MONITORING DECENTRALIZED SPECIFICATIONS

Yliès Falcone

Ylies.Falcone@univ-grenoble-alpes.fr — www.ylies.fr

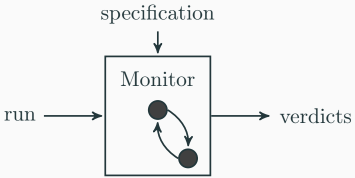
(overview of joint work with A. Bauer, C. Colombo, and A. El-Hokayem)

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

Grenoble - Kobe Workshop
Feb. 26, 2018, Grenoble, France

MONITORING (AKA RUNTIME VERIFICATION) $\hookrightarrow$ OVERVIEW

- Lightweight verification technique
- Checks whether a run of a (blackbox) 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)



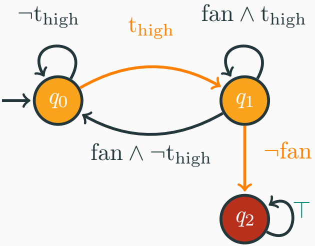
1. Components ($\mathcal{C}$)
2. Atomic propositions (AP)
3. Observations/Events ($AP \rightarrow \mathbb{B}_2$, possibly partial)
4. Trace: a sequence of events for each component (partial function)

Example

1. $\{c_0, c_1\}$ (Temp sensor + Fan)
2. $\{\mathrm{t_{low}}, \mathrm{t_{med}}, \mathrm{t_{high}}, \mathrm{t_{crit}}, \mathrm{fan}\}$ (e.g., $\mathrm{t_{crit}}$ “temperature critical”)
3. $\{\langle \mathrm{t_{low}}, \top\rangle, \langle \mathrm{fan}, \bot\rangle\}$ — “temperature is low and fan is not on”
4. $$\begin{bmatrix} 0 \mapsto c_0 & \mapsto \{\langle \mathrm{t_{low}}, \top\rangle, \langle \mathrm{t_{med}}, \bot\rangle, \ldots\} & 0 \mapsto c_1 & \mapsto \{\langle \mathrm{fan}, \bot\rangle\} \\ 1 \mapsto c_0 & \mapsto \{\langle \mathrm{t_{med}}, \top\rangle, \ldots\} & 1 \mapsto c_1 & \mapsto \{\langle \mathrm{fan}, \bot\rangle\} \\ 2 \mapsto c_0 & \mapsto \{\langle \mathrm{t_{high}}, \top\rangle, \ldots\} & 2 \mapsto c_1 & \mapsto \{\langle \mathrm{fan}, \top\rangle\} \end{bmatrix}$$

$$\{\langle \mathrm{t_{low}}, \top\rangle, \langle \mathrm{fan}, \bot\rangle, \ldots\} \cdot \{\langle \mathrm{t_{med}}, \top\rangle, \langle \mathrm{fan}, \bot\rangle, \ldots\} \cdot \{\langle \mathrm{t_{high}}, \top\rangle, \langle \mathrm{fan}, \top\rangle, \ldots\}$$

"Fan must always be turned on when temperature is high"



$$\boldsymbol{G}(t_{high} \implies \boldsymbol{X} fan)$$

1. At $t = 1$, from q_0:

   1.1 Observe

t_{high}	$\top$
fan	$\bot$

   1.2 Eval

$\neg t_{high}$	$\bot$
t_{high}	$\top$

2. At $t = 2$, from q_1:

   2.1 Observe

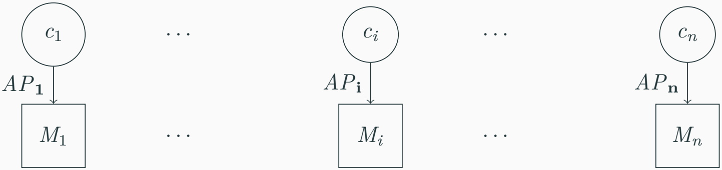
t_{high}	$\top$
fan	$\bot$

   2.2 Eval

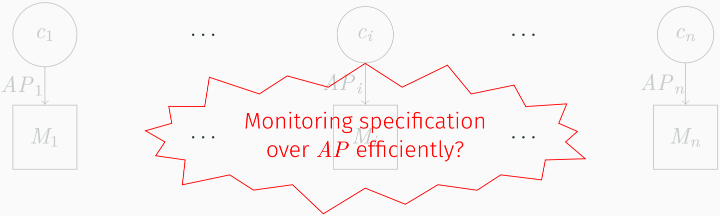
$fan \wedge \neg t_{high}$	$\bot$
$fan \wedge t_{high}$	$\bot$
$\neg fan$	$\top$

Monitoring this property requires a central observation point!

- General setting
  - $\mathcal{C} = \{c_0, \ldots, c_n\}$: components
  - $AP = AP_0 \cup \ldots \cup AP_n$: atomic propositions, partitioned by $\mathcal{C}$
  - no central observation point
  - but monitors attached to components
- Challenges:
  - partial views of AP – unknown global state
  - partial execution of the monitor (evaluation)
  - communication between and organisation of monitors



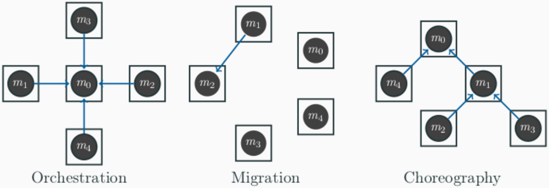
- General setting
- Challenges:
  - partial views of AP – unknown global state
  - partial execution of the monitor (evaluation)
  - communication between and organisation of monitors



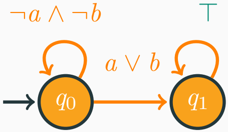
RESULTS

A methodology of design and evaluation of decentralized monitoring

1. Predictable monitor behavior
    - Specifications in LTL or as Automata
    - Data-structure: Execution History Encoding (EHE)
2. Separated monitor synthesis from monitoring strategies
    - Centralized specification → Decentralized specification
        - Monitors can now focus on parts of the specification
        - Monitors communicate with other monitors (explicitly)
    - Topologies of monitors (and dependencies)



3. THEMIS tool for the design and (reproducible) evaluation of decentralised monitoring algorithms



t	q	expr
0	q_0	$\top$
1	q_0	$\top \wedge \quad \neg\langle 1, a\rangle \wedge \neg\langle 1, b\rangle$
1	q_1	$\langle 1, a\rangle \vee \langle 1, b\rangle$
2	q_0	$(\neg\langle 1, a\rangle \wedge \neg\langle 1, b\rangle) \wedge (\neg\langle 2, a\rangle \wedge \neg\langle 2, b\rangle)$
2	q_1	$[(\neg\langle 1, a\rangle \wedge \neg\langle 1, b\rangle) \wedge (\langle 2, a\rangle \vee \langle 2, b\rangle)] \vee [(\langle 1, a\rangle \vee \langle 1, b\rangle) \wedge \top]$

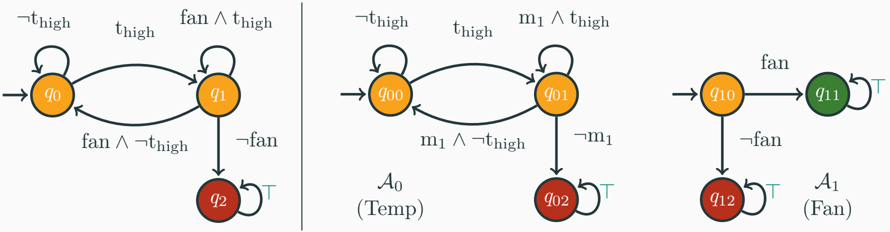
$\vdots$

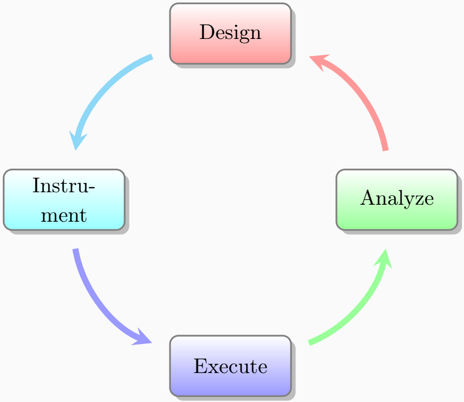
1. Soundness (provided that observations can be totally ordered)
   - For the same trace, EHE and $\mathcal{A}$ report the same state/verdict
2. Strong Eventual Consistency (SEC)
   - EHE is a state-based replicated data-type (CvRDT)
   - → Order of messages does not effect the outcome
   - → Monitors that exchange their EHE find the same verdict
3. Predictable size
   - The EHE encodes all potential and past states, as needed
   - → Can assess the complexity of algorithms by how they manipulate EHE

Algorithm	delay	# Msg	|Msg|
Orchestration	$\Theta(1)$	$\Theta(\|\mathcal{C}\|)$	$O(\|AP_c\|)$
Migration	$O(\|\mathcal{C}\|)$	$O(m)$	$O(m\|\mathcal{C}\|^2)$
Choreography	$O(\mathrm{depth}(\mathrm{m_{root}}))$	$\Theta(\|E\|)$	$\Theta(1)$

DECENTRALIZED SPECIFICATION

- Each monitor is associated with a tuple $\langle \mathcal{A}, c \rangle$
  - $\mathcal{A}$ is its specification automaton
  - c is the component the monitor is attached to
- The transition labels of an automaton $\mathcal{A}$ are restricted to:
  - Atomic propositions local to the attached component
  - References to other monitors
- Formal semantics and underlying issues in papers :-)





- **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

Use a common API to build algorithms and measures

(1) Design (monitoring algorithms)

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;
      }
    }
  }
```

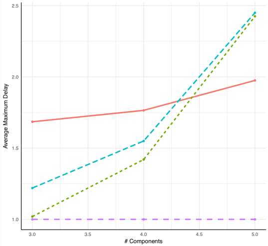
(2) Instrument (# msg)

```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);
}
```

(3) Execute (simulation) and (4) Analyze

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

	alg	comps	avg(msg_num)	avg(msg_data)	count(*)
1	Migration	3	2.04226336011177	267.8458714635	572600
2	Migration	4	2.16402472527473	668.129401098901	364000
3	Migration	5	3.33806822465134	3954.09705050886	530600
4	MigrationRR	3	32.7222301781348	482.572275585051	572600
5	MigrationRR	4	31.8533351648352	932.708425824176	364000
6	MigrationRR	5	19.2345269506219	4361.30746324915	530600



SUMMARY AND FUTURE WORK

- ⋆ Decentralized Monitoring of (De)Centralized Specifications
  1. Aim for predictable behavior → EHE data structure
  2. Separate synthesis from monitoring → decentralized specifications
  3. Methodology + tool support for designing, measuring, comparing and extending decentralized RV algorithms → THEMIS tool

`https://gitlab.inria.fr/monitoring/themis-demo`

- ⋆ Future/Ongoing Work
  1. Centralised specification → equivalent decentralized specifications
  2. Runtime enforcement of centralized and decentralized specifications
  3. Home Automation systems on iCasa with G. Vega and P. Lalanda
     - How to write clear, scalable, and modular specifications?
     - How to efficiently organize monitors?
     - How to manage interactions (and conflicts) between monitors?

Ábrahám, E., Palamidessi, C. (eds.): Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings, Lecture Notes in Computer Science, vol. 8461. Springer (2014)

Bartocci, E., Falcone, Y., Bonakdarpour, B., Colombo, C., Decker, N., Havelund, K., Joshi, Y., Klaedtke, F., Milewicz, R., Reger, G., Rosu, G., Signoles, J., Thoma, D., Zalinescu, E., Zhang, Y.: First international competition on runtime verification: rules, benchmarks, tools, and final results of crv 2014. International Journal on Software Tools for Technology Transfer pp. 1–40 (2017), `http://dx.doi.org/10.1007/s10009-017-0454-5`

Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. 20(4), 14 (2011)

Bauer, A.K., Falcone, Y.: Decentralised LTL monitoring. In: Giannakopoulou and Méry [13], pp. 85–100

- Broy, M., a. Peled, D., Kalus, G. (eds.): engineering dependable software systems, NATO science for peace and security series, d: information and communication security, vol. 34. ios press (2013)
- Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. Formal Methods in System Design 49(1-2), 109–158 (2016)
- Défago, X., Petit, F., Villain, V. (eds.): Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6976. Springer (2011)
- Duret-Lutz, A.: Manipulating LTL formulas using Spot 1.0. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13). Lecture Notes in Computer Science, vol. 8172, pp. 442–445. Springer, Hanoi, Vietnam (Oct 2013)
- El-Hokayem, A., Falcone, Y.: THEMIS: A tool for decentralized monitoring algorithms. In: Proceedings of 26th ACM SIGSOFT

International Symposium on Software Testing and Analysis (ISSTA'17-DEMOS), Santa Barbara, CA, USA, July 2017 (2017)

Falcone, Y., Cornebize, T., Fernandez, J.: Efficient and generalized decentralized monitoring of regular languages. In: Ábrahám and Palamidessi [1], pp. 66–83

Falcone, Y., Fernandez, J., Mounier, L.: What can you verify and enforce at runtime? STTT 14(3), 349–382 (2012)

Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Broy et al. [5], pp. 141–175

Giannakopoulou, D., Méry, D. (eds.): FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7436. Springer (2012)

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectj. In: Knudsen [15], pp. 327–353

- Knudsen, J.L. (ed.): ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Budapest, Hungary, June 18-22, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2072. Springer (2001)
- Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. 78(5), 293–303 (2009)
- Misra, J., Nipkow, T., Sekerinski, E. (eds.): FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings, Lecture Notes in Computer Science, vol. 4085. Springer (2006)
- Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra et al. [17], pp. 573–586
- Shapiro, M., Preguiça, N.M., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago et al. [7], pp. 386–400