MONITORING DECENTRALIZED SPECIFICATIONS

Antoine El-Hokayem    Yliès Falcone
December 7, 2017

Univ. Grenoble Alpes, Inria, CNRS (Grenoble, France)

GDR GPL - Journée MTV2
Paris, France
(Decentralized) Monitoring
Monitoring (aka runtime verification) → Overview

- **Lightweight** verification technique
- Checks whether a run of a program conforms to a specification (As opposed to model checking which verifies all runs)
- Monitors are synthesized and integrated to observe the system
- Monitors determine a **verdict**: \(\mathbb{B}_3 = \{\top, \bot, ?\}\)
  - \(\top\) (true): run complies with specification
  - \(\bot\) (false): run does not comply with specification
  - ?: verdict cannot be determined (yet)

![Diagram of monitoring process](image)
MONITORING $\leftrightarrow$ SYSTEM ABSTRACTION

1. Components ($C$)
2. Atomic propositions ($AP$)
3. Observations/Events ($AP \rightarrow \mathbb{B}_2$, possibly partial)
4. Trace: a sequence of events for each component (partial function)

Example

1. $\{c_0, c_1\}$ (Temp sensor + Fan)
2. $\{t_{\text{low}}, t_{\text{med}}, t_{\text{high}}, t_{\text{crit}}, \text{fan}\}$ (e.g., $t_{\text{crit}}$ “temperature critical”)
3. $\{\langle t_{\text{low}}, \top \rangle, \langle \text{fan}, \bot \rangle\} \rightarrow \text{“temperature is low and fan is not on”}$

$$
\begin{bmatrix}
    0 \mapsto c_0 & \mapsto \{\langle t_{\text{low}}, \top \rangle, \langle t_{\text{med}}, \bot \rangle, \ldots\} & 0 \mapsto c_1 & \mapsto \{\langle \text{fan}, \bot \rangle\} \\
    1 \mapsto c_0 & \mapsto \{\langle t_{\text{med}}, \top \rangle, \ldots\} & 1 \mapsto c_1 & \mapsto \{\langle \text{fan}, \bot \rangle\} \\
    2 \mapsto c_0 & \mapsto \{\langle t_{\text{high}}, \top \rangle, \ldots\} & 2 \mapsto c_1 & \mapsto \{\langle \text{fan}, \top \rangle\}
\end{bmatrix}
$$

$$
\{\langle t_{\text{low}}, \top \rangle, \langle \text{fan}, \bot \rangle, \ldots\} \cdot \{\langle t_{\text{med}}, \top \rangle, \langle \text{fan}, \bot \rangle, \ldots\} \cdot \{\langle t_{\text{high}}, \top \rangle, \langle \text{fan}, \top \rangle, \ldots\}
$$
“Fan must always be turned on when temperature is high”

\[ G(\text{t}_{\text{high}} \implies X \text{fan}) \]

1. At \( t = 1 \), from \( q_0 \):
   - 1.1 Observe
     - \( \text{t}_{\text{high}} \) \( \top \)
     - \( \text{fan} \) \( \bot \)
   - 1.2 Evaluate
     - \( \neg \text{t}_{\text{high}} \) \( \bot \)
     - \( \text{t}_{\text{high}} \) \( \top \)

2. At \( t = 2 \), from \( q_1 \):
   - 2.1 Observe
     - \( \text{t}_{\text{high}} \) \( \top \)
     - \( \text{fan} \) \( \bot \)
   - 2.2 Evaluate
     - \( \text{fan} \land \neg \text{t}_{\text{high}} \) \( \bot \)
     - \( \text{fan} \land \text{t}_{\text{high}} \) \( \bot \)
     - \( \neg \text{fan} \) \( \top \)

Monitoring this property requires a central observation point!

A. El-Hokayem, Y. Falcone, Monitoring Decentralized Specifications
DECENTRALIZED MONITORING $\rightarrow$ PROBLEM STATEMENT

- General setting
  - $C = \{c_0, \ldots, c_n\}$: components
  - $AP = AP_0 \cup \ldots \cup AP_n$: atomic propositions, partitioned by $C$
  - no central observation point
  - but monitors attached to components

- Issues in decentralized monitoring
  - partial views of $AP$ – unknown global state
  - partial execution of the automaton (evaluation)
  - communication between monitors

\[
\begin{align*}
&c_1 \quad \ldots \quad c_i \quad \ldots \quad c_n \\
&\downarrow AP_1 \quad \ldots \quad \downarrow AP_i \quad \ldots \quad \downarrow AP_n \\
&M_1 \quad \ldots \quad M_i \quad \ldots \quad M_n
\end{align*}
\]
DECENTRALIZED MONITORING \(\rightarrow\) PROBLEM STATEMENT

- General setting
- Issues in decentralized monitoring
  - partial views of \(AP\) – unknown global state
  - partial execution of the automaton (evaluation)
  - communication between monitors
- Existing approaches:
  - based on LTL rewriting — unpredictability of monitor performance
  - all monitors check the same specification — inefficiency

A. El-Hokayem, Y. Falcone, Monitoring Decentralized Specifications
**GOALS**

Define a methodology of design and evaluation of decentralized monitoring

1. **Aim for predictable behavior**
   - Move from LTL → Automata
   - Common ground to compare existing (and future) strategies

2. **Separate** monitor synthesis from monitoring strategies
   - Centralized specification → Decentralized specification
   - Monitorability of a decentralized specification
   - Define a general decentralized monitoring algorithm

★ Extend tooling support for the design methodology
★ Ensure reproducibility
(Decentralized) Monitoring

Monitoring with EHEs

Monitoring Decentralized Specifications

The THEMIS Approach

Conclusions
Monitoring with EHEs
EXECUTION HISTORY ENCODING $\rightarrow$ INFORMATION AS Atoms

- Encode the execution as a datastructure that
  - supports **flexibility** when receiving **partial** information
  - is insensitive to the reception **order** of information
  - has **predictable** size and operations

- Atomic propositions $\rightarrow$ Atoms
  - Allow algorithms to **add data** to observations ($\text{enc} : AP \rightarrow Atoms$)
  - Ordering information (timestamp, round number, vector clock etc.)

- Monitors store Atoms in their Memory

- Monitors need to evaluate $\mathit{Expr}_{Atoms}$
  - rewrite using Memory
  - simplify using Boolean logics (much easier than simplification for LTL)

\[
\mathit{Expr}_{Atoms} \times \mathit{Mem} \rightarrow \mathbb{B}_3 \\
\text{eval}(expr, M) = \text{simplify}(\text{rw}(expr, M)) \\
\text{eval}(\langle 1, t_{\text{high}} \rangle \land \langle 2, \text{fan} \rangle, [\langle 1, t_{\text{high}} \rangle \mapsto \bot]) = \bot \land \langle 2, \text{fan} \rangle = \bot
\]
EXECUTION HISTORY ENCODING $\rightarrow$ AUTOMATA EXECUTION

- **EHE** is a partial function:

  $$\mathcal{I} : \mathbb{N} \times Q_A \rightarrow Expr_{Atoms}$$

  $$\mathcal{I}(t, q) = expr$$

- For a given timestamp $t$
- The automaton is in state $q$ iff
- $eval(expr, M) = \top$

$$\mathcal{I}(2, q_0) = [\neg \langle 1, t_{\text{high}} \rangle \land \neg \langle 2, t_{\text{high}} \rangle]$$

$$\lor [\langle 1, t_{\text{high}} \rangle \land (\langle 2, \text{fan} \rangle \land \neg \langle 2, t_{\text{high}} \rangle)]$$

$eval(\mathcal{I}(2, q_0), [\langle 1, t_{\text{high}} \rangle \mapsto \bot])$

$$= eval(\neg \langle 2, t_{\text{high}} \rangle, \ldots) = ?$$

- **EHE** is constructed recursively and lazily (as needed and on-the-fly) using $A$
**EXECUTION HISTORY ENCODING** $\rightarrow$ **CONSTRUCTION**

$$\mathcal{I}^2 = \text{mov}([0 \mapsto q_0 \mapsto \top], 0, 2)$$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$q$</th>
<th>$\text{expr}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$q_0$</td>
<td>$\top$</td>
</tr>
<tr>
<td>1</td>
<td>$q_0$</td>
<td>$\top \land \neg \langle 1, a \rangle \land \neg \langle 1, b \rangle$</td>
</tr>
<tr>
<td>1</td>
<td>$q_1$</td>
<td>$\langle 1, a \rangle \lor \langle 1, b \rangle$</td>
</tr>
<tr>
<td>2</td>
<td>$q_0$</td>
<td>$(\neg \langle 1, a \rangle \land \neg \langle 1, b \rangle) \land (\neg \langle 2, a \rangle \land \neg \langle 2, b \rangle)$</td>
</tr>
<tr>
<td>2</td>
<td>$q_1$</td>
<td>$[(\neg \langle 1, a \rangle \land \neg \langle 1, b \rangle) \land (\langle 2, a \rangle \lor \langle 2, b \rangle)] \lor [(\langle 1, a \rangle \lor \langle 1, b \rangle) \land \top]$</td>
</tr>
</tbody>
</table>

...
EXECUTION HISTORY ENCODING $\rightarrow$ Properties

1. **Soundness** (provided that observations can be totally ordered)
   - For the same trace, EHE and $\mathcal{A}$ report the same state
     $\rightarrow$ They find the same verdict

2. **Strong Eventual Consistency** (SEC)
   - We can merge EHEs by disjoining ($\lor$) each entry $\langle t, q \rangle$
     - $\lor$ is commutative, associative and idempotent
   $\rightarrow$ EHE is a state-based replicated data-type (CvRDT)
   $\rightarrow$ Monitors that exchange their EHE find the same verdict
   $\rightarrow$ Can monitor centralized specification shared with multiple monitors

3. **Predictable size**
   - The EHE encodes all potential and past states, as needed
     - The more we keep track of potential states, the bigger the size
   $\rightarrow$ We can assess algorithms by how they manipulate the EHE
EXECUTION HISTORY ENCODING \rightarrow ANALYSIS

- Information Delay ($\delta$)
  Timestamps needed to expand before determining a state
  Potential states to keep track of
- Size of expression grows with each move beyond $t$
- Size of EHE:

$$|\mathcal{I}^\delta| = O(\delta |Q| \sum_{1}^{\delta} LP)$$

$$= O(\delta^2 |Q| LP)$$

$$t \mapsto q \mapsto T$$

$$t + 1 \mapsto \begin{cases} q_0 \mapsto e_{10} \\ q_1 \mapsto e_{11} \\ \vdots \end{cases} |Q|$$

$$t + 2 \mapsto \begin{cases} q_{|Q|-1} \mapsto e_{1(|Q|-1)} \\ q_0 \mapsto e_{20} \\ \vdots \end{cases} |Q|$$

$$t + \delta \mapsto \begin{cases} q_0 \mapsto e_{\delta 0} \\ q_1 \mapsto e_{\delta 1} \\ \vdots \end{cases} |Q|$$

$$q_{|Q|-1} \mapsto e_{\delta(|Q|-1)}$$
Monitoring Decentralized Specifications
DECENTRALIZED SPECIFICATION

- Each monitor is associated with a tuple $\langle A, c \rangle$
  - $A$ is its specification automaton
  - $c$ is the component the monitor is attached to
- The transition labels of an automaton $A$ are restricted to:
  - Atomic propositions local to the attached component
  - References to other monitors

\[
\begin{align*}
\neg t_{\text{high}} & \quad t_{\text{high}} & \quad \text{fan} \land t_{\text{high}} \\
\text{fan} \land \neg t_{\text{high}} & \quad \neg \text{fan} & \quad \neg m_1 & \quad m_1 \land t_{\text{high}} \\

\end{align*}
\]
DECENTRALIZED SPECIFICATION → SEMANTICS & MONITORABILITY

- For an automaton $A_k$, to evaluate a label $m_j$ at $t$ with a trace $tr$
  - Run $tr$ starting with $t$ on $A_j$ starting from $q_{j_0}$
  - Consider the verdict of the run to be the observation $m_j$ at $t$

(!!) If $A_j$ never reaches a final verdict we will not be able to monitor $A_k$

(??) Monitorability: “From any state in $A_k$, we can reach a final verdict”
- monitorable($A_k$) iff $\forall q \in Q_{A_k}, \exists q_f \in Q_{A_k}, \exists e_f \in \text{paths}(q, q_f)$, s.t.
  1. Path can be taken: $e_f$ is satisfiable;
  2. Path leads to a verdict: $\text{ver}_k(q_f) \in \{\bot, \top\}$;
  3. All its dependencies are monitorable:
    $\forall m_j \in \text{dep}(e_f)$: monitorable($A_j$).

- Expressions that determine paths between states ($n =$ path length)
- $\text{paths}(q_s, q_e) = \left\{ \text{expr} \mid \exists n \in \mathbb{N} : I^n(n, q_e) = \text{expr} \wedge I^n = \text{mov}([0 \mapsto q_s \mapsto \top], 0, n) \right\}$
GENERALIZED MONITORING ALGORITHM ➔ OVERVIEW

1. Setup (Deploy)
   1.1 Analyze and convert the *specification* as necessary
   1.2 Create monitors, and assign them a specification
       (!) The monitor handles encoding of $AP$ and Memory
   1.3 Attach monitors to components

2. Monitoring
   2.1 Wait to receive *observations* from attached component
   2.2 Receive messages (EHE) from monitors
   2.3 Process observations and messages (update the local EHE)
   2.4 Communicate with other monitors
The THEMIS Approach
THEMIS \rightarrow \textbf{OVERVIEW}

- **Design**: Design a monitoring algorithm
- **Instrument**: Create or re-use metrics. Metrics are automatically instrumented using AspectJ
- **Execute**: Use THEMIS tools to execute one or more monitoring run(s)
- **Analyze**: Measures are stored in a database for postmortem analysis
Setup

```java
Map<Integer, ? extends Monitor> setup() {
    config.getSpec().put("root", Convert.makeAutomataSpec(
        config.getSpec().get("root")));
    Map<Integer, Monitor> mons = new HashMap<Integer, Monitor>();
    Integer i = 0;
    for(Component comp : config.getComponents()) {
        MonMigrate mon = new MonMigrate(i);
        attachMonitor(comp, mon);
        mons.put(i, mon);
        i++;
    }
    return mons;
}
```

Monitor

```java
void monitor(int t, Memory<Atom> observations) throws ReportVerdict, ExceptionStopMonitoring {
    m.merge(observations);
    if(receive()) isMonitoring = true;
    if(isMonitoring) {
        if(!observations.isEmpty())
            ehe.tick();
        boolean b = ehe.update(m, -1);
        if(b) {
            VerdictTimed v = ehe.scanVerdict();
            if(v.isFinal())
                throw new ReportVerdict(v.getVerdict(), t);
            ehe.dropResolved();
        }
        int next = getNext();
        if(next != getID()) {
            Representation toSend = ehe.sliceLive();
            send(next, new RepresentationPacket(toSend));
            isMonitoring = false;
        }
    }
}
```
EXAMPLES ⇐ METRICS

```java
void setupRun(MonitoringAlgorithm alg) {
    addMeasure(new Measure("msg_num","Msgs",0L,Measures.addLong));
}

after(Integer to, Message m) : Commons.sendMessage(to, m) {
    update("msg_num", 1L);
}
```

```
SELECT alg, comps, avg(msg_num), avg(msg_data), count(*)
FROM bench WHERE alg in ('Migration', 'MigrationRR')
GROUP BY alg, comps
```

<table>
<thead>
<tr>
<th>alg</th>
<th>comps</th>
<th>avg(msg_num)</th>
<th>avg(msg_data)</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration</td>
<td>3</td>
<td>2.0422633601177</td>
<td>267.845871463</td>
<td>572600</td>
</tr>
<tr>
<td>Migration</td>
<td>4</td>
<td>2.16402472527473</td>
<td>668.129401098901</td>
<td>364000</td>
</tr>
<tr>
<td>Migration</td>
<td>5</td>
<td>3.33806822465134</td>
<td>3954.09705050886</td>
<td>530600</td>
</tr>
<tr>
<td>MigrationRR</td>
<td>3</td>
<td>32.7222301781348</td>
<td>482.572275585051</td>
<td>572600</td>
</tr>
<tr>
<td>MigrationRR</td>
<td>4</td>
<td>31.8533351648352</td>
<td>932.708425824176</td>
<td>364000</td>
</tr>
<tr>
<td>MigrationRR</td>
<td>5</td>
<td>19.2345269506219</td>
<td>4361.30746324915</td>
<td>530600</td>
</tr>
</tbody>
</table>
```
EXISTING ALGORITHMS

A. El-Hokayem, Y. Falcone, Monitoring Decentralized Specifications
STUDYING EXISTING ALGORITHMS ↔ EXPECTED BEHAVIOR

Orchestration
- $\delta$ is constant
- #Msgs is linear in components
- $|\text{Msg}|$ constant: observations per component

Migration
- $\delta$ is linear in components
- #Msgs is constant
- $|\text{Msg}|$ is size of EHE: $O(\delta^2)$, quadratic in components

Choreography
- $\delta$ is linear in network depth (algorithm)
- #Msgs is linear in network edges
- $|\text{Msg}|$ is constant

#Msgs and $|\text{Msg}|$ are predicted on a per round basis
Conclusions
SUMMARY AND FUTURE WORK

★ Decentralized Monitoring of (De)Centralized Specifications
  1. Aim for predictable behavior → Automata + EHE data structure
  2. Separate synthesis from monitoring: decentralized specifications
  3. Methodology + tool support for designing, measuring, comparing and extending decentralized RV algorithms
  4. Adapted and compared existing algorithms

★ Future Work
  1. Centralised specification → equivalent decentralized specifications
     • Optimize existing methods
     • Take into account topology of the monitored system
  2. Extend THEMIS
     • New metrics
     • Support a fully-asynchronous monitoring approach
     • Better visualization of (the behavior of) algorithms
  3. Runtime enforcement of centralized and decentralized specifications


Bartocci, E.: Sampling-based decentralized monitoring for networked embedded systems. In: Bortolussi et al. [12], pp. 85–99, http://dx.doi.org/10.4204/EPTCS.124.9


Cotard, S., Faucou, S., Béchennec, J., Queudet, A., Trinquet, Y.: A data flow monitoring service based on runtime verification for AUTOSAR. In: Min et al. [36], pp. 1508–1515


Diekert, V., Muscholl, A.: On distributed monitoring of asynchronous systems. In: Ong and de Queiroz [38], pp. 70–84, http://dx.doi.org/10.1007/978-3-642-32621-9_5


Falcone, Y.: You should better enforce than verify. In: Barringer et al. [5], pp. 89–105


Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Broy et al. [13], pp. 141–175


Misra, J., Nipkow, T., Sekerinski, E. (eds.): FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada,


Shapiro, M., Preguiça, N.M., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago et al. [17], pp. 386–400

