ON THE MONITORING OF DECENTRALIZED SPECIFICATIONS

Yliès Falcone

(www.ylies.fr)

IST Austria, 3 October 2019

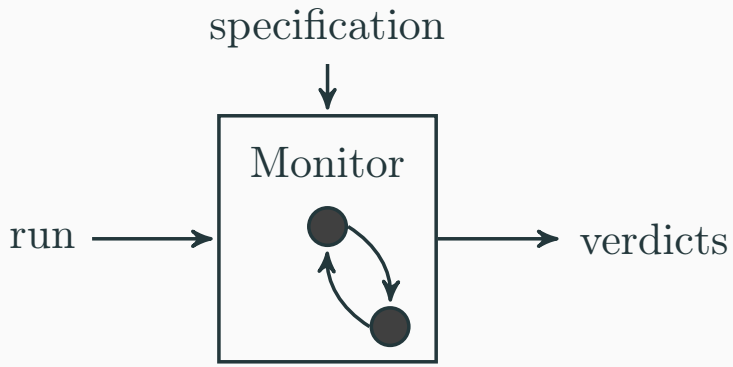
Univ. Grenoble Alpes, Inria, CNRS, Laboratoire d'Informatique de Grenoble, France

Based on joint recent work with Antoine El-Hokayem to appear in ACM TOSEM and some earlier work with Andreas Bauer and Cristian Colombo in Springer FMSD.

(Decentralized) Monitoring

MONITORING (AKA RUNTIME VERIFICATION) ↪ OVERVIEW

- Lightweight verification technique.
- Checks whether a run of a program conforms to a specification. (Incomplete, as opposed to exhaustive verification techniques.)
- Specification is formalized.
- Monitors are synthesized and integrated to observe the system.
- Monitors determine a verdict in $\mathbb{B}_3 = \{\top, \bot, ?\}$:
  - $\top$ (true): run complies with specification
  - $\bot$ (false): run does not comply with specification
  - ? (undetermined): verdict cannot be determined yet



MONITORING $\hookrightarrow$ System Abstraction

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

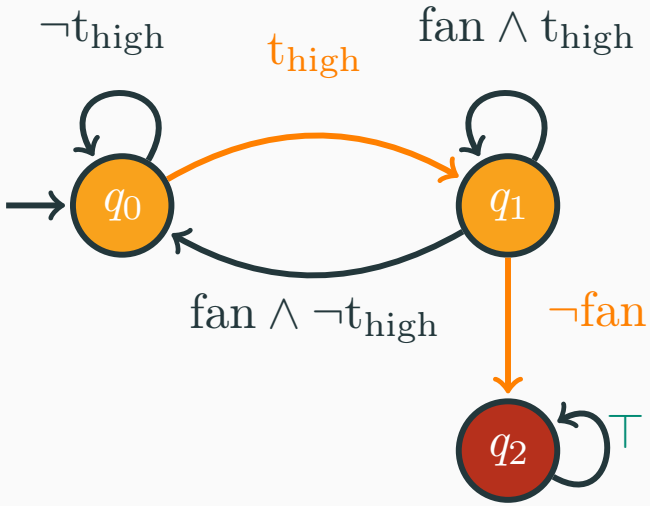
Example

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

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

MONITORING USING AUTOMATA ↪ EXAMPLE

"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	$\bot$

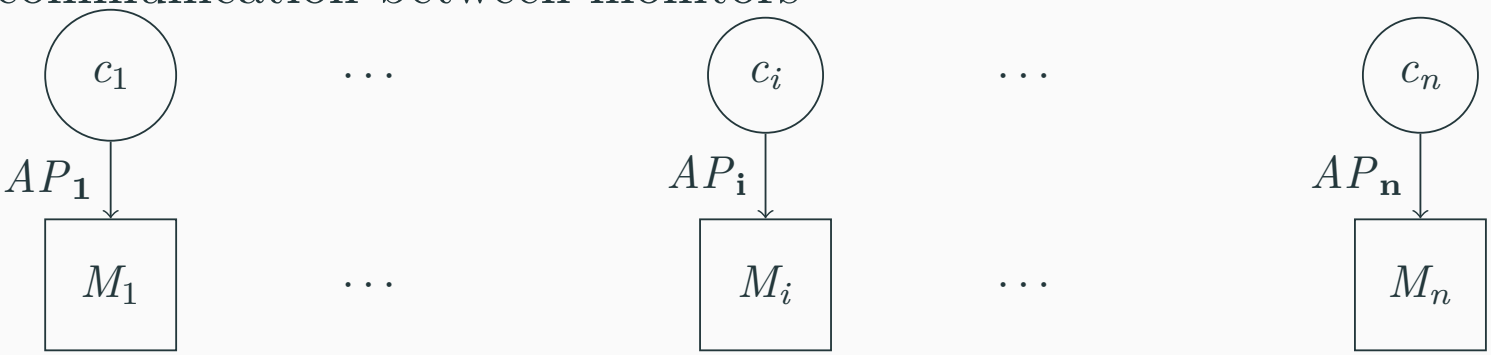
   - 2.2 Eval

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

Monitoring this property requires a central observation point!

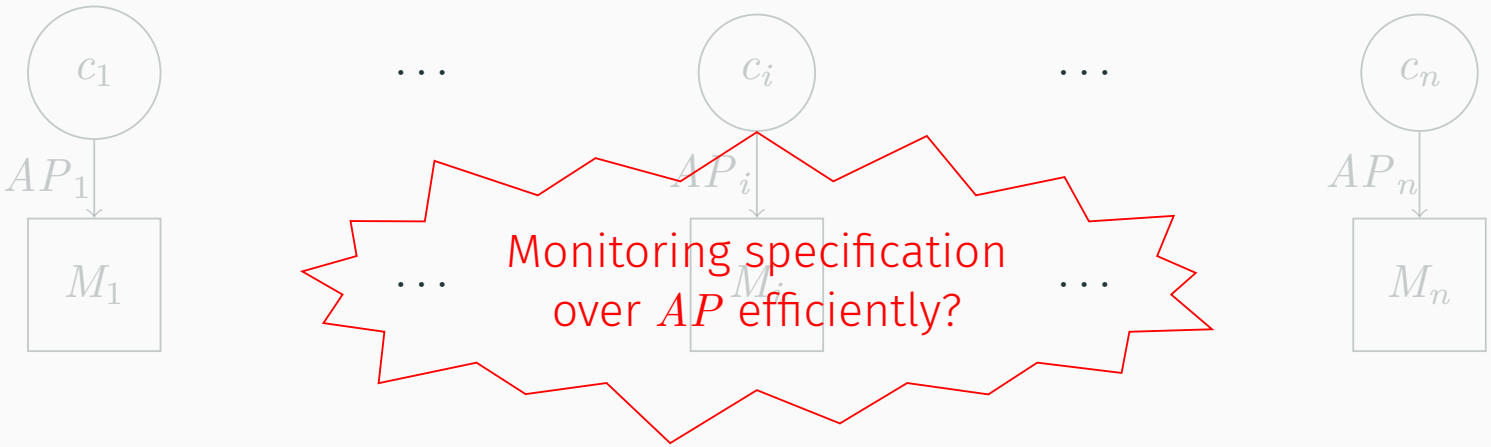
DECENTRALIZED MONITORING ↪ PROBLEM STATEMENT

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



DECENTRALIZED MONITORING ↪ 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



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

Experiments

Bringing Runtime Verification Home

Conclusions

Monitoring with EHEs

EXECUTION HISTORY ENCODING ↪ 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 → Atoms
  - Allow algorithms to add data to observations (enc : $AP \rightarrow Atoms$).
  - Ordering information (timestamp, round number, vector clock etc).
- Monitors store Atoms in their Memory
- Monitors need to evaluate $Expr_{Atoms}$
  - rewrite using Memory
  - simplify using Boolean logics (much easier than simplification for LTL)

$$Expr_{Atoms} \times Mem \rightarrow \mathbb{B}_3$$

$$\text{eval}(expr, \mathcal{M}) = \text{simplify}(\text{rw}(expr, \mathcal{M}))$$

$$\text{eval}(\langle 1, \text{t}_{\text{high}} \rangle \wedge \langle 2, \text{fan} \rangle, [\langle 1, \text{t}_{\text{high}} \rangle \mapsto \bot]) = \bot \wedge \langle 2, \text{fan} \rangle = \bot$$

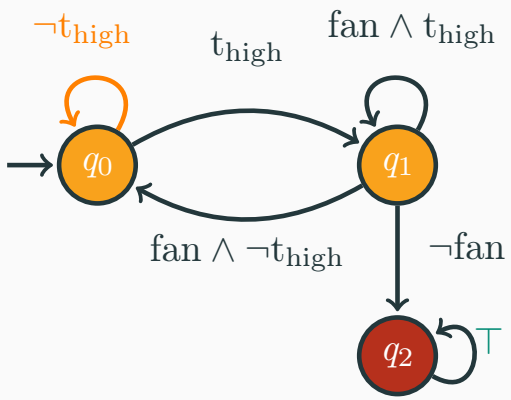
EXECUTION HISTORY ENCODING $\hookrightarrow$ AUTOMATA EXECUTION

- EHE is a partial function:

$$\mathcal{I} : \mathbb{N} \times Q_{\mathcal{A}} \to Expr_{Atoms}$$
$$\mathcal{I}(t, q) = expr$$

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

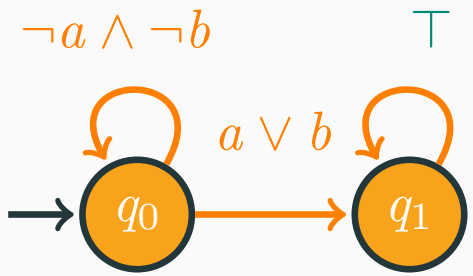
$$\mathcal{I}(2, q_0) = [\neg\langle 1, \text{t}_{\text{high}}\rangle \wedge \neg\langle 2, \text{t}_{\text{high}}\rangle]$$
$$\vee\, [\langle 1, \text{t}_{\text{high}}\rangle \wedge (\langle 2, \text{fan}\rangle \wedge \neg\langle 2, \text{t}_{\text{high}}\rangle)]$$
$$\text{eval}(\mathcal{I}(2, q_0), [\langle 1, \text{t}_{\text{high}}\rangle \mapsto \bot])$$
$$= \text{eval}(\neg\langle 2, \text{t}_{\text{high}}\rangle, \ldots) = ?$$



- EHE is constructed recursively & lazily (as needed and on-the-fly) using $\mathcal{A}$.

EXECUTION HISTORY ENCODING ↪ CONSTRUCTION

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



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	$[(\neg\langle 1, a\rangle \wedge \neg\langle 1, b\rangle) \wedge (\langle 2, a\rangle \vee \langle 2, b\rangle)] \vee [(\langle 1, a\rangle \vee \langle 1, b\rangle) \wedge \top]$

$\vdots$

EXECUTION HISTORY ENCODING ↪ PROPERTIES

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

   → They find the same verdict.

2. Strong Eventual Consistency
   - We can merge EHEs by disjoining ($\vee$) each entry $\langle t, q \rangle$.
   - $\vee$ is commutative, associative and idempotent.

   → EHE is a state-based replicated data-type (CvRDT) [Shapiro].

   → Monitors that exchange their EHE find the same verdict.

   → 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.

   → We can assess algorithms by how they manipulate the EHE.

EXECUTION HISTORY ENCODING $\hookrightarrow$ ANALYSIS

- Information delay (δ):
  - expanded timestamps with no state determined;
  - potential states to keep track of.
- Size of expressions grows with each move beyond t.
- Size of one expression $S(t')$, $t' > t$:

$$S(t') = |Q| \times (S(t'-1) + L)$$
$$= \mathcal{O}(|Q|^{t'}).$$

- Size of EHE:

$$|\mathcal{I}^{\delta}| = \mathcal{O}\left(\delta \times |Q| \times L \times |Q|^{\delta}\right).$$

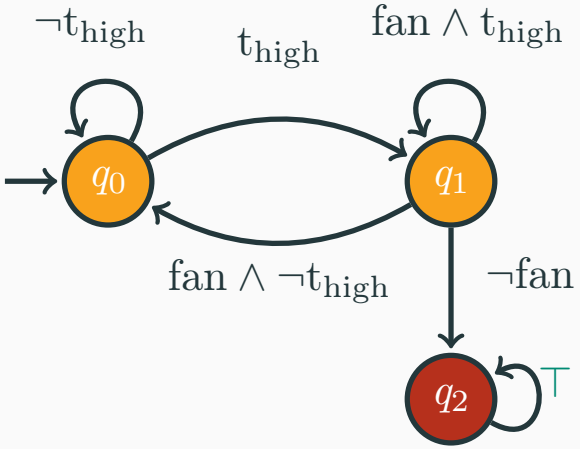
$$t \mapsto q \mapsto \top$$

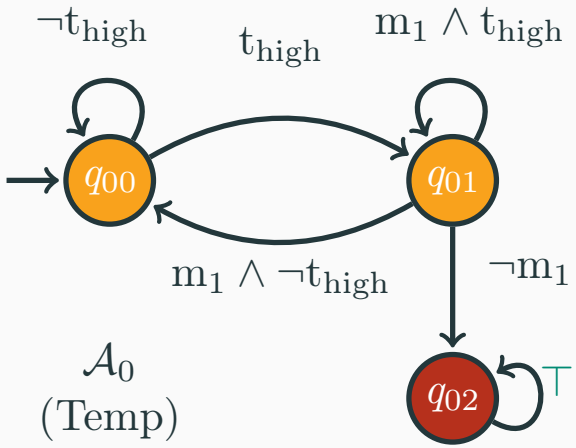
$$\delta \begin{cases} t+1 \mapsto \begin{cases} q_0 \mapsto e_{10} \\ q_1 \mapsto e_{11} \\ \vdots \\ q_{|Q|-1} \mapsto e_{1(|Q|-1)} \end{cases} & |Q| \\ t+2 \mapsto \begin{cases} q_0 \mapsto e_{20} \\ \vdots \\ q_{|Q|-1} \mapsto e_{2(|Q|-1)} \end{cases} & |Q| \\ \vdots \\ t+\delta \mapsto \begin{cases} q_0 \mapsto e_{\delta 0} \\ q_1 \mapsto e_{\delta 1} \\ \vdots \\ q_{|Q|-1} \mapsto e_{\delta(|Q|-1)} \end{cases} & |Q| \end{cases}$$

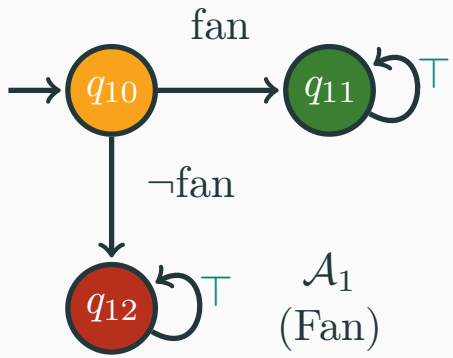
Monitoring Decentralized Specifications

DECENTRALIZED SPECIFICATIONS

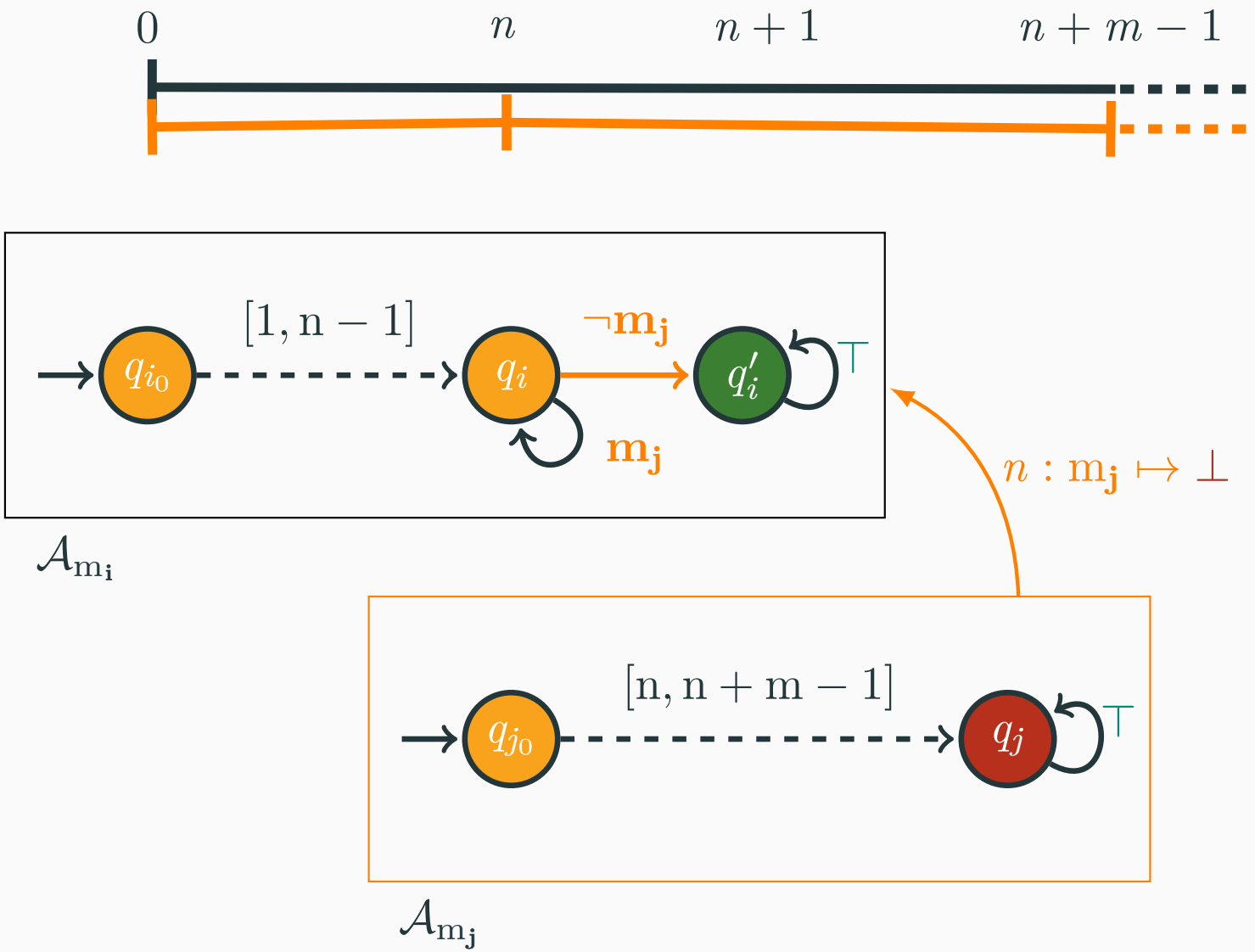
- A single automaton → Set of automata/monitors (Mons).
- Each monitor is associated with a component ($\mathcal{L} : \text{Mons} \to \mathcal{C}$).
- Set of references to monitors (atomic propositions) (AP_{mons})
- The transition labels of an automaton $m \in \text{Mons}$ are restricted to:
  - Atomic propositions local to the attached component ($\mathcal{L}(m)$).
  - References to other monitors.







DECENTRALIZED SPECIFICATIONS ↪ EVALUATING REFERENCES/SEMANTICS



★ Managing buffering and potential states using EHE.

PROPERTIES OF DECENTRALIZED SPECIFICATIONS ↪ MONITORABILITY

- Monitorability: "Is a given specification monitorable?"
- Non-monitorable $\implies$ monitors will never yield a verdict
- [Pnueli] For any (finite) trace t, does there exist a continuation t' s.t. $t \cdot t'$ yields a final verdict?
- Monitorability of automata: are all states co-reachable to states labeled by final verdicts?
  - Necessary & sufficient
  - Decidable in $\mathcal{O}(|Q| + |\delta|)$ time (quadratic in $|Q|$ worst-case).
- Decentralized specification: needs to account for dependencies.
  1. Every automaton must be monitorable; and
  2. Graph of monitor dependencies has no cycle.
  3. Decidable: cycle detection (monitor dependency graph, DFS/SCC)
  4. This is (only) a sufficient condition.
     (boolean simplification can eliminate dependencies: $\mathrm{s} \vee \mathrm{m}_1$)

GENERAL MONITORING ALGORITHM ↪ OVERVIEW

- Generalizes existing algorithms for decentralized monitoring of LTL/automata specifications.
- 2 stages: setup and monitoring.

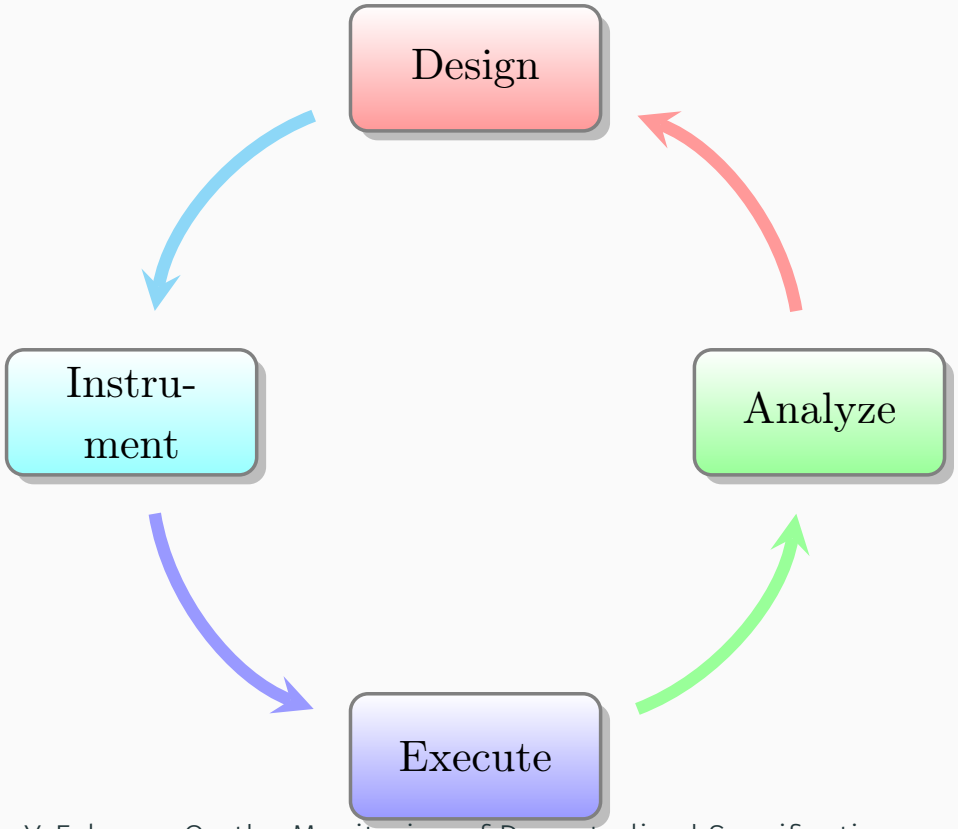
1. Setup (Deploy)
   - 1.1 Analyze and convert the specification as necessary.
   - 1.2 Create monitors and assign them a specification.
     - (!) The monitor handles encoding of AP and Memory.
   - 1.3 Attach monitors to components.
2. Monitoring
   - 2.1 Wait to receive observations from attached component.
   - 2.2 Receive messages (EHE or verdicts) from monitors.
   - 2.3 Process observations and messages (update the local EHE).
   - 2.4 Communicate with other monitors.

The THEMIS Approach

THEMIS ↪ Overview

Java and AspectJ implementation (5,700 LOC).

- Library: all necessary building blocks to develop, simulate, instrument, and execute decentralized algorithms.
- Command-line tools: basic functionality to generate traces, execute a monitoring run and execute a full experiment (multiple parametrized runs).



Design	Design a monitoring algorithm
Instru-ment	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

```
1  Map<Integer, ? extends Monitor>
   ↪  setup() {
2    config.getSpec().put("root",
3        Convert.makeAutomataSpec(
4          config.getSpec().get("root")));
5    Map<Integer, Monitor> mons = new
     ↪  HashMap<Integer, Monitor>();
6    Integer i = 0;
7    for(Component comp :
     ↪  config.getComponents()) {
8      MonMigrate mon = new
       ↪  MonMigrate(i);
9      attachMonitor(comp, mon);
10     mons.put(i, mon);
11     i++;
12   }
13   return mons;
14 }
```

Monitor

```
1  void monitor(int t, Memory<Atom> observations)
2  throws ReportVerdict, ExceptionStopMonitoring {
3      m.merge(observations);
4      if(receive()) isMonitoring = true;
5      if(isMonitoring) {
6        if(!observations.isEmpty())
7          ehe.tick();
8        boolean b = ehe.update(m, -1);
9        if(b) {
10         VerdictTimed v = ehe.scanVerdict();
11         if(v.isFinal())
12           throw new
             ↪  ReportVerdict(v.getVerdict(), t);
13         ehe.dropResolved();
14       }
15       int next  = getNext();
16       if(next != getID()) {
17         Representation toSend = ehe.sliceLive();
18         send(next, new
           ↪  RepresentationPacket(toSend));
19         isMonitoring = false;
20       }
21     }
22   }
```

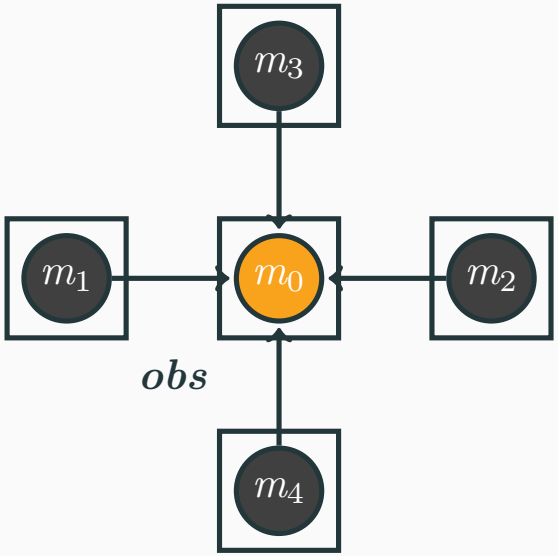
EXAMPLES ↪ METRICS

```
1 void setupRun(MonitoringAlgorithm alg) {
2  addMeasure(new Measure("msg_num","Msgs",0L,Measures.addLong));
3 }
4 after(Integer to, Message m) : Commons.sendMessage(to, m) {
5  update("msg_num" , 1L);
6 }
```

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

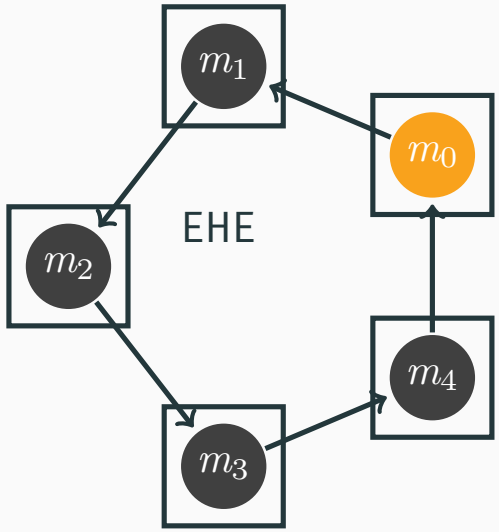
	alg	comps	avg(msg_num)	avg(msg_data)	count(*)
1	Migration	3	2.04226336011177	267.8458714635	572600
2	Migration	4	2.16402472527473	668.129401098901	364000
3	Migration	5	3.33806822465134	3954.09705050886	530600
4	MigrationRR	3	32.7222301781348	482.572275585051	572600
5	MigrationRR	4	31.8533351648352	932.708425824176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

STUDYING EXISTING ALGORITHMS ↪ PRINCIPLES OF THE ALGORITHMS



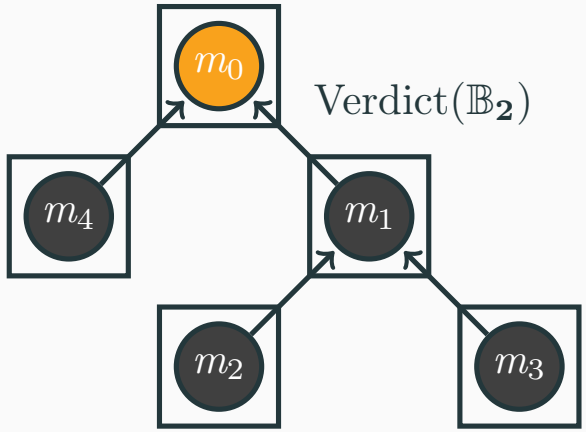
Orchestration

- one central monitor
- observations are forwarded to the central monitor



Migration

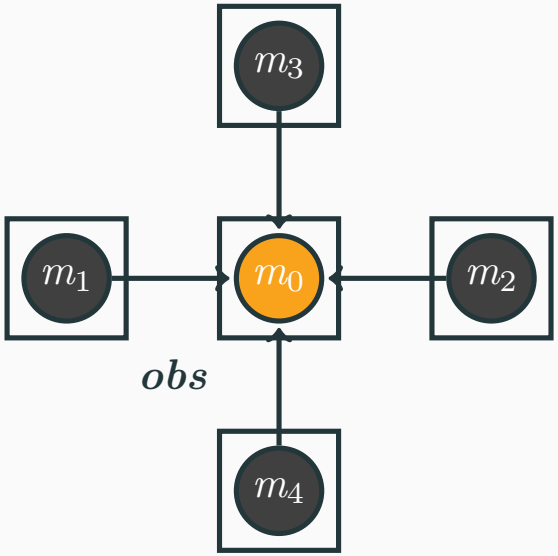
- monitor state “hops"
- monitor updates it with local information
- forward to the next monitor

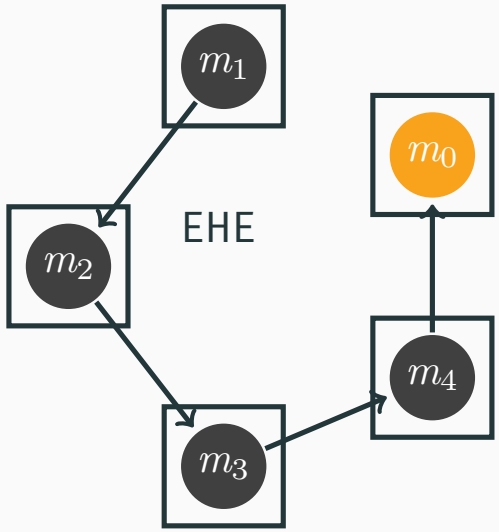


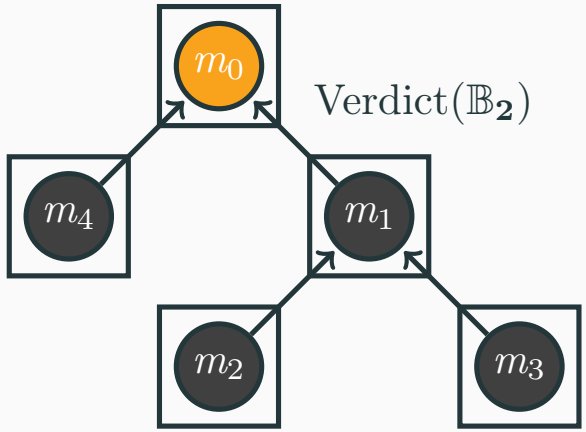
Choreography

- DAG of monitors
- a monitor evaluates a sub-specification
- verdict propagates in the DAG

Studying existing algorithms ↪ Expected Behavior







Orchestration

- δ is constant
- #Msgs is linear in components
- |Msg| constant: observations per component

Migration

- δ is linear in components
- #Msgs is constant
- |Msg| is size of EHE: exponential in components

Choreography

- δ is linear in network depth (split algorithm)
- #Msgs is linear in network edges
- |Msg| is constant

EXISTING ALGORITHMS

GraphStream

LIG cnrs Inria Université Grenoble Alpes

Experiments

Studying existing algorithms ↪ Verifying Behavior

Simulate the behavior of orchestration, migration, and choreography.

Confirm the trends predicted by the analysis.

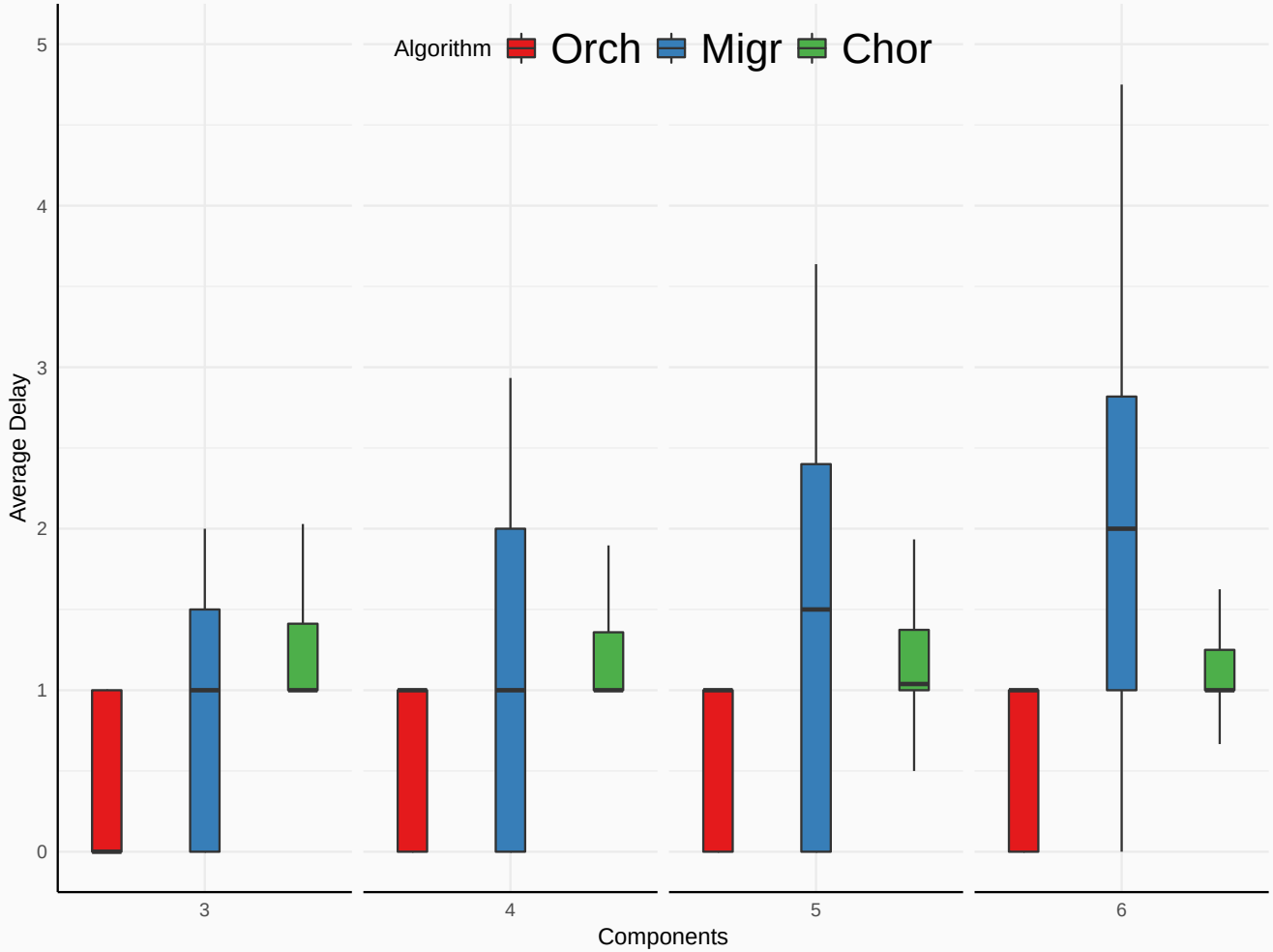
Experiment Setup ($5,868,800$ runs): [1]

- 200 synthetic random traces of 100 events (2 observations/component).
- Vary $|\mathcal{C}|$ from 3 to 5.
- At least 1,000 random specifications per scenario.

[1] More experiments and results in paper:

- several probability distributions for events,
- more metrics,
- a case study on the Chiron UI.

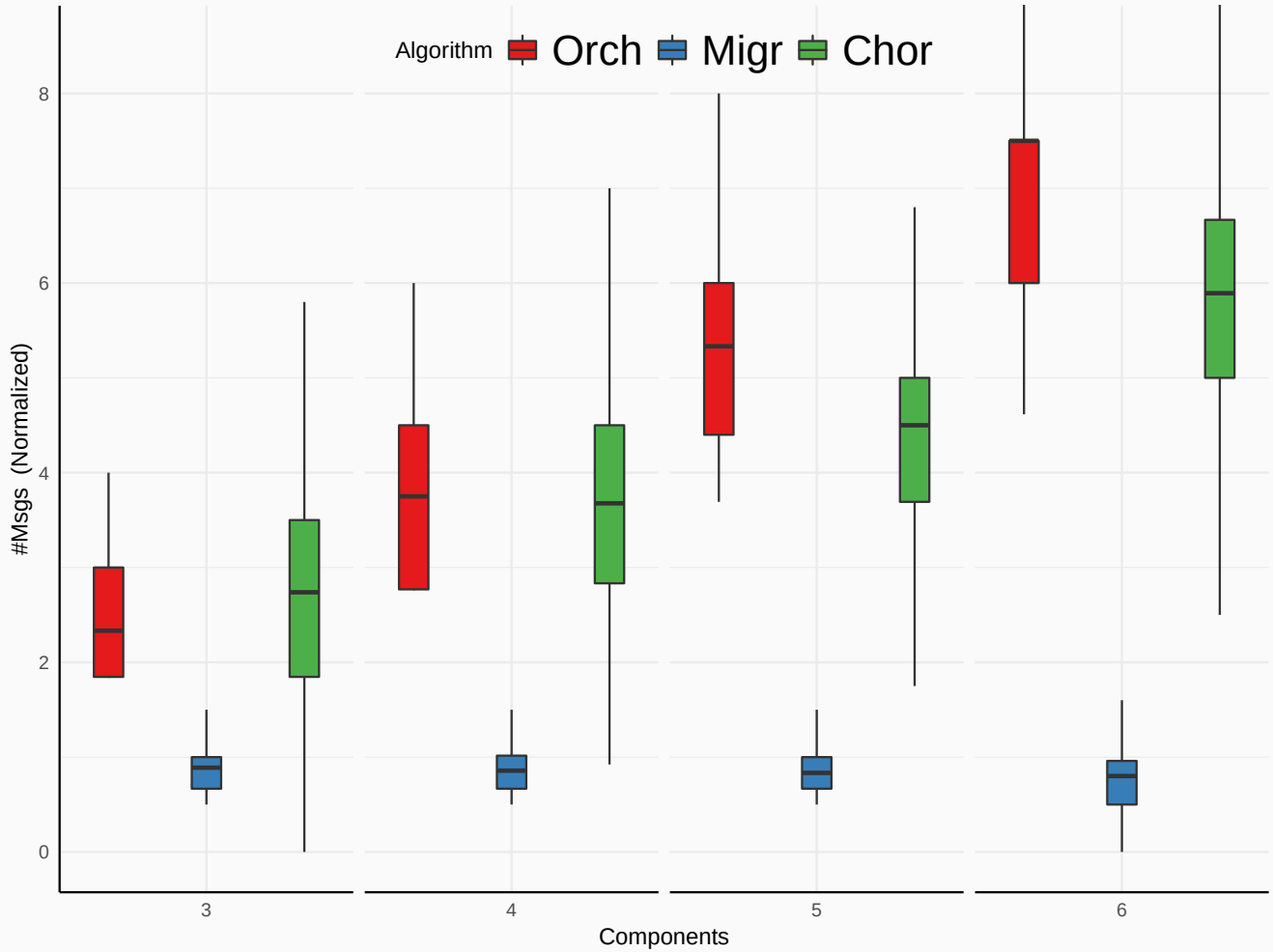
RESULTS ↪ DELAY



Recall from the analysis:

- Orchestration is constant.
- Migration is linear in components.
- Choreography is linear in network depth.

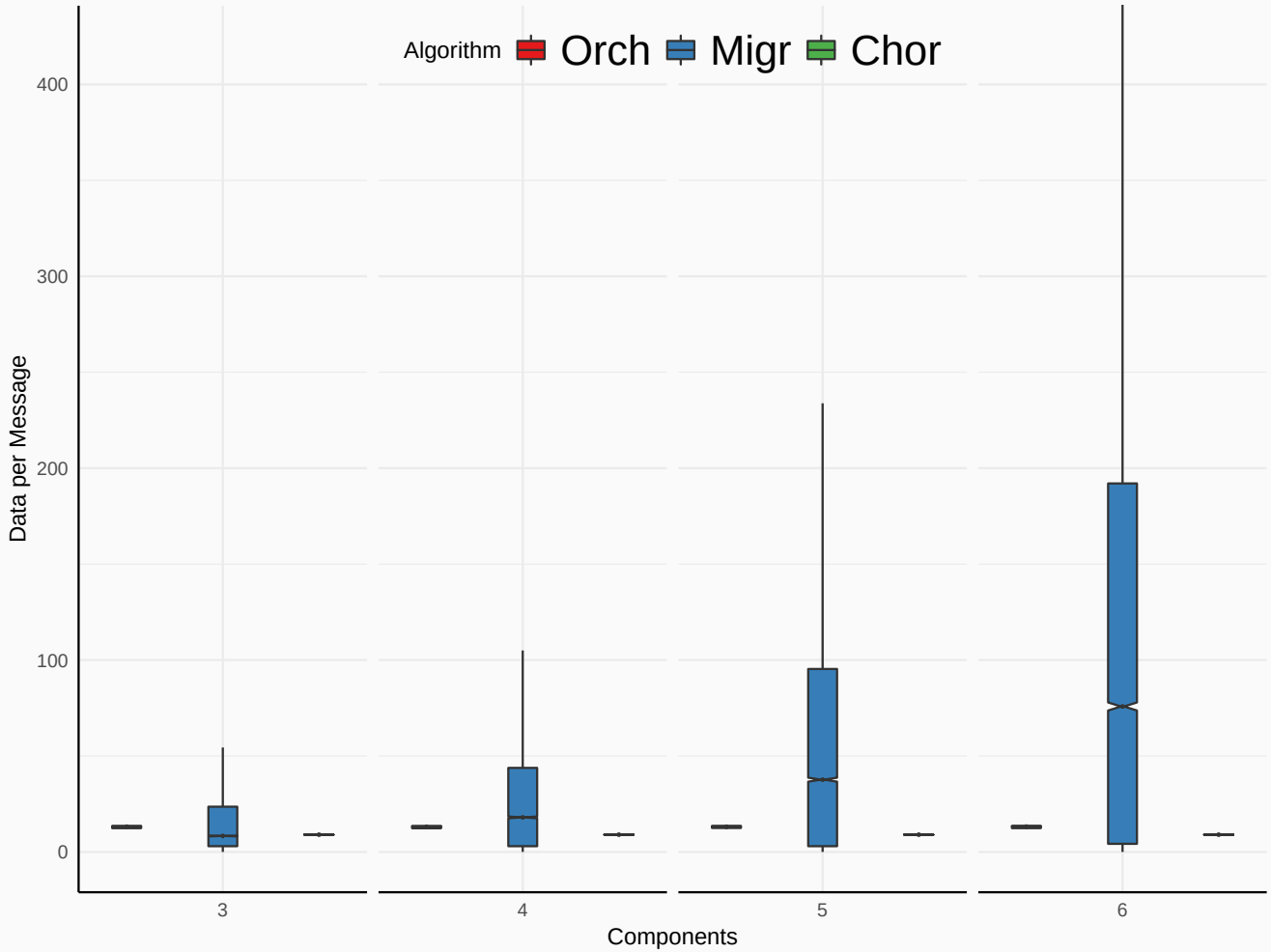
RESULTS ↪ NUMBER OF MESSAGES



Recall from the analysis:

- Orchestration is linear in components.
- Migration is constant.
- Choreography is linear in network edges.

RESULTS ↪ DATA TRANSFERRED



Recall from the analysis:

- Orchestration is constant.
- Migration is exponential in components.
- Choreography is constant.

RESULTS (AVERAGE VALUES)

Alg.	$\lvert\mathcal{C}\rvert$	δ	#Msgs	Data	#S	#S/Mon	Conv
Chor	3	2.37	2.02	18.05	15.27	6.63	0.18
	4	2.49	2.54	22.62	18.22	6.79	0.20
	5	2.37	3.08	27.18	21.29	6.95	0.22
Migr	3	1.02	0.36	49.46	4.80	4.80	1.00
	4	1.38	0.41	128.26	5.67	5.67	1.00
	5	2.28	0.57	646.86	9.40	9.40	1.00
Migrr	3	1.09	0.86	58.02	5.00	5.00	1.00
	4	1.49	0.85	144.62	5.91	5.91	1.00
	5	2.32	0.83	684.81	9.60	9.60	1.00
Orch	3	0.63	1.68	21.01	4.13	4.13	1.00
	4	0.65	2.43	30.42	4.11	4.11	1.00
	5	0.81	3.04	38.51	5.55	5.55	1.00

Lower Conv $\Longrightarrow$ more evenly distributed computation across monitors

Bringing Runtime Verification Home

MONITORED ENVIRONMENT

Amiqual4Home[2]

Experimental platform consisting of a smart apartment, a rapid prototyping platform, and tools for observing human activity.

- Hierarchical setup: 2 floors, 7 rooms, 219 sensors.
- Existing public datasets of full sensors traces (Orange4Home, ContextAct@A4H).
- Databases are annotated with user activities.

Monitoring Context

- 22 specifications written for up to 27 sensors.
- Traces from 07:30 to 17:30 (36,000 timestamps) from Orange4Home.

[2] amiqual4home.inria.fr

PROPERTY GROUPS

- System Properties: ensure system working properly
  - Verify Light Switches (each room i + global house)

$$\text{sc\_light}(i) \stackrel{\text{def}}{=} \square(\text{switch}_i \implies \text{X}(\text{light}_i \,\text{U}\, \neg\text{switch}_i)), i \in [0..n]$$

$$\text{sc\_ok} \stackrel{\text{def}}{=} \bigwedge_{i \in [0..n]} \text{sc\_light}(i)$$

- Activities of Daily Living (ADL): detecting user behavior
  - Formalize an activity as a property over sensors output.
  - A knowledge-based approach (vs. Machine Learning approaches).
  - Examples: sleeping, cooking, watching tv.
- Meta-Properties: properties of other Properties
  - Properties that are defined on top of other properties.
  - firehazard $\stackrel{\text{def}}{=} \square(\text{napping} \implies \neg\text{cooking})$

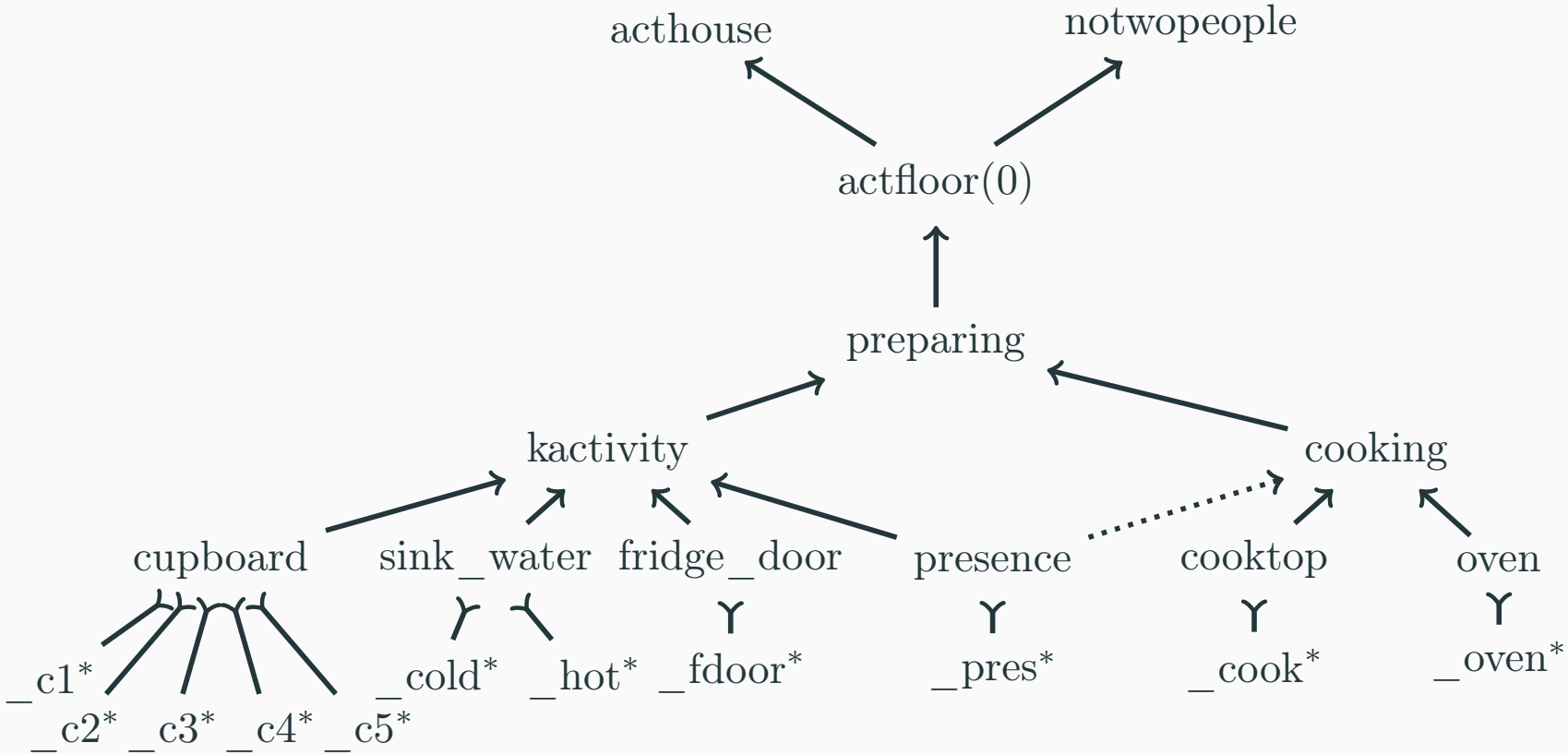
PROPERTIES ↪ EXAMPLE PROPERTIES

ADL	m_toilet	toilet_water
	sink_usage	$\Box_3(\text{m\_bathroom\_sink\_water})$
	bathroom_sink	$\text{bathroom\_sink\_cold} \vee \text{bathroom\_sink\_hot}$
	shower_usage	$\Box_2(\text{m\_bathroom\_shower\_water})$
	napping	$\Box_{25}(\text{m\_bedroom\_bed\_pressure})$
	dressing	$\Diamond_4(\text{m\_bedroom\_closet\_door} \vee \text{m\_bedroom\_drawers}))$
	reading	$\text{m\_bedroom\_light} \wedge \Diamond_4(\neg\text{dressing} \wedge \neg\text{napping})$
	office_tv	$\Diamond_3(\text{m\_office\_tv})$
	computing	$\Diamond_3(\text{m\_office\_deskplug})$
	livingroom_tv	$\Diamond_3(\text{m\_livingroom\_tv} \wedge \text{m\_livingroom\_couch})$
	eating	$\neg\text{m\_kitchen\_presence} \wedge \Box_6(\text{m\_livingroom\_table})$
Meta	actfloor(0)	$\text{cooking} \vee \text{preparing} \vee \text{eating} \vee \text{washing\_dishes} \vee \text{livingroom\_tv} \vee \text{m\_toilet}$
	acthouse	$\text{actfloor}(0) \vee \text{actfloor}(1)$
	notwopeople	$\neg(\text{actfloor}(0) \wedge \text{actfloor}(1))$
	firehazard	$\text{napping} \implies \neg\text{cooking}$

DECENTRALIZATION ↪ TAKING ADVANTAGE OF HIERARCHIES

1. Abstraction/Modularity
   - 1.1 Sub-specifications are building blocks for more complex specifications.
   - ⋆ (Meta) Specifications of specifications.
   - 1.2 Change (or refine) existing sub-specifications without changing those that depend on them.
   - 1.3 Abstraction from Implementation: references should eventually return a verdict.
2. Scalability/Efficiency
   - 2.1 Factor the monitoring cost of sub-specifications.
   - 2.2 Smaller automata/formulae to represent complex inter-dependent specifications (Monitor Synthesis).
   - 2.3 Manage duplication of computation and computation.
   - 2.4 Communication modeled by dependencies.
   - 2.5 Monitor placement can be optimized for system architecture.

Decentralized Specifications ↪ Dependency Hierarchies & Reference



+ Reduction of atomic propositions and size of specifications
+ Re-use: no need to recompute same dependencies
+ Abstraction: references hides implementation

MONITOR SYNTHESIS ↪ ATOMIC PROPOSITIONS

- Synthesizing monitors is doubly exponential.
  1. Number of atomic propositions
  2. Size of formula

★ Goal: Reference Sub-specifications
- Reduce number of atomic propositions ($|\mathrm{AP}|^{\mathrm{d}} < |\mathrm{AP}|^{\mathrm{c}}$)
- Reduce formula size (atomic proposition instead of formula)

Name	$\lvert\mathrm{AP}\rvert^{\mathrm{d}}$	$\lvert\mathrm{AP}\rvert^{\mathrm{c}}$	d
sc_light(i)	2	2	1
sc_ok	4	8	2
toilet*	1	1	0
sink_usage	1	2	1
napping	1	1	1
dressing	2	3	1
reading	3	5	2
kactivity*	4	9	1
preparing	2	11	2
actfloor(0)	6	16	3
actfloor(1)	7	11	3
acthouse	2	27	4
notwopeople	2	27	4
firehazard	2	3	2

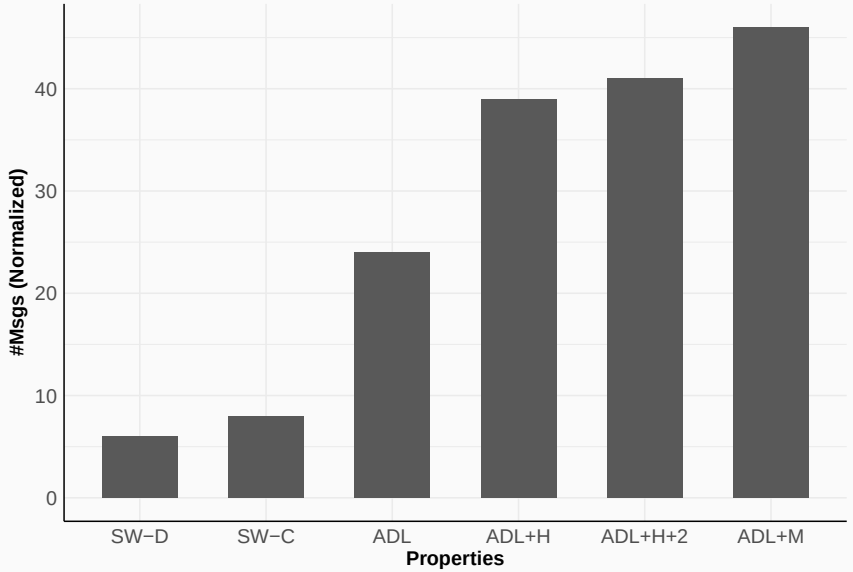
RE-USING COMPUTATION AND COMMUNICATION

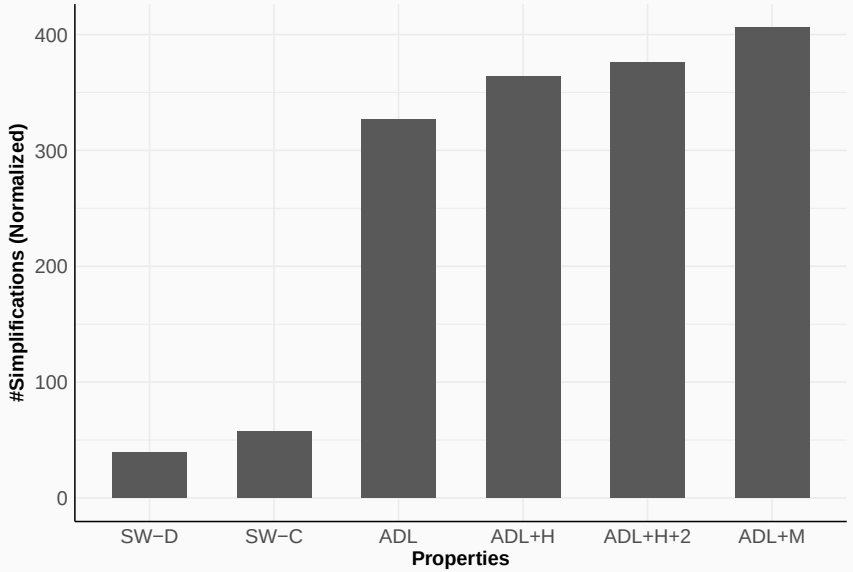
- A shared sub-specification is monitored once.
- Higher-up specifications do not need the sub-specification sensors.
  - Centralized sc_ok (SW-C) uses 8 sensors instead of 6 for decentralized (SW-D).
- Adding meta-properties incurs less overhead due to re-use.

ADL	All ADL properties (baseline)
ADL+H	ADL + actfloor(i) ($i \in [0..1]$), acthouse
ADL+H+2	ADL+H + notwopeople
ADL+M	All meta properties

$$\text{acthouse} \stackrel{\text{def}}{=} \text{actfloor}(0) \vee \text{actfloor}(1)$$

$$\text{notwopeople} \stackrel{\text{def}}{=} \neg(\text{actfloor}(0) \wedge \text{actfloor}(1))$$

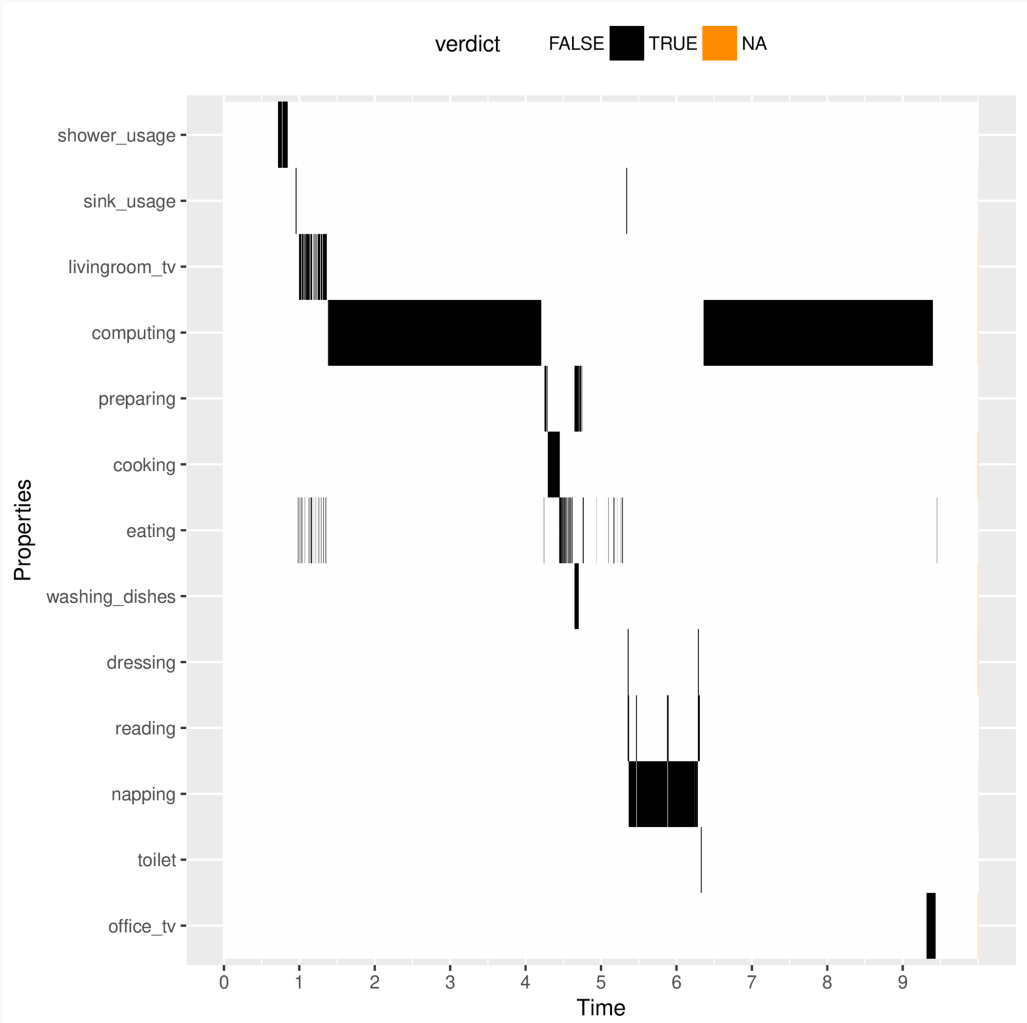




SCHEDULE

Suggested (left) vs Reconstructed (right)

08:00	Entering	Entrance	12:00	Eating	Livingroom
	Up	Staircase		Dishes	Kitchen
	Shower-ing	Bathroom		Cleaning	Kitchen
			13:00	Up	Staircase
	Sink	Bathroom		Sink	Bathroom
08:30	Down	Staircase	13:15	Dressing	Bedroom
	TV	Livingroom		Reading	Bedroom
09:00	Up	Staircase	13:45	Napping	Bedroom
	Comput-ing	Office		Dressing	Bedroom
			14:00	Comput-ing	Office
			16:30	TV	Office
11:30	Down	Staircase		Down	Staircase
	Prepar-ing	Kitchen	17:00	Leaving	Entrance
	Cooking	Kitchen			



HOW GOOD IS OUR METHOD AT DETECTING ADL?

★ Depends on Property
- Availability of sensors
- Rigidity of specification

- Examples:

toilet: only water usage $\Longrightarrow$ low recall
reading: no sensors $\Longrightarrow$ inferred from others
napping: changing specification

Formula	Precision	Recall	F1
$\Box_{25}(\text{weight})$	0.43	0.95	0.60
$\Box_{3}(\text{weight})$	0.43	0.99	0.60
$\Diamond_{3}(\text{weight})$	0.43	1.0	0.60
$\Box_{3}(\text{pres} \wedge \text{weight})$	0.34	0.14	0.20
$\Box_{3}(\neg\ell \wedge \text{weight})$	1.00	0.97	0.99

weight: bed pressure sensor
pres: bedroom presence sensor
ℓ: bedroom light sensor

Property	Precision	Recall	F1
computing	0.98	0.99	0.99
office_tv	1.00	0.80	0.89
cooking	0.88	0.88	0.88
shower_usage	1.00	0.50	0.67
washing_dishes	1.00	0.47	0.64
livingroom_tv	1.00	0.43	0.60
dressing	1.00	0.41	0.58
toilet*	1.00	0.18	0.30
sink_usage	1.00	0.13	0.23
eating	0.61	0.35	0.44
napping	0.43	0.95	0.60
preparing	0.23	0.77	0.35
reading	0.37	0.04	0.06

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.
5. Application to smart homes.

★ Future Work

1. Centralised specification → equivalent decentralized specifications.
   - Optimize existing methods.
   - Take into account topology of the monitored system.
2. Extend THEMIS (metrics, better visualization of algorithm behavior).
3. Runtime enforcement of centralized and decentralized specifications.


11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings. IEEE Computer Society (1999)


Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science 113, 145 – 162 (2005)


Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014. IEEE (2014)


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, Proceedings, Lecture Notes in Computer Science, vol. 8461. Springer (2014)


Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.): Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games, Lecture Notes in Computer Science, vol. 5125. Springer (2008)


Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N. (eds.): Runtime Verification - First International

Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6418. Springer (2010)


Bartocci, E.: Sampling-based decentralized monitoring for networked embedded systems. In: Bortolussi et al. [13], pp. 85–99


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)


Basin, D.A., Klaedtke, F., Zalinescu, E.: Failure-aware runtime verification of distributed systems. In: Harsha and Ramalingam [30], pp. 590–603


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 [29], pp. 85–100


Bonakdarpour, B., Fraigniaud, P., Rajsbaum, S., Travers, C.: Challenges in fault-tolerant distributed runtime verification. In: Margaria and Steffen [36], pp. 363–370


Bortolussi, L., Bujorianu, M.L., Pola, G. (eds.): Proceedings Third International Workshop on Hybrid Autonomous Systems, HAS 2013, Rome, Italy, 17th March 2013, EPTCS, vol. 124 (2013)


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)


Buchfuhrer, D., Umans, C.: The complexity of boolean formula minimization. In: Aceto et al. [5], pp. 24–35


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)


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. [37], pp. 1508–1515


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)


Diekert, V., Leucker, M.: Topology, monitorable properties and runtime verification. Theoretical Computer Science 537, 29 – 41 (2014), theoretical Aspects of Computing (ICTAC 2011)


Diekert, V., Muscholl, A.: On distributed monitoring of asynchronous systems. In: Ong and de Queiroz [39], pp. 70–84, 10.1007/978-3-642-32621-9_5


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)


El-Hokayem, A., Falcone, Y.: Themis website (2017), https://gitlab.inria.fr/monitoring/themis


Falcone, Y.: You should better enforce than verify. In: Barringer et al. [6], pp. 89–105


Falcone, Y., Cornebize, T., Fernandez, J.: Efficient and generalized decentralized monitoring of regular languages. In: Ábrahám and Palamidessi [4], 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: Engineering Dependable Software Systems, pp. 141–175 (2013)


Finkelstein, A., Estublier, J., Rosenblum, D.S. (eds.): 26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom. IEEE Computer Society (2004)


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)


Harsha, P., Ramalingam, G. (eds.): 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India, LIPIcs, vol. 45. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)


Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectj. In: Knudsen [33], pp. 327–353


Kim, M., Viswanathan, M., Ben-Abdallah, H., Kannan, S., Lee, I., Sokolsky, O.: Formally specified monitoring of temporal properties. In: 11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings [1], pp. 114–122


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)


Leucker, M., Schmitz, M., à Tellinghusen, D.: Runtime verification for interconnected medical devices. In: Margaria and Steffen [36], pp. 380–387


Margaria, T., Steffen, B. (eds.): Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II, Lecture Notes in Computer Science, vol. 9953 (2016)


Min, G., Hu, J., Liu, L.C., Yang, L.T., Seelam, S., Lefèvre, L. (eds.): 14th IEEE International Conference on High Performance Computing and Communication & 9th

IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27, 2012. IEEE Computer Society (2012)


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)


Ong, C.L., de Queiroz, R.J.G.B. (eds.): Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7456. Springer (2012)


Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra et al. [38], pp. 573–586


Rosu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Autom. Softw. Eng. 12(2), 151–197 (2005)


Scheffel, T., Schmitz, M.: Three-valued asynchronous distributed runtime verification. In: Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014 [3], pp. 52–61




Related Work and Goals

RELATED WORK ↪ DECENTRALIZED RV

- General setting
  - $\mathcal{C}$: a set of components
  - AP: a set of atomic propositions, partitioned by $\mathcal{C}$
- Issues in decentralized monitoring
  - partial views of AP – unknown global state
  - partial execution of the automaton (evaluation)
  - communication between monitors
- Rewriting-based techniques
  - (safety) LTL [Rosu et al 05], (full) LTL [BauerFalcone12,ColomboFalcone16]
  - (safety) MTTL (real-time systems) [ThatiRosu05,Basin et al 15]
  - Common assumptions
    - Reliable network with fully-connected components
    - Global clock
    - Oblivious to order of messages

(!) Unpredictable runtime behavior of rewriting

→ Hard to compare various strategies

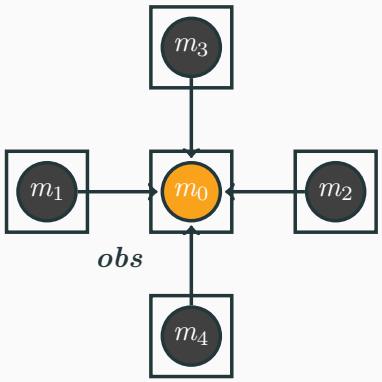
RELATED WORK ↪ DECENTRALIZED RV (CONT'D)

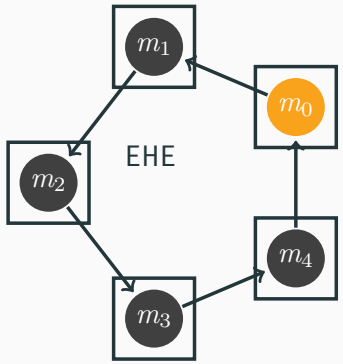
- Automata-based techniques for regular languages [Falcone et al 14]
  - Same assumptions as rewriting
  + More expressive than LTL
  + Predictable behavior
  – Tightly linked to specification (synthesis)
  – No monitor topology nor communication strategy
- Monitor Consensus [MostafaBonakdarpour16]
  - monitors deciding the same verdict
  - Assumptions
    - Fully-connected components
    - Asynchronous Systems (Alternating Numbers)
    + Unreliable links (Monitors + System)
    – $2k + 2$ verdicts when resilience up to k failures

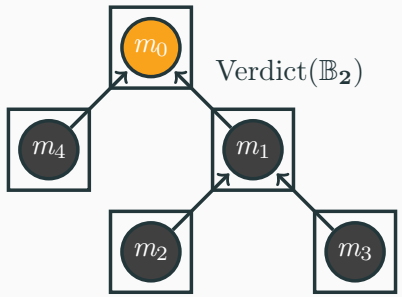
  → Determine consensus on a verdict in case of failures

(!) All monitors check the same specification

STUDYING EXISTING ALGORITHMS







- Example algorithms
  - Orchestration: Central monitor + forwarding monitors.
  - Migration: Specification hops from one component to another.
  - Choreography: Monitors are organized in a tree.
- Expected behavior of algorithms:

Algorithm	δ	# Msg	|Msg|
Orchestration	$\Theta(1)$	$\Theta(\|\mathcal{C}\|)$	$\Theta(\|AP_c\|)$
Migration	$\mathcal{O}(\|\mathcal{C}\|)$	$\mathcal{O}(m)$	$\mathcal{O}(\|Q\|^{\|\mathcal{C}\|})$
Choreography	$\mathcal{O}(\text{depth}(\text{rt}) + \|\text{tr}\|)$	$\Theta(\|E\|)$	$\Theta(1)$