).ulk(B).lk(B)`~~	

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

No extensions of this trace can satisfy the property

= StrongFailure

Practicalities

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

$$\mathit{Configuration} = \mathit{State} \times \mathit{Binding}$$

$$\mathit{Binding} \rightharpoondown \mathcal{P}(\mathit{Configuration})$$

- 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?