Unified Static and Runtime Verification of Object-Oriented Software

Wolfgang Ahrendt[1],
Mauricio Chimento[1], Gerardo Schneider[2], Gordon J. Pace[3]

[1]Chalmers University of Technology, Gothenburg, Sweden
[2]University of Gothenburg, Sweden
[3]University of Malta

Tallinn, August 2014

Static Verification vs. Runtime Verification

- Static verification
  - High precision
  - Use abstractions for increased automation

  but
  - Powerful judgements hard to achieve automatically
  - Often losing aspects of concrete system
- Runtime verification
  - Full precision (including real deployment)
  - Full automation

  but
  - Cannot judge future runs
  - Computational overhead of monitoring the running system

Project on Unified Static and Runtime Verification

Unified Static and Runtime Verification of Object-Oriented SW

Members:

- Wolfgang Ahrendt,
  Chalmers University of Technology
- Mauricio Chimento,
  Chalmers University of Technology
- Gerardo Schneider,
  University of Gothenburg

External collaborator:

- Gordon J. Pace,
  University of Malta

Vetenskapsrådet

Project on Unified Static and Runtime Verification

Unified Static and Runtime Verification of Object-Oriented SW

STARVOORS

Members:

- Wolfgang Ahrendt,
  Chalmers University of Technology
- Mauricio Chimento,
  Chalmers University of Technology
- Gerardo Schneider,
  University of Gothenburg

External collaborator:

- Gordon J. Pace,
  University of Malta

Vetenskapsrådet

Framework for Unified Static and Runtime Verification

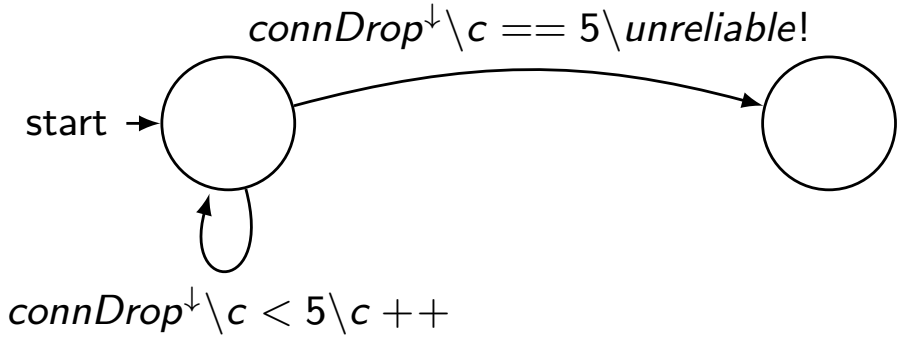
- Combine static and runtime verification
  - Combine data centric and control centric properties
  - Unified specification for both
- Use (partial) static verification results for partial evaluation of properties
- Runtime verification of resulting properties
- Increase safety and efficiency

LARVA: A Runtime Verification Tool for Java

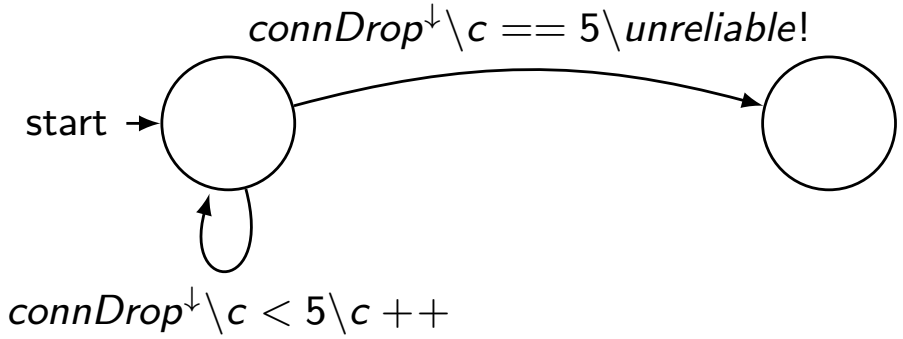
LARVA≡ Logical Automata for Runtime Verification and Analysis

- targets Java applications
- checks control oriented properties (untimed and real-time), specified in
  - DATE (Dynamic Automata with Timers and Events)
  - Lustre
  - duration calculus

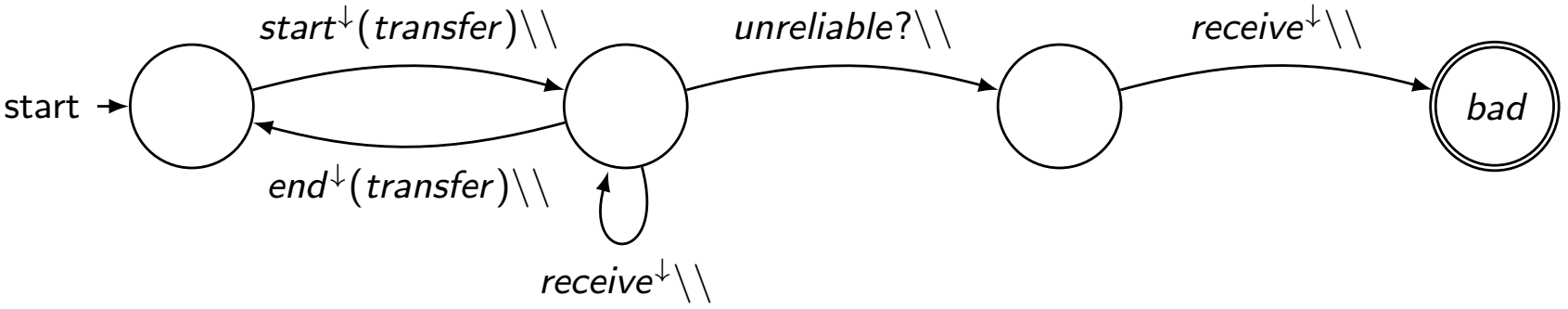
DATE Automaton Example



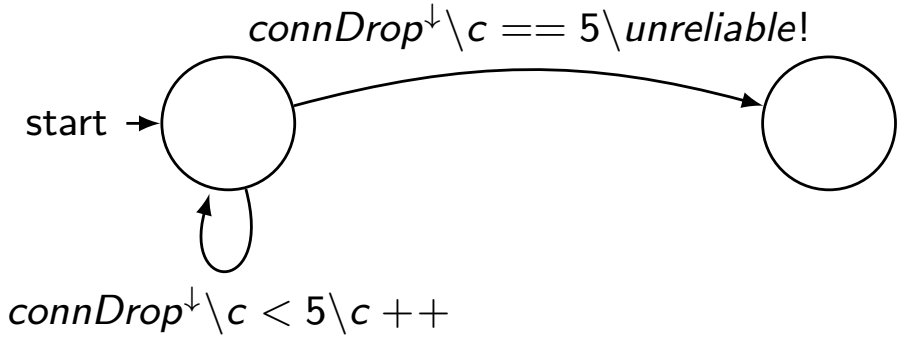
DATE Automaton Example



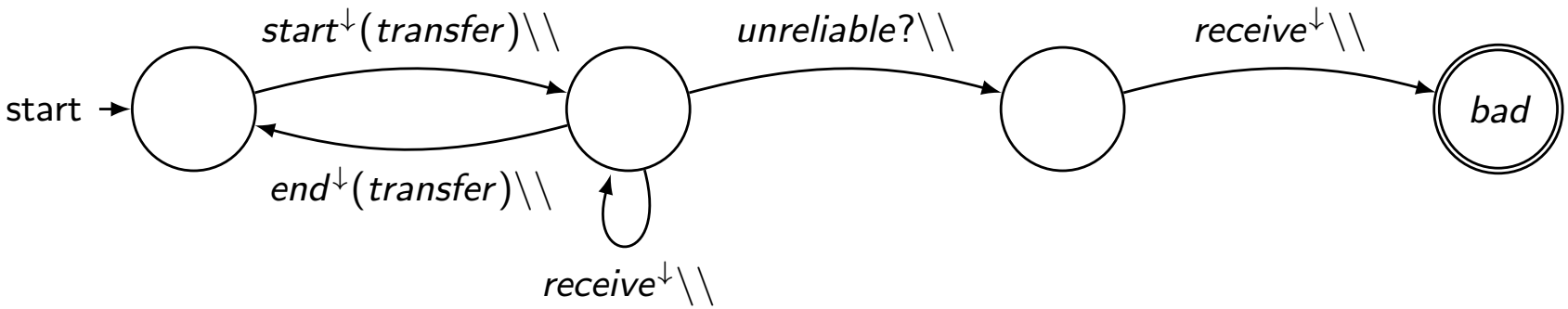
foreach transfer :



DATE Automaton Example



foreach transfer :



In general:

- communicating automata, event-triggered transitions, timers
- events: method entry/exit, timer events, synchronising events

LARVA Functionality

- LARVA input
  - DATE automaton (or alternative format)
  - application code

LARVA Functionality

- LARVA input
  - DATE automaton (or alternative format)
  - application code
- LARVA output
  - monitor
  - instrumented application code,
    with triggers for monotor

KeY

KeY is an *approach* and *tool* for the

- Formal specification of foremost *functional* properties
- Deductive verification, i.e., using theorem proving

of

- OO software, foremost JAVA and ABS

KeY

- Dynamic logic (generalisation of Hoare logic) as program logic
- Verification = symbolic execution + induction/invariants
- Sequent calculus
- Prover is automated + interactive
- most elaborate KeY instance KeY-Java
  - Java as target language
  - Supports specification language JML

Specification Language for Data and Control

ppDATE:

- Extending DATE with
  pre/post-conditions, associated to the automata's states:

$$q \xrightarrow{event|cond \mapsto act} q'$$

$$\tau(q) = \{\, ..., \{pre\}\, \texttt{method}\, \{post\}, ... \}$$

- Transition enabled if *cond* holds

Violating Traces

ppDATE trace $w \in (\Sigma^{\updownarrow} \times \Theta)^*$ is violating prefix if either

Violating Traces

ppDATE trace $w \in (\Sigma^{\updownarrow} \times \Theta)^*$ is violating prefix if either

- $(q_0, v_0) \stackrel{w}{\Rightarrow} (q, v)$ and $q \in \mathit{BadStates}$

Violating Traces

ppDATE trace $w \in (\Sigma^{\updownarrow} \times \Theta)^*$ is violating prefix if either

- $(q_0, v_0) \overset{w}{\Rightarrow} (q, v)$ and $q \in \mathit{BadStates}$
- $w = w_1 +\!\!+ \langle (m^{\downarrow}_{id}, \theta_1) \rangle +\!\!+ w_2 +\!\!+ \langle (m^{\uparrow}_{id}, \theta_2) \rangle$
  such that:
  1. $(q_0, v_0) \overset{w_1}{\Rightarrow} (q, v)$
  2. $\tau(q) \ni \{\mathit{pre}\}\, m\, \{\mathit{post}\}$
  3. $\theta_1 \models \mathit{pre}$
  4. $\theta_2 \not\models \mathit{post}$

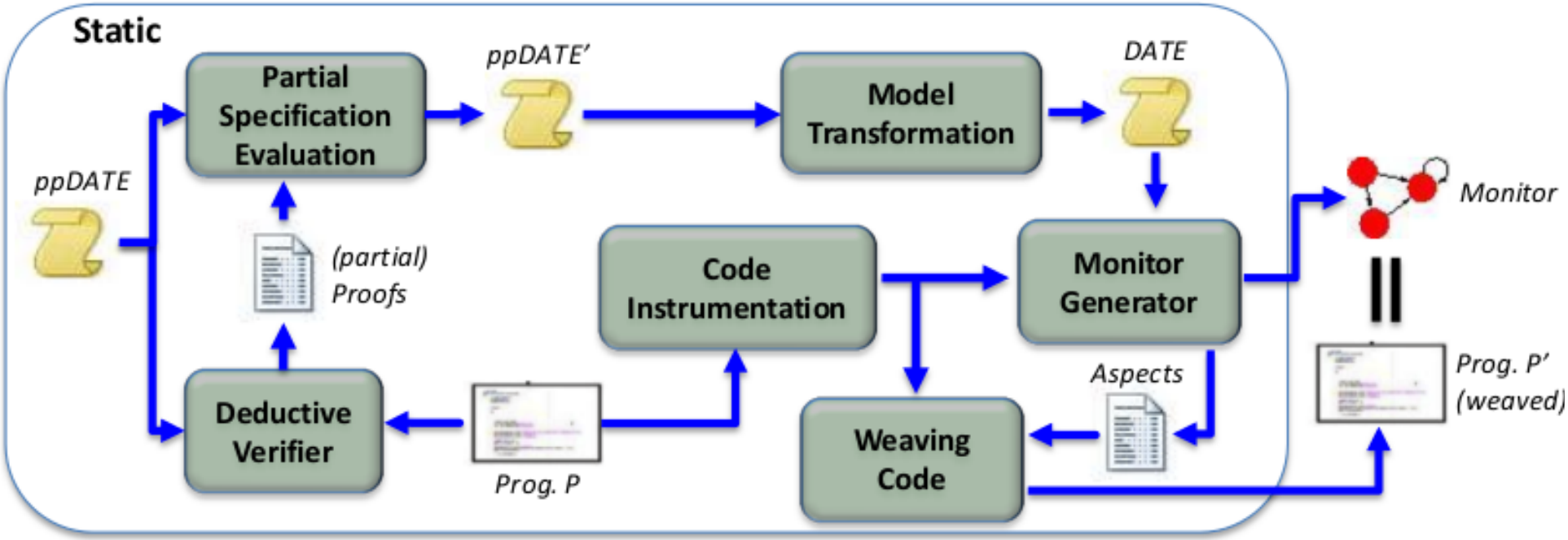
Violating Traces

ppDATE trace $w \in (\Sigma^{\updownarrow} \times \Theta)^*$ is violating prefix if either

- $(q_0, v_0) \overset{w}{\Rightarrow} (q, v)$ and $q \in \mathit{BadStates}$
- $w = w_1 +\!\!+ \langle (m^{\downarrow}_{id}, \theta_1) \rangle +\!\!+ w_2 +\!\!+ \langle (m^{\uparrow}_{id}, \theta_2) \rangle$
  such that:
  1. $(q_0, v_0) \overset{w_1}{\Rightarrow} (q, v)$
  2. $\tau(q) \ni \{\mathit{pre}\}\, m\, \{\mathit{post}\}$
  3. $\theta_1 \models \mathit{pre}$
  4. $\theta_2 \not\models \mathit{post}$

A violating trace has a violating prefix

High-level description of the framework

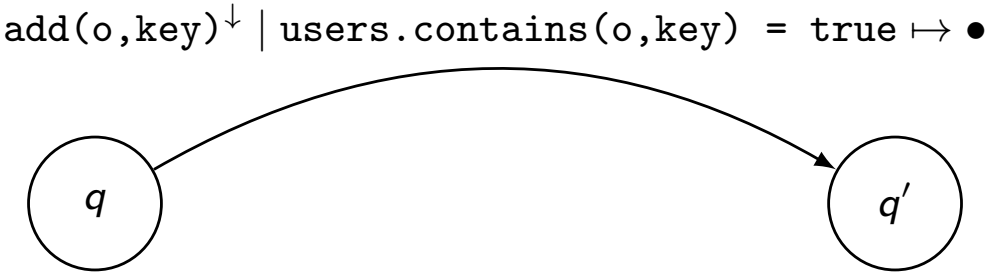


Case study: Login Example

Scenario:

- At login, new users are added to set `users`
- Assume `users` is implemented using hashing with open addressing
- Adding implemented by `users.add(u,key)`

Case study: Login Example



$$\tau(q) = \{\, \{\texttt{users.size} < \texttt{users.capacity}\}\, \texttt{add}\, \{\mathit{post}\}\, \}$$

$$\mathit{post}$$
$$\equiv$$
$$(\exists\ \texttt{int}\ i;\ i \geq 0\ \&\&\ i < \texttt{users.capacity}\,;\texttt{users.h}[i] = \texttt{o}\,;)$$

Case study: Login Example - Static Analysis

- Translation of Hoare triple to JML

```
class HashTable {
 ...
/*@ public normal_behavior
  @ requires size < capacity;
  @ ensures (\exists int i;
  @                  i>= 0 && i < capacity;
  @                  h[i] == o);
  @ assignable size, h[*];
  ......
  @*/
public void add (Object o, int key) {}
}
```

Case study: Login Example - Static Analysis

```
public void add (Object o, int key) {
   ...
   int i = hash(key);
   if (h[i] == null) {
      h[i] = o; size++;
   }
   else {
      while ... \\ store at next free slot
        ...}
}
```

Case study: Login Example - Static Analysis

- KeY tries to prove:
  $\texttt{size} < \texttt{capacity} \rightarrow \langle \texttt{add(o, key)} \rangle \exists i.\, \texttt{h}[i] = \texttt{o}$
- KeY will produce branches:
  $\ldots, \texttt{h[key\%capacity]} = \texttt{null} \vdash \ldots$
  and
  $\ldots, \neg\, \texttt{h[key\%capacity]} = \texttt{null} \vdash \ldots$
- first branch closes automatically, the second doesn't

Case study: Login Example - Partial Specification Evaluation

- First, for $\tau(q)$ replace $\{pre\}\,\mathtt{add}\,\{post\}$ by

Case study: Login Example - Partial Specification Evaluation

- First, for $\tau(q)$ replace $\{pre\}\,\mathtt{add}\,\{post\}$ by

$$\{pre \wedge \neg \texttt{users.h[key\%capacity] = null}\}\,\mathtt{add}\,\{post\}$$

and

$$\{pre \wedge \texttt{users.h[key\%capacity] = null}\}\,\mathtt{add}\,\{true\}$$

Case study: Login Example - Partial Specification Evaluation

- Second, new argument is added to distinguish different calls

Case study: Login Example - Partial Specification Evaluation

- Second, new argument is added to distinguish different calls

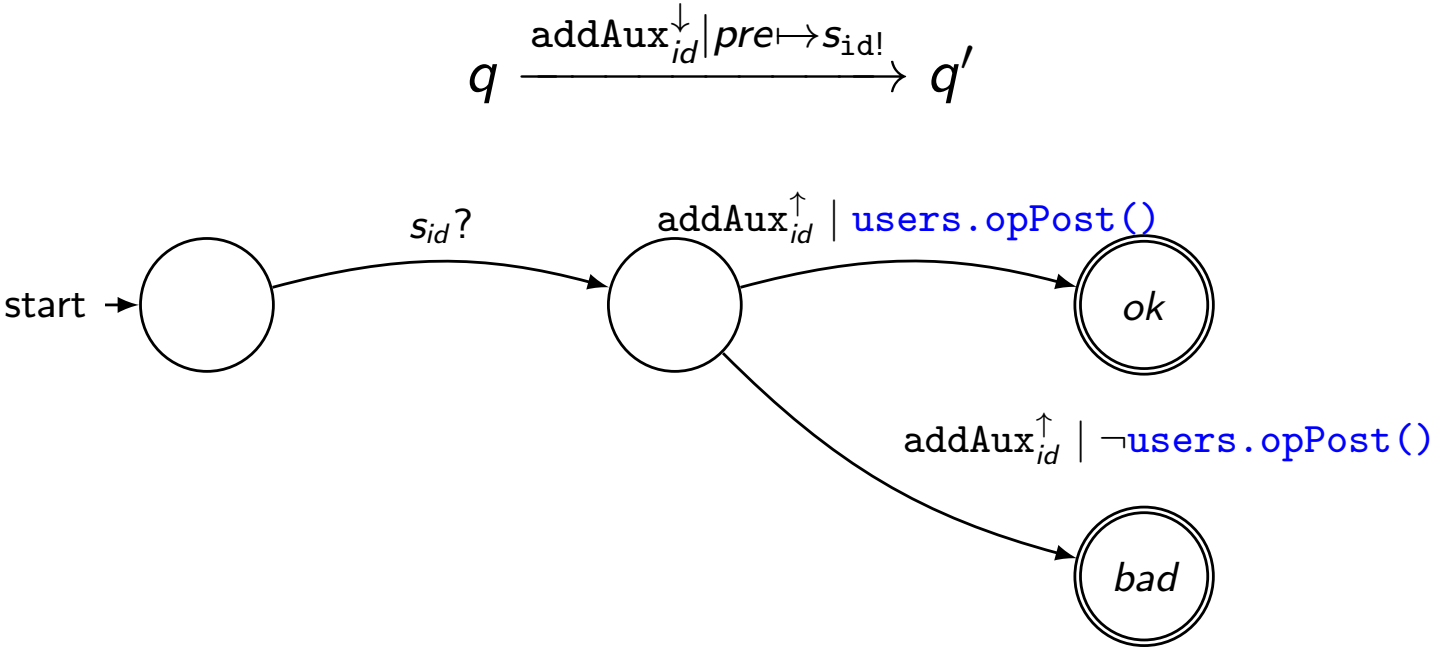
```
public void add (Object o, int key) {
    addAux(fid.getNewId(),o,key);
}
public void addAux (Integer id, Object o, int key) {
  //same code as add had before.
}
```

$$\{pre \wedge \neg \texttt{users.h[key\%capacity] = null}\}\ \texttt{addAux}\ \{post\}$$

and

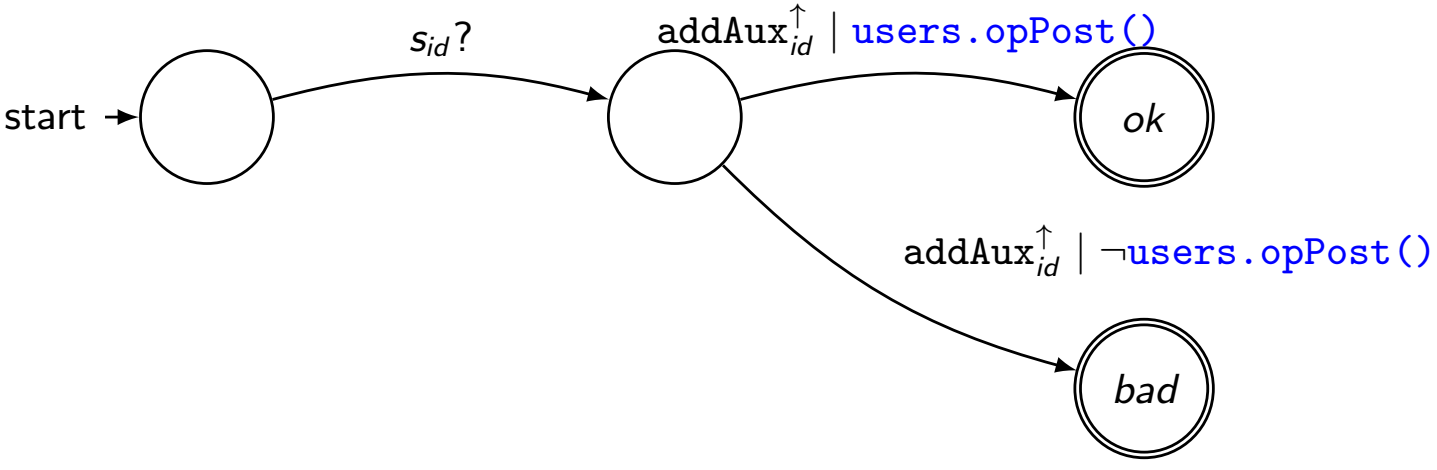
$$\{pre \wedge \texttt{users.h[key\%capacity] = null}\}\ \texttt{addAux}\ \{true\}$$

Case study: Login Example - Model Transformation



Case study: Login Example - Model Transformation

$$q \xrightarrow{\texttt{addAux}^{\downarrow}_{id}|\texttt{users.contains(o, key)} = \mathit{true} \mapsto \text{if } \mathit{pre} \text{ then } s_{\texttt{id!}}} q'$$



$$\xrightarrow{\texttt{addAux}^{\downarrow}_{id}|\neg(\texttt{users.contains(o, key)} = \texttt{true}) \wedge (\mathit{pre} \wedge \neg\, \texttt{users.h[key\%capacity]} = \texttt{null}) \mapsto s_{\texttt{id!}}} q$$

Case study: Login Example - Monitor Generation

- Finally, LARVA generates the monitors which will control the partially verified property.

Reference

- Wolfgang Ahrendt, Gordon J. Pace, Gerardo Schneider
  A Unified Approach for Static and Runtime Verification:
  – Framework and Applications
  ISoLA 2012
  Springer, LNCS 7609