On Decentralized Monitoring

Yliès Falcone
(www.ylies.fr)

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

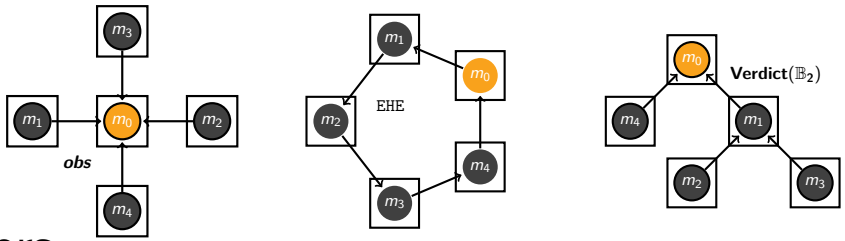
22 November 2021

Based on work with Andreas Bauer, Christian Colombo, and Antoine El-Hokayem.

Talk overview

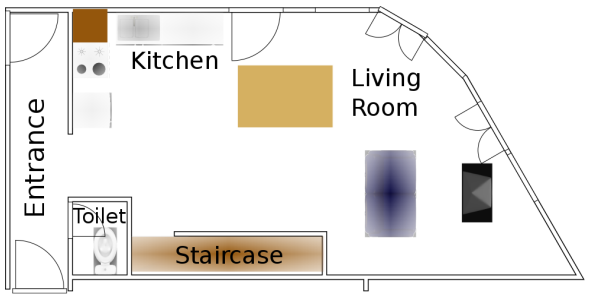
Part I: Approaches to monitoring spatially distributed systems

- Specifications in Linear-time Temporal Logic and as finite-state automata
- Organization of monitors:
  - orchestration: master/slave monitors
  - migration: "flat" monitors
  - choreography: hierarchical monitors
- Generalization of existing algorithms
- Decentralized specifications



Part II: Bringing Monitoring Home

- Monitoring a real smart house
- Challenges for monitoring
- Effectiveness of monitoring techniques



Perspectives and research opportunities

Agenda

Background and Motivations

Some Approaches to Decentralized Monitoring

Generalization: Monitoring Decentralized Specifications with Execution History Encodings

The `THEMIS` Approach

Bringing Runtime Verification Home

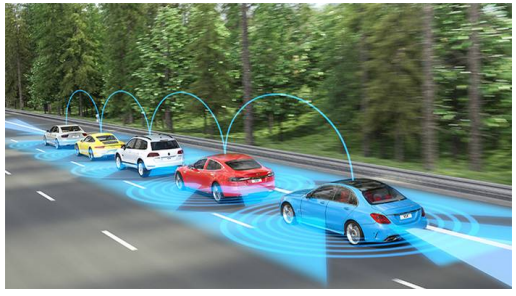
Conclusions

Agenda

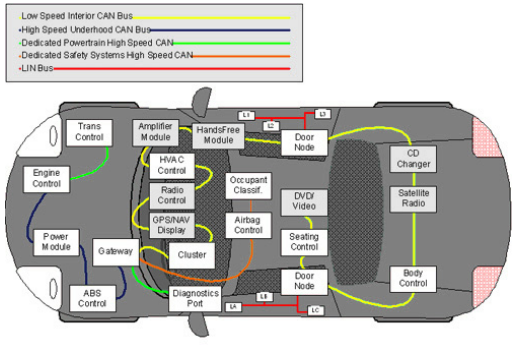
Systems tend to be more decentralized and distributed

"Smart" Anything

Car platoon



Modern car



Decentralized Finance

Decentralization/distribution is desirable for operational and security reasons . . . or comes by design

Exhaustive (static) verification of such systems is often impossible

Considered system: a smart apartment

Amiqual4Home[1]

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

Verification Context

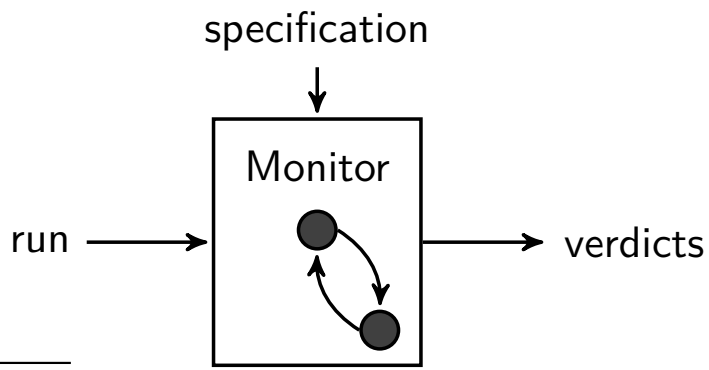
- 22 specifications written for up to 27 sensors
  - system behavior
  - activity of daily living
  - meta specifications
- Traces from 07:30 to 17:30 (36,000 timestamps) from Orange4Home

[1] amiqual4home.inria.fr

Monitoring (aka Runtime Verification)

Overview

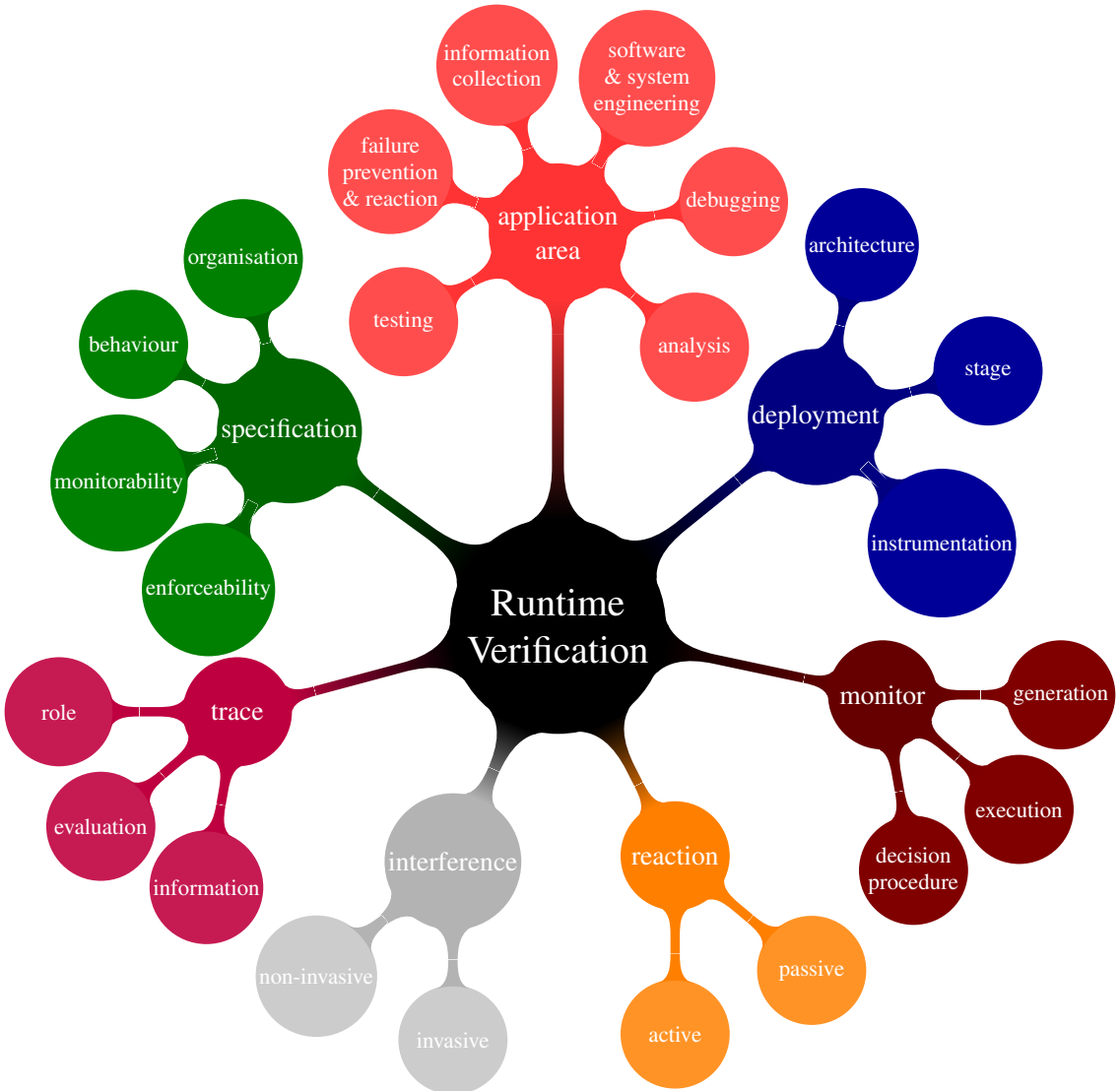
- Lightweight verification technique.
- Checks whether a run of a system conforms to a specification (Incomplete, as opposed to exhaustive verification techniques)
- Specification is formalized
- Monitors are synthesized and integrated to observe the system
- Monitors determine a verdict in $\mathbb{B}_3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



Lectures on Runtime Verification - Introductory and Advanced Topics. Bartocci, Falcone. Lecture Notes in Computer Science 10457, Springer 2018

Monitoring (aka Runtime Verification)

Overview (ctd)



A taxonomy for classifying runtime verification tools. Falcone, Krstic, Reger, Traytel. Int. Journal on Software Tools for Technology Transfer volume 23, pages 255–284 (2021)

Monitoring

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

Specifications in Runtime Verification

- Formally expressed specifications
  - Automata
  - Logics: LTL, MTL, etc.
  - (!) Possible: LTL $\rightarrow$ Büchi $\rightarrow$ Moore Automaton

Example (Specifications)

1. "Temperature should never reach critical": $\boldsymbol{G}(\neg \mathtt{t}_{\mathtt{crit}})$
2. "Fan must cool the environment": $\boldsymbol{G}(\mathtt{fan}\, \boldsymbol{U}\, \mathtt{t}_{\mathtt{low}})$
3. "Fan must always be turned on when temperature is high": $\boldsymbol{G}(\mathtt{t}_{\mathtt{high}} \implies \boldsymbol{X}\mathtt{fan})$

(!) Can span multiple components

Monitoring using Automata

Example

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

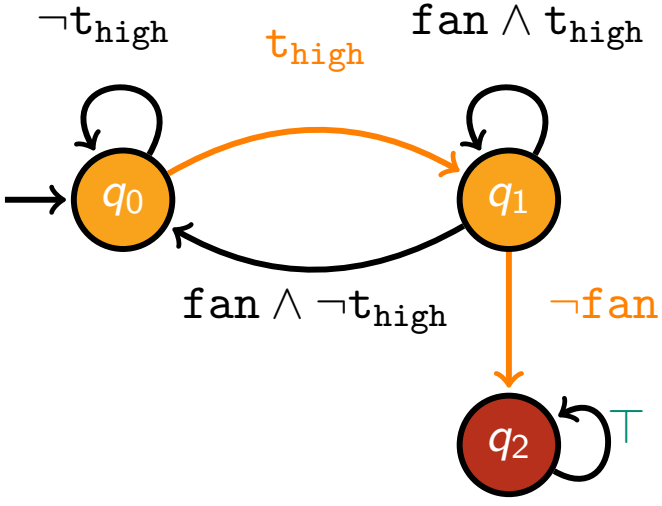


Figure: *

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

1. At $t = 1$, from q_0:
   - 1.1 Observe

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

   - 1.2 Eval

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

2. At $t = 2$, from q_1:
   - 2.1 Observe

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

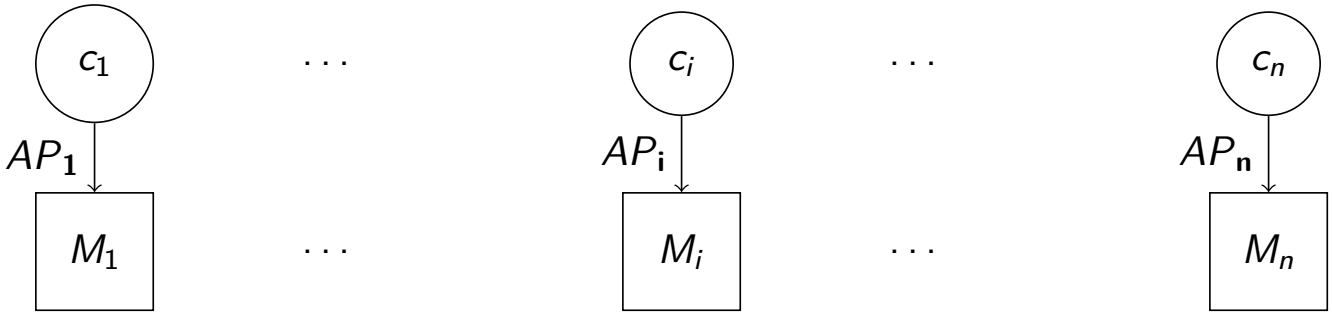
   - 2.2 Eval

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

Monitoring this property requires a central observation point!

Decentralized Monitoring: Problem statement

- General setting
  - $\mathcal{C} = \{c_1, \ldots, c_n\}$: components
  - $AP = AP_1 \cup \ldots \cup AP_n$: 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
- Requirements for online monitoring:
  - efficiency in monitor computation and communication
  - predictability of monitor performance



Assumptions

- Local behavior is sufficiently observable.
- Existence of a **global clock**
  - local monitors awareness
  - realistic in several industrial critical systems
  - can be achieved in a distributed systems with synch. algorithms

Automotive domain uses *FlexRay* data bus, which has (among others) a synchronous transfer mode:

Examples: Steer-by-wire, brake-by-wire, engine management, etc.

Flight-control systems mostly synchronous (fly-by-wire):

SIGNAL, Lustre, Astrée verifier, etc.

- Monitors can *directly communicate with each others* in a reliable fashion but possibly out of order.

Traian Pop et al. "Timing analysis of the FlexRay communication protocol". In: Real-Time Syst. 39 (1–3 2008). Gunzert and Nägele. "Component-Based Development and Verification of Safety Critical Software for a Brake-by-Wire System with Synchronous Software Components". In PDSE. IEEE, 1999

Agenda

I - Considering Synchronous Communications and LTL

Using a Linear-time Temporal Logic formula φ as specification

- Partial evaluation of LTL formula
- Assumes fixed communication delay
- **Decentralized progression** $\approx$ decentralized monitoring step:
  - for an *operator*, we use expansion laws and fix-point semantics:

$$P(\varphi_1 \vee \varphi_2, \sigma) = P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \qquad P(\mathbf{G}\,\varphi, \sigma) = P(\varphi, \sigma) \wedge \mathbf{G}(\varphi)$$
$$P(\varphi_1\,\mathbf{U}\,\varphi_2, \sigma) = P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1\,\mathbf{U}\,\varphi_2 \qquad \ldots$$

  - for *atomic propositions*, we evaluate or record a **past obligation**:

$$P(p, \sigma, APi) = \begin{cases} \top & \text{if } p \in \sigma \\ \bot & \text{if } p \notin \sigma \wedge p \in APi \\ \overline{\mathbf{X}}\,p & \text{otherwise} \end{cases}$$

  - for *past obligations*, we use the local memory:

$$P(\overline{\mathbf{X}}^m\,p, \sigma, APi) = \begin{cases} \top & \text{if } p \in APi \cap Mem_i(m) \\ \bot & \text{if } p \in APi \setminus Mem_i(m) \\ \overline{\mathbf{X}}^{m+1}\,p & \text{otherwise} \end{cases}$$

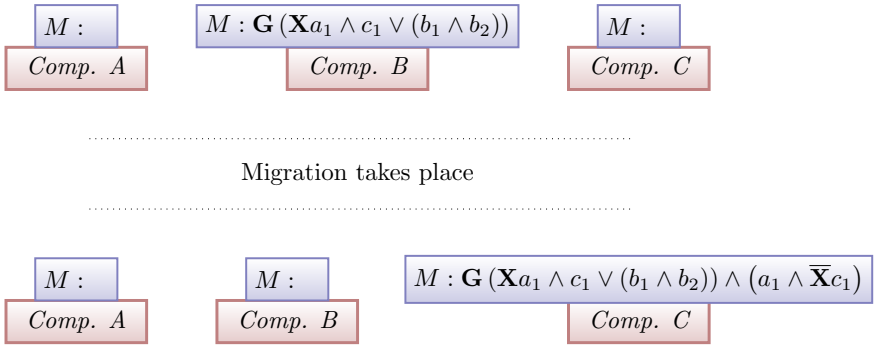
On some component C_i with atomic propositions APi.

Decentralised LTL monitoring. Bauer and Falcone. FM 2012: 85-100

I - Considering Synchronous Communications and LTL

Decentralized Monitoring Algorithm by **migration**

Each local monitor communicates with the monitor that can resolve the **most urgent** part of the formula



Properties

- **soundness**: any decentralized verdict is a centralized verdict;
- (eventual) **completeness**: decentralized verdict is eventually reached (provided no message loss)

- ⊕ simplicity
- ⊖ potential formula explosion problem
- ⊖ performance unpredictability
- ⊖ assumes fixed communication delay

Decentralised LTL monitoring. Bauer and Falcone. Formal Methods System Design. 48(1-2): 46-93 (2016)

II - Organizing Monitors

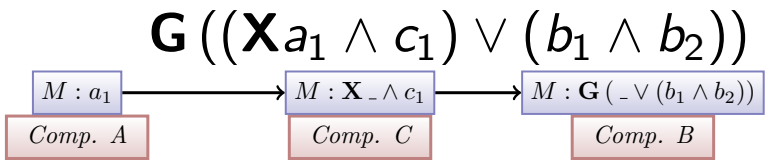
- Organizing monitors into a network + protocol for cooperation
- Using the syntactic structure of the formula
- Global formula → local formula

Arbitrary scoring function to place formula on monitors

Automated synthesis of network of monitors with references pointing to the verdict of other monitors

Decentralized Monitoring Algorithm by **choreography**

Each monitor reports to its parents and receives reports from its children



- ⊞ reduced message size
- ⊞ less computation
- ⊖ more messages
- ⊖ potential formula explosion problem

Organising LTL monitors over distributed systems with a global clock. Colombo and Falcone. Formal Methods Syst. Des. 49(1-2): 109-158 (2016)

Agenda

Execution History Encoding

Information as Atoms / Construction

- 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 ($\mathrm{enc} : AP \rightarrow Atoms$).
  - Ordering information (timestamp, round number, vector clock etc).

- Monitors store Atoms in their Memory

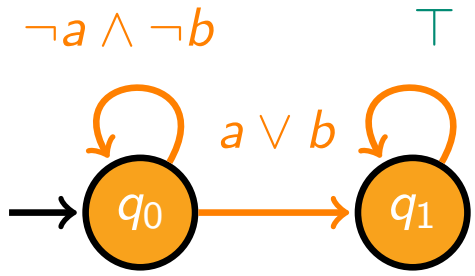
- EHE is constructed recursively & lazily (as needed and on-the-fly) using $\mathcal{A}$.

Monitoring decentralized specifications. El-Hokayem and Falcone: ISSTA 2017: 125-135

Execution History Encoding

Construction

Construction of the EHE, between $t = 0$ and $t = 2$, given that at $t = 0$, the automaton is in q_0



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 `EHE`s 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.

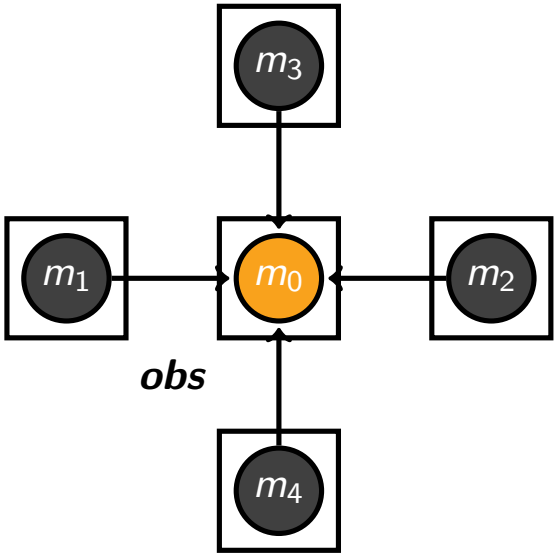
3. Predictable size[2]: $\mathcal{O}\left(\delta \times \mathit{Label} \times |Q|^{\delta+1}\right)$
   - The EHE encodes all potential and past states, as needed.
   - The more we keep track of potential states, the bigger the size.

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

[2] δ is the information delay, *Label* the max label size in the automaton, and Q the set of states.

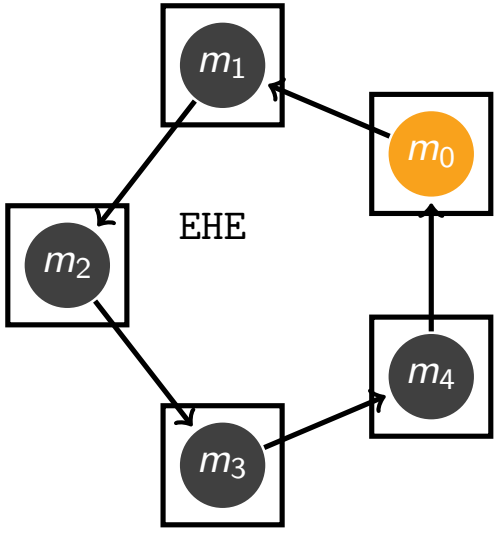
Studying Existing Algorithms

Principles of the algorithms



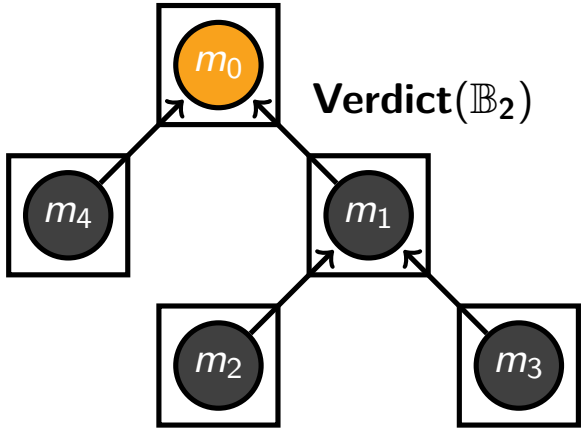
Orchestration

- 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

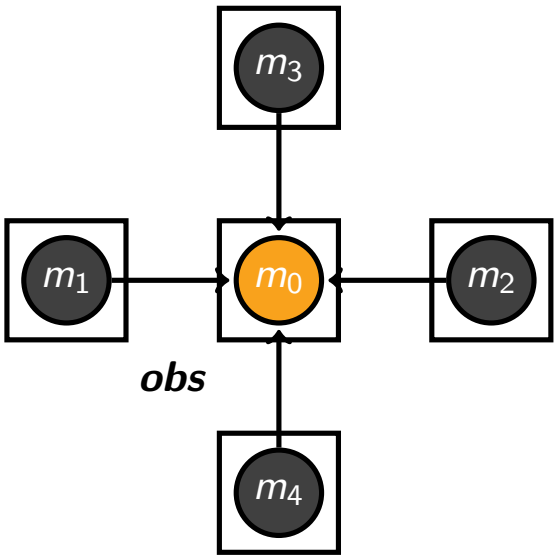


Choreography

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

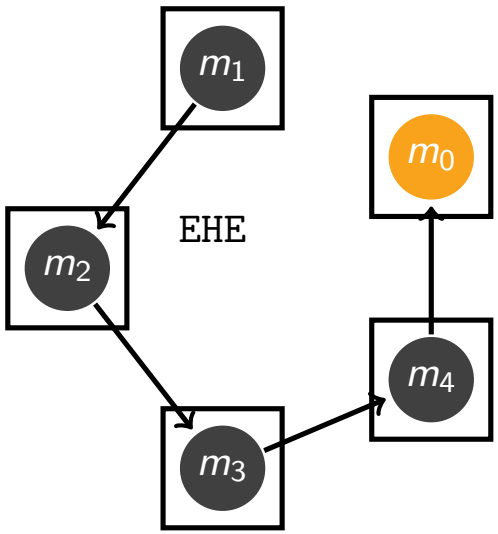
Studying Existing Algorithms

Expected Behavior



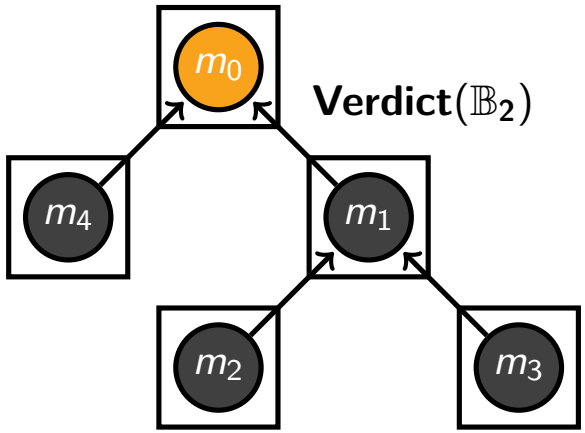
Orchestration

- 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

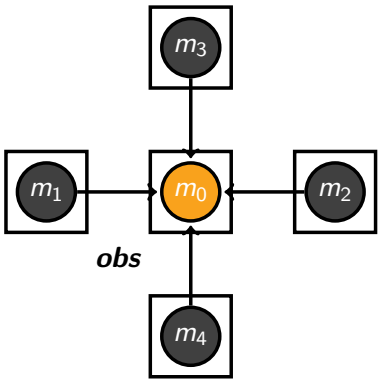


Choreography

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

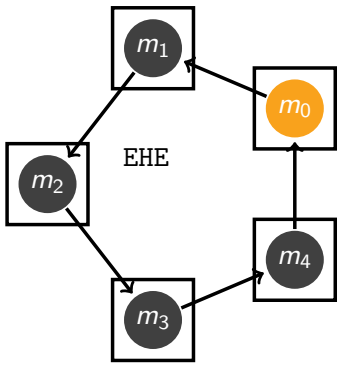
Studying Existing Algorithms

Orchestration



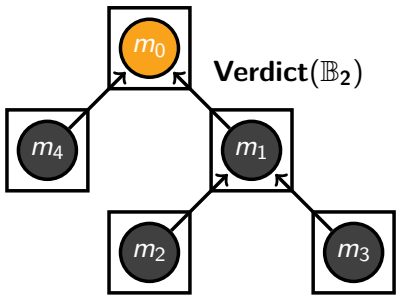
Central monitor + forwarding monitors

Migration



Specification hops from one component to another

Choreography



Monitors are organized in a tree

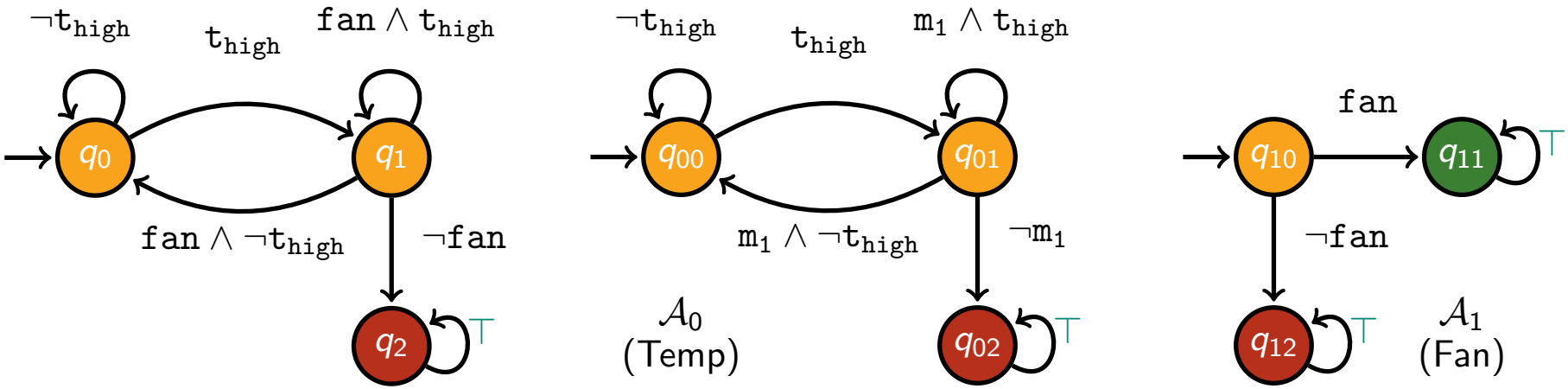
Expected behavior of algorithms

Algorithm	δ	**# Msg**	$\lvert\mathbf{Msg}\rvert$
Orchestration	$\Theta(1)$	$\Theta(\lvert\mathcal{C}\rvert)$	$\Theta(\lvert AP_c\rvert)$
Migration	$\mathcal{O}(\lvert\mathcal{C}\rvert)$	$\mathcal{O}(m)$	$\mathcal{O}(\lvert Q\rvert^{\lvert\mathcal{C}\rvert})$
Choreography	$\mathcal{O}(\mathrm{depth}(tree) + \lvert trace\rvert)$	$\Theta(\lvert Edges\rvert)$	$\Theta(1)$

$\mathcal{C}$: number of components, AP_c: global alphabet, m: number of active monitors, $|Q|$: number of states in the underlying automaton, *Edges*: number of edges in the network.

Decentralized Specifications

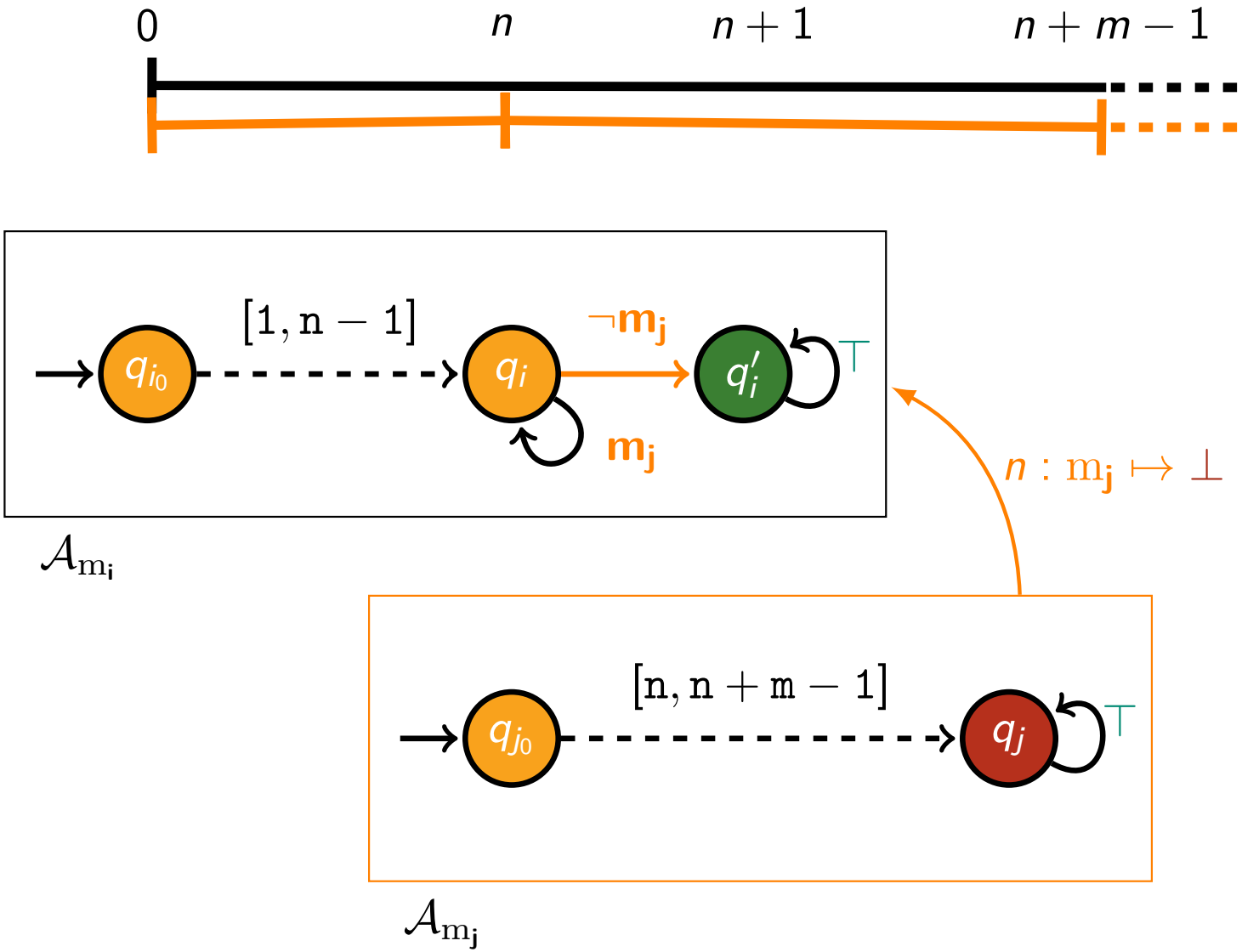
- A single automaton $\rightarrow$ Set of automata/monitors (Mons).
- Each monitor is associated with a component ($\mathcal{L} : \mathrm{Mons} \rightarrow \mathcal{C}$).
- Set of references to monitors (atomic propositions) (AP_{mons})
- The transition labels of an automaton $m \in \mathrm{Mons}$ are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



On the Monitoring of Decentralized Specifications: Semantics, Properties, Analysis, and Simulation.
El-Hokayem and Falcone ACM Trans. Softw. Eng. Methodol. 29(1): 1:1-1:57 (2020)

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.

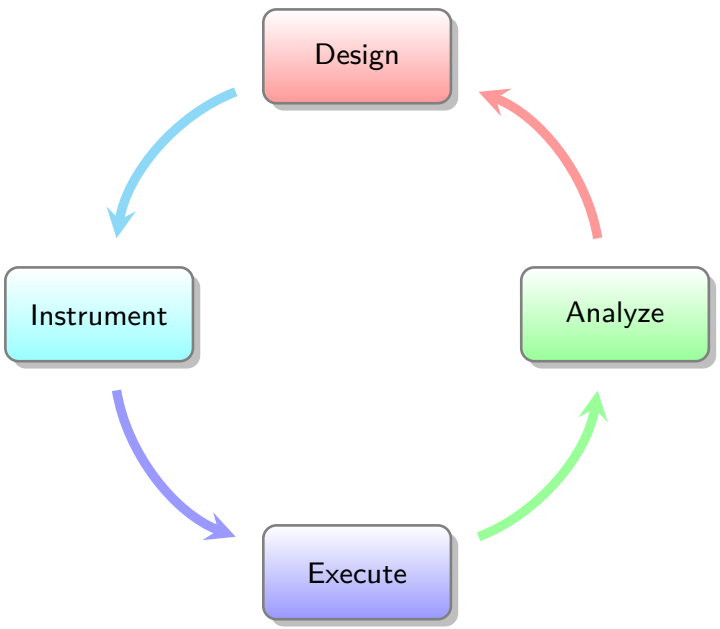
2. Monitoring
   - 2.1 Wait to receive observations from attached component.
   - 2.2 Receive messages (EHE or verdicts) from monitors.
   - 2.3 Process observations and messages (update the local EHE).
   - 2.4 Communicate with other monitors.

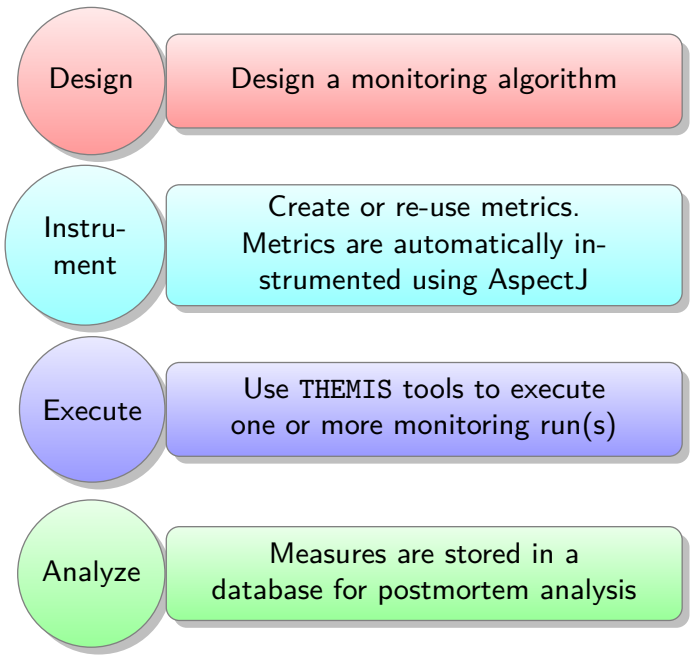
Agenda

The THEMIS tool

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).





THEMIS: a tool for decentralized monitoring algorithms. El-Hokayem and Falcone: ISSTA 2017: 372-375

Demo, source code, and tutorial: `https://gitlab.inria.fr/monitoring/themis-demo`

The Migration Algorithm in THEMIS

Figure: 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 }
```

Figure: 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

Verifying Behavior

Simulate the behavior of orchestration, migration, and choreography.

Confirm the trends predicted by the analysis.

Experiment Setup ($5,868,800$ runs): [3]

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

[3] More experiments and results in paper:

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

Existing Algorithms

GraphStream

HEMIS
Monitoring Framework

LIG cnrs Inria Université Grenoble Alpes

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

Agenda

Monitored Environment

Amiqual4Home[4]

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[5]

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

[4] `amiqual4home.inria.fr`

[5] **Bringing Runtime Verification Home**. Antoine El-Hokayem, Yliès Falcone. RV 2018 and extended version in Software Tool for Technology Transfer 2021.

Property Groups

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

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

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

- 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.
  - $\texttt{firehazard} \stackrel{\text{def}}{=} \mathbf{G}(\texttt{napping} \implies \neg\texttt{cooking})$

Properties

Example Properties

ADL	m_toilet	toilet_water
	sink_usage	$\mathbf{G}_3$(m_bathroom_sink_water)
	bathroom_sink	bathroom_sink_cold $\vee$ bathroom_sink_hot
	shower_usage	$\mathbf{G}_2$(m_bathroom_shower_water)
	napping	$\mathbf{G}_{25}$(m_bedroom_bed_pressure)
	dressing	$\mathbf{F}_4$(m_bedroom_closet_door $\vee$ m_bedroom_drawers))
	reading	m_bedroom_light $\wedge$ $\mathbf{F}_4$($\neg$dressing $\wedge$ $\neg$napping)
	office_tv	$\mathbf{F}_3$(m_office_tv)
	computing	$\mathbf{F}_3$(m_office_deskplug)
	livingroom_tv	$\mathbf{F}_3$(m_livingroom_tv $\wedge$ m_livingroom_couch)
	eating	$\neg$m_kitchen_presence $\wedge$ $\mathbf{G}_6$(m_livingroom_table)
Meta	actfloor(0)	cooking $\vee$ preparing $\vee$ eating $\vee$ washing_dishes $\vee$ livingroom_tv $\vee$ m_toilet
	acthouse	actfloor(0) $\vee$ actfloor(1)
	notwopeople	$\neg$(actfloor(0) $\wedge$ actfloor(1))
	firehazard	napping $\implies$ $\neg$cooking

Decentralization

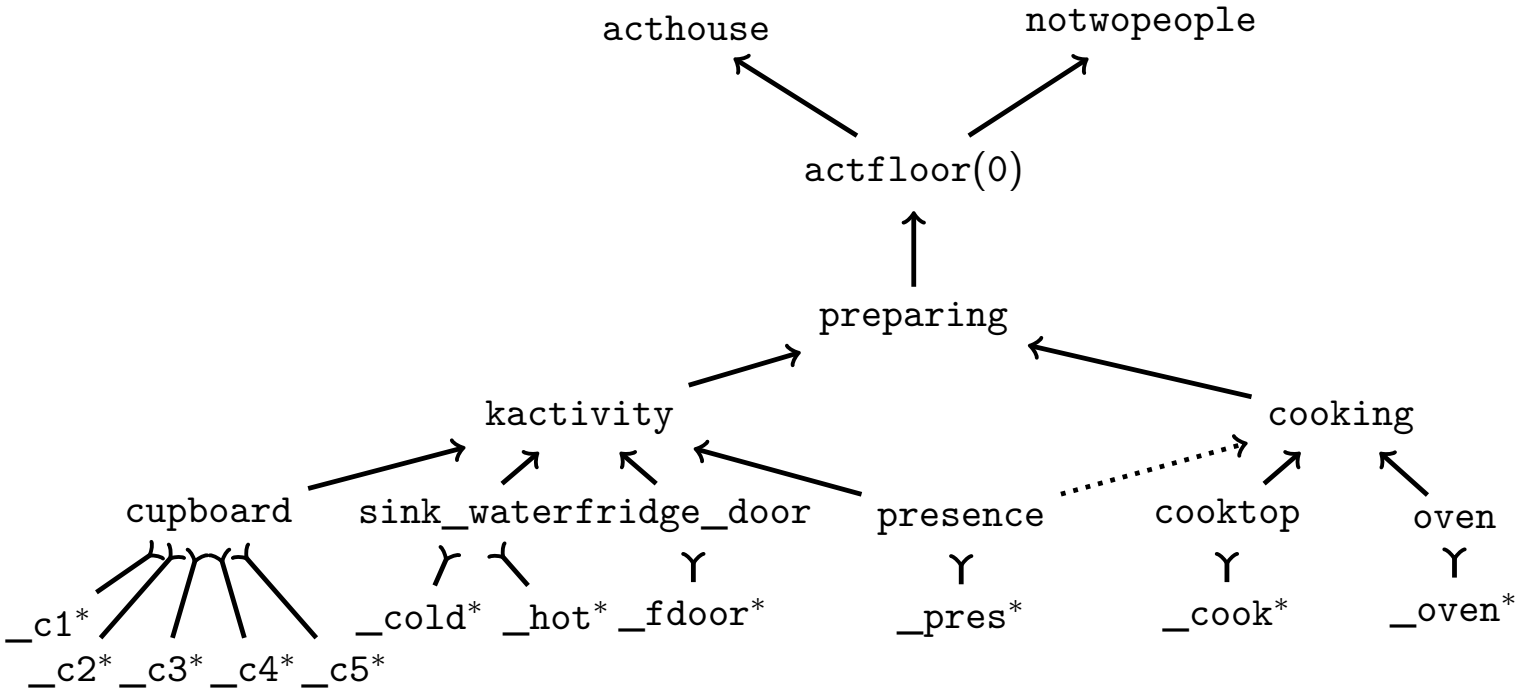
Taking Advantage of Hierarchies and Decentralized Specifications

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 Avoid duplication of 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

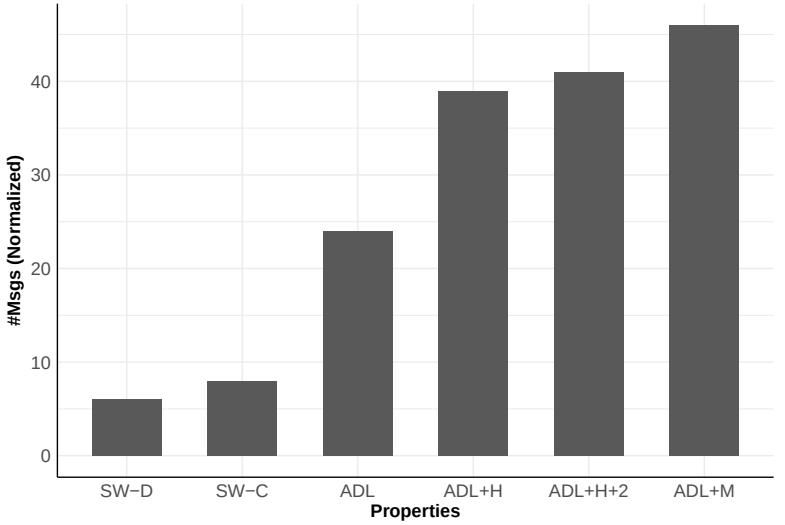
Re-using computation and communication

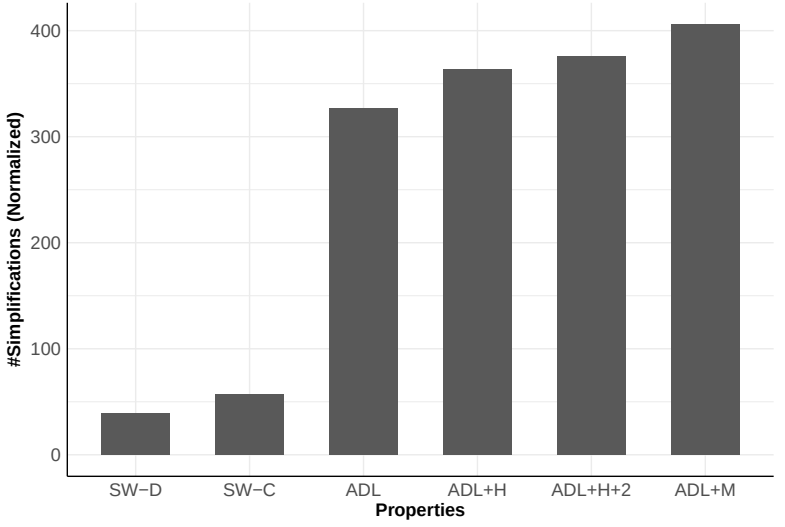
- A shared sub-specification is monitored once.
- Higher-up specifications do not need the sub-specification sensors.
- Adding meta-properties incurs less overhead due to re-use.

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

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

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

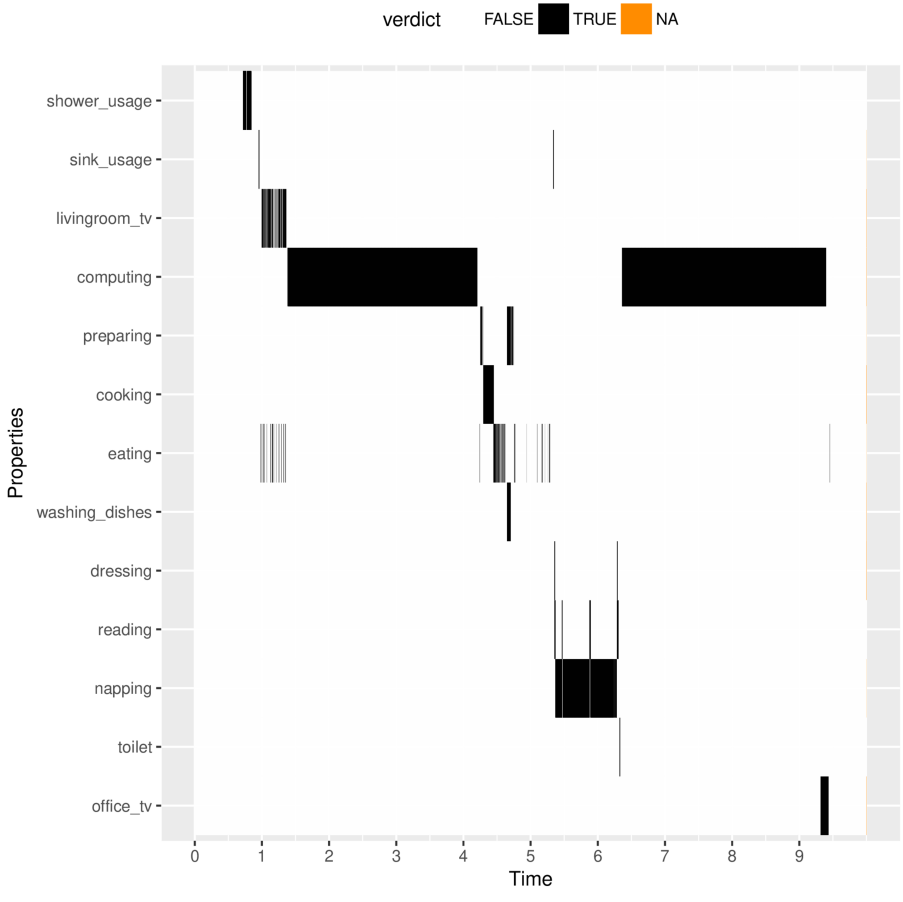




Schedule

Suggested (left) vs Reconstructed (right)

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



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
$\mathbf{G}_{25}$(`weight`)	0.43	0.95	0.60
$\mathbf{G}_{3}$(`weight`)	0.43	0.99	0.60
$\mathbf{F}_{3}$(`weight`)	0.43	1.0	0.60
$\mathbf{G}_{3}$(`pres` $\wedge$ `weight`)	0.34	0.14	0.20
$\mathbf{G}_{3}$($\neg\ell \wedge$ `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

Agenda

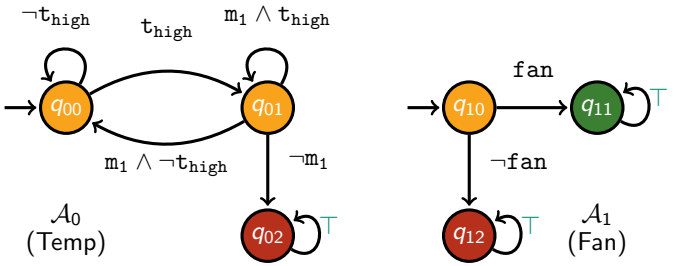
Summary

Decentralized Monitoring of (De)Centralized Specifications

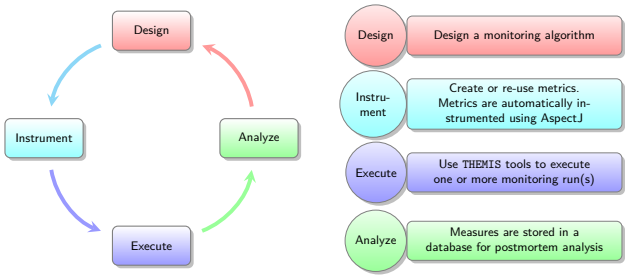
Aim for predictable behavior → Automata + EHE data structure.

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

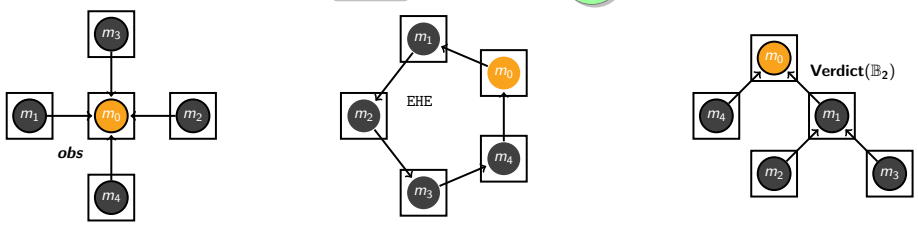
Separate synthesis from monitoring: decentralized specifications.



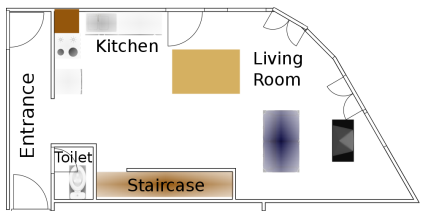
Methodology + tool support for designing, measuring, comparing and extending decentralized RV algorithms.



Adapted and compared existing algorithms



Application to smart homes.



Research Perspectives

- Centralized specification $\rightarrow$ equivalent decentralized specifications.
  - Optimize existing methods.
  - Take into account the topology of the monitored system.
- More expressive specifications (time, data).
- Going fully distributed: using vector clocks in atoms.
- Extend `THEMIS`
  - metrics,
  - monitor deployment / component instrumentation,
  - better visualization of algorithm behavior.
- Runtime enforcement of centralized and decentralized specifications.
- Case studies for modern decentralized systems (e.g., UAVs, smart cities, ...)

Agenda

Related Work and Goals

More Details

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

Agenda

Related Work and Goals

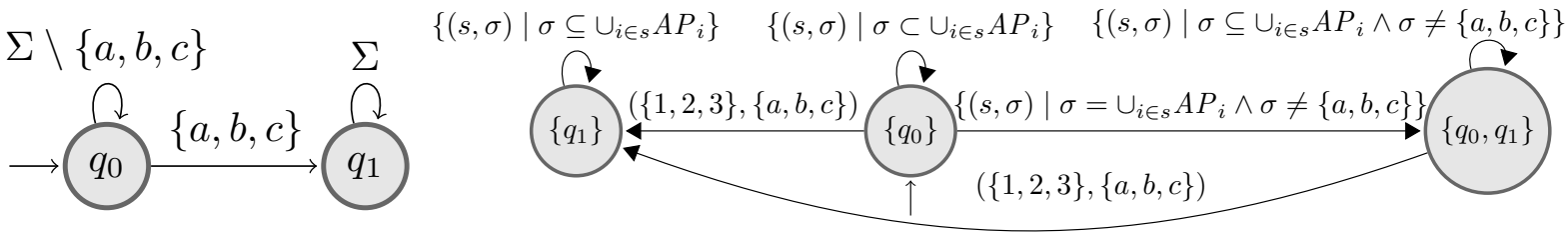
More Details

Using finite-state automata

- Using finite-state automata as specification formalism
- All regular properties and no monitorability issue (finite-word semantics)

Decentralization of a monitor and monitoring algorithm

- **State estimator** synthesized by "inverse determinisation" wrt partial information – (s, σ): σ occurred on components in s



- Monitors exchange information of the form (t, s, σ) about the occurrence of atomic propositions in σ on components in s at time t

⊕ monitor size determined statically

⊕ no formula explosion

⊖ redundancy of monitor computation/specification

Efficient and Generalized Decentralized Monitoring of Regular Languages. Falcone, Cornebize and Fernandez. FORTE 2014: 66-83

Execution History Encoding

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 $S(t')$, $t' > t$:

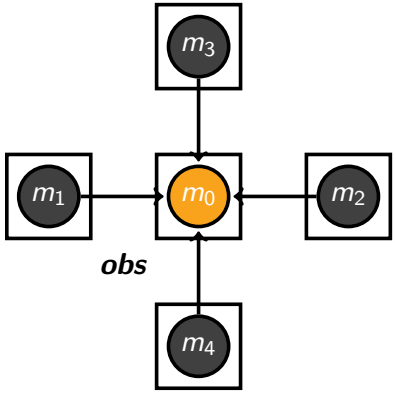
$$S(t') = |Q| \times (S(t'-1) + L)$$
$$= \mathcal{O}(|Q|^{t'}).$$

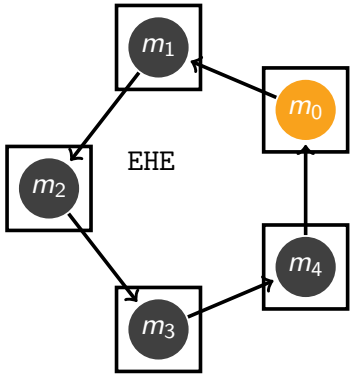
- Size of EHE:
$|\mathcal{I}^{\delta}| = \mathcal{O}\left(\delta \times |Q| \times L \times |Q|^{\delta}\right).$

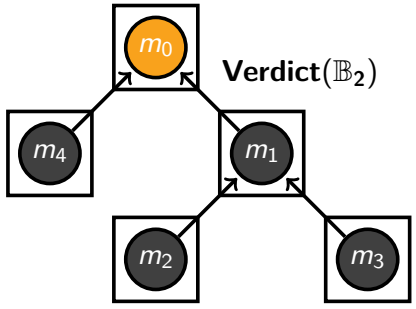
$$t \mapsto q \mapsto \top$$

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

Studying Existing Algorithms





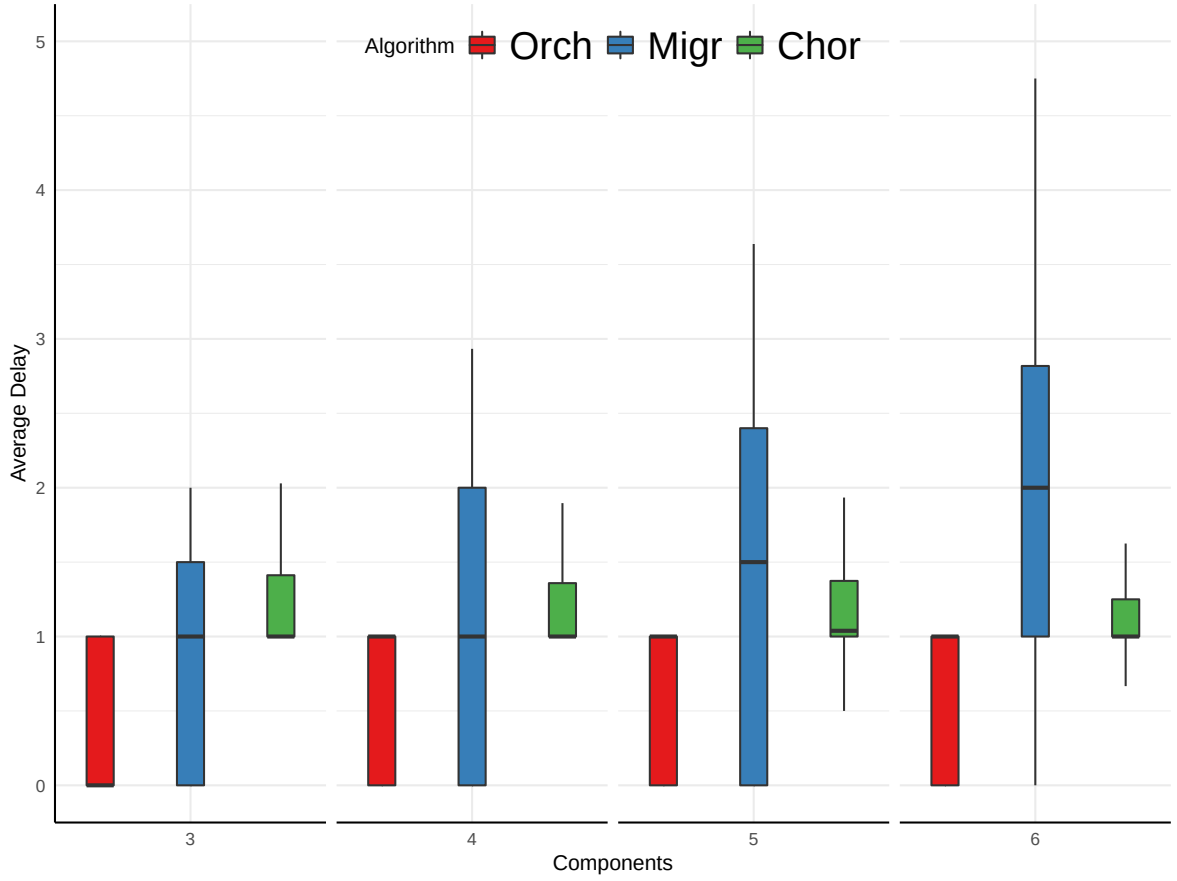


- Example algorithms
  - Orchestration: Central monitor + forwarding monitors.
  - Migration: Specification hops from one component to another.
  - Choreography: Monitors are organized in a tree.
- Expected behavior of algorithms:

Algorithm	δ	**# Msg**	$\lvert\mathbf{Msg}\rvert$
Orchestration	$\Theta(1)$	$\Theta(\lvert\mathcal{C}\rvert)$	$\Theta(\lvert AP_c\rvert)$
Migration	$\mathcal{O}(\lvert\mathcal{C}\rvert)$	$\mathcal{O}(m)$	$\mathcal{O}(\lvert Q\rvert^{\lvert\mathcal{C}\rvert})$
Choreography	$\mathcal{O}(\mathrm{depth}(\mathrm{rt}) + \lvert\mathrm{tr}\rvert)$	$\Theta(\lvert E\rvert)$	$\Theta(1)$

Results

Delay

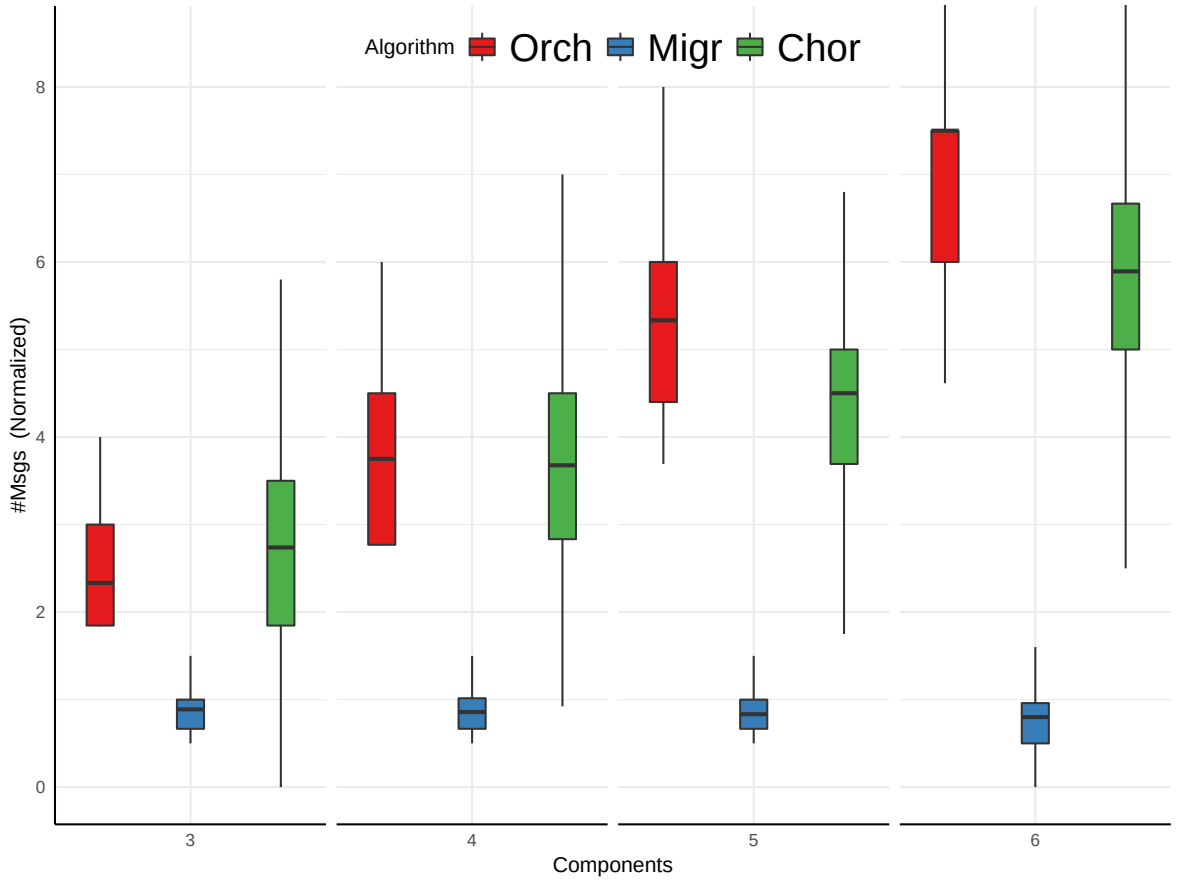


Recall from the analysis:

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

Results

Number of Messages

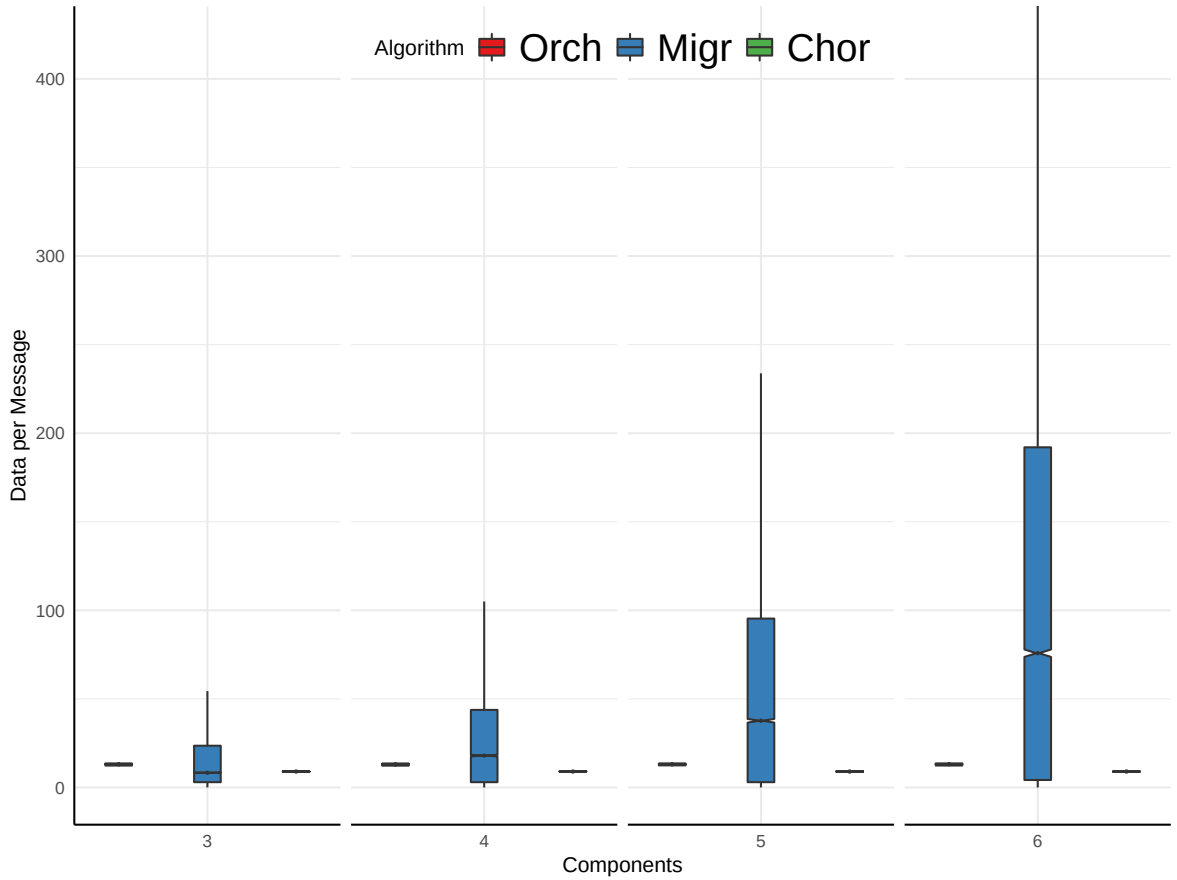


Recall from the analysis:

- 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.