Organising LTL Monitors over Systems with a Global Clock

Yliès Falcone

joint work with Andreas Bauer (NICTA Canberra, Australia)

and Christian Colombo (U of Malta, Malta)

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

DRV Workshop, Bertinoro, Italy

Outline

Outline – Background

Outline – Background

"Classical" runtime validation method: monitoring

Runtime Verification [Klaus Havelund, Grigore Rosu]

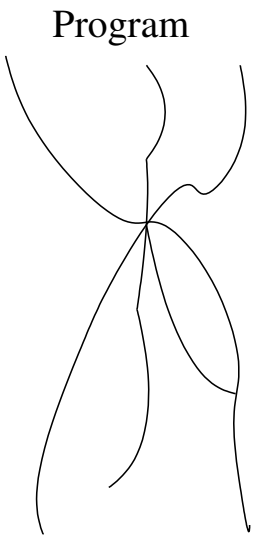
- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness specification

"Classical" runtime validation method: monitoring

Runtime Verification [Klaus Havelund, Grigore Rosu]

- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness specification

Get a **program/system**

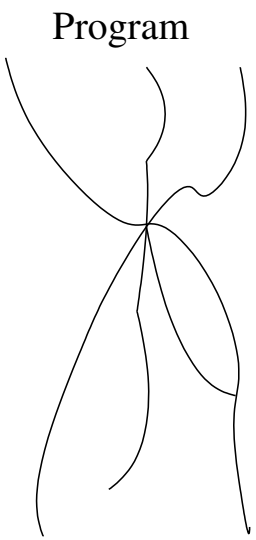


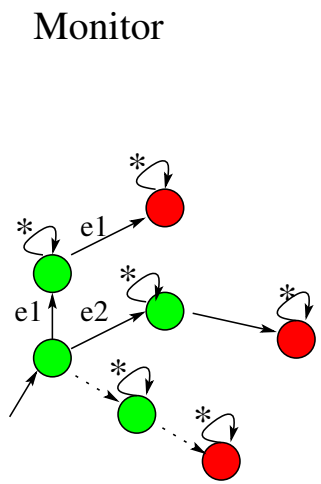
"Classical" runtime validation method: monitoring

Runtime Verification [Klaus Havelund, Grigore Rosu]

- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness specification

Get a **program/system**
Synthesize a monitor: a decision procedure for the specification



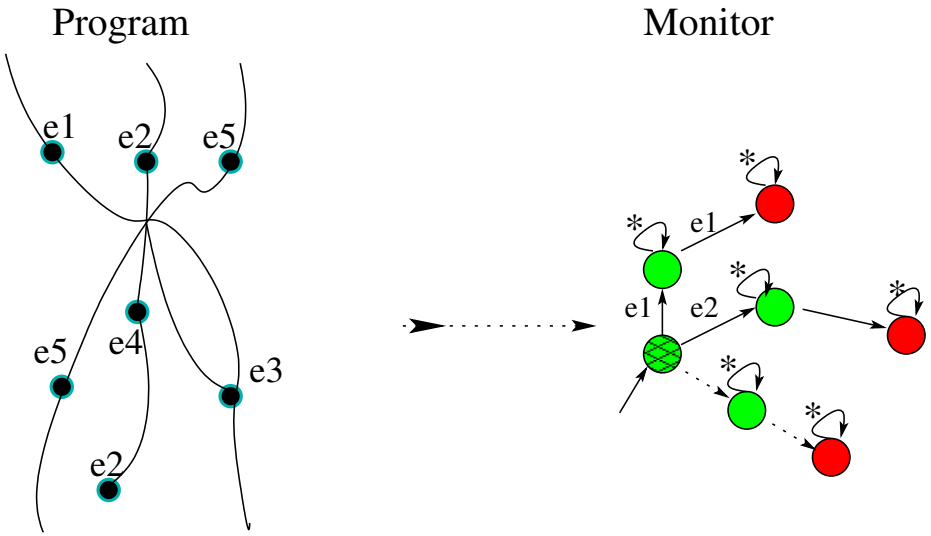


"Classical" runtime validation method: monitoring

Runtime Verification [Klaus Havelund, Grigore Rosu]

- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness specification

Get a **program/system**
Synthesize a monitor: a decision procedure for the specification
Instrument the underlying program to observe relevant events: $e_i \in \Sigma$

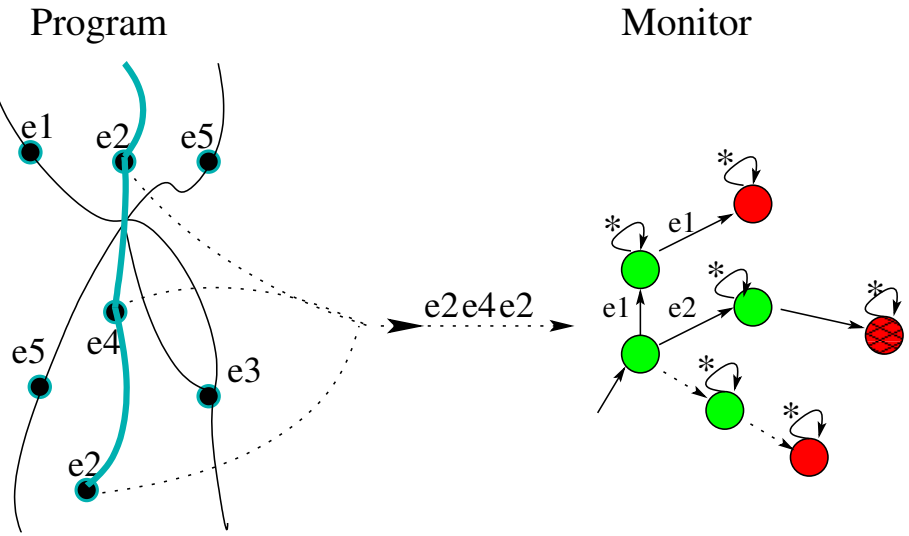


"Classical" runtime validation method: monitoring

Runtime Verification [Klaus Havelund, Grigore Rosu]

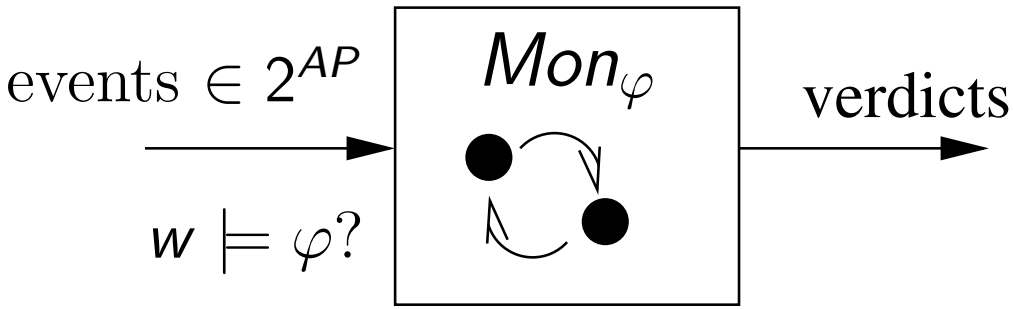
- A lightweight verification technique "bridging the gap" between **testing** and **verification**
- **Checking** whether a run of the system under scrutiny satisfies a given correctness specification

Get a **program/system**
Synthesize a monitor: a decision procedure for the specification
Instrument the underlying program to observe relevant events: $e_i \in \Sigma$
A monitor acts at runtime as an **oracle** for the specification (validation/violation)



"Classical" runtime validation method: monitoring

Determine a set of atomic propositions AP of the system
e.g., for a car $AP = \{speed\_low, seat\_belt\_1\_on, \ldots\}$



Several existing tools (e.g., Java-MOP [Rosu et al.], RuleR [Barringer et al.],. . .)

Applied to several domains: Java/C programs, Web services, Space flight software, system biology. . .

Outline – Background

Linear-time Temporal Logic

Pnueli 77

One of the most widely used specification formalism

Consider a set of atomic propositions AP

Syntax:

$$\varphi ::= p \in \mathrm{AP} \mid (\varphi) \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi$$

where:

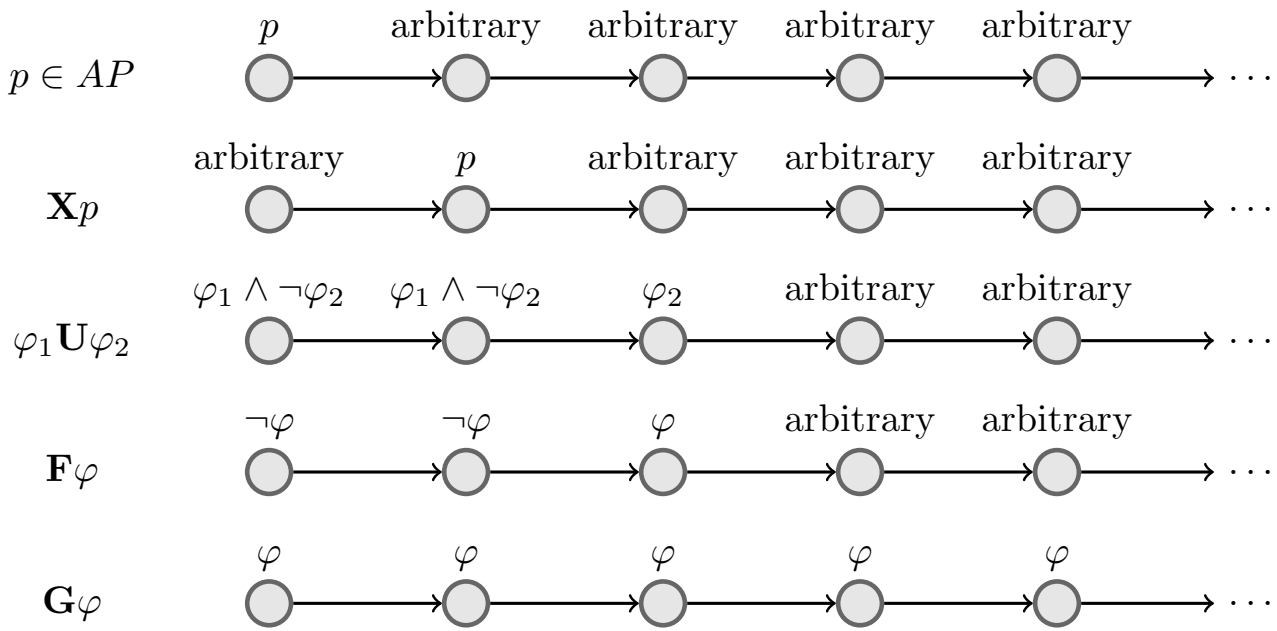
- $\neg, \vee$ are operators from propositional logic
- **X** is the "next" operator
- **U** is the until operator

Additional operators:

- **F** is the "eventually" operator: $\mathbf{F}\varphi = \texttt{true}\ \mathbf{U}\ \varphi$
- **G** is the "globally" operator: $\mathbf{G}\varphi = \neg(\mathbf{F}(\neg\varphi))$

Linear-time Temporal Logic

Semantics



Given $w \in \Sigma^\infty$ and $i \geq 0$ the (inductive) semantics is:

$$
\begin{array}{lcl}
w^i \models p & \Leftrightarrow & p \in w(i), \text{ for any } p \in AP \\
w^i \models \neg\varphi & \Leftrightarrow & w^i \not\models \varphi \\
w^i \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & w^i \models \varphi_1 \vee w^i \models \varphi_2 \\
w^i \models \mathbf{X}\varphi & \Leftrightarrow & w^{i+1} \models \varphi \\
w^i \models \varphi_1 \mathbf{U} \varphi_2 & \Leftrightarrow & \exists k \in [i, \infty[.\ w^k \models \varphi_2 \wedge \forall l \in [i, k[.\ w^l \models \varphi_1
\end{array}
$$

LTL for monitoring: LTL_3 - Bauer et al.

LTL has mostly been used in validation techniques such as model-checking

LTL for monitoring: LTL_3 - Bauer et al.

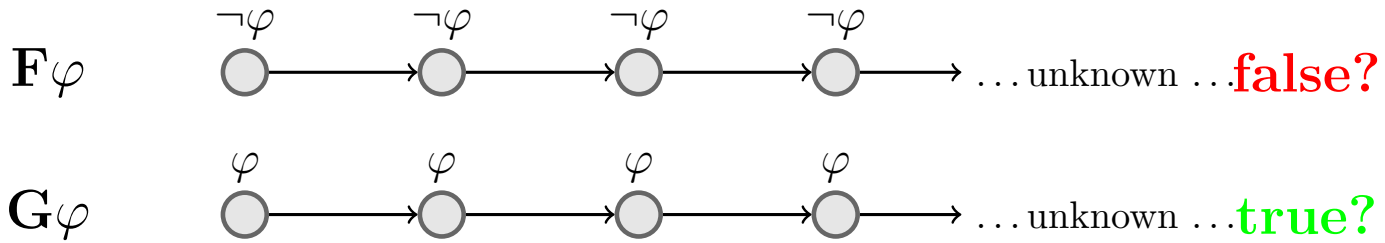
LTL has mostly been used in validation techniques such as model-checking
The semantics needs to be adapted for monitoring
2 issues with a semantics over infinite sequences:

- liveness properties
- we do not "know" the future

LTL for monitoring: LTL_3 - Bauer et al.

LTL has mostly been used in validation techniques such as model-checking
The semantics needs to be adapted for monitoring
2 issues with a semantics over infinite sequences:

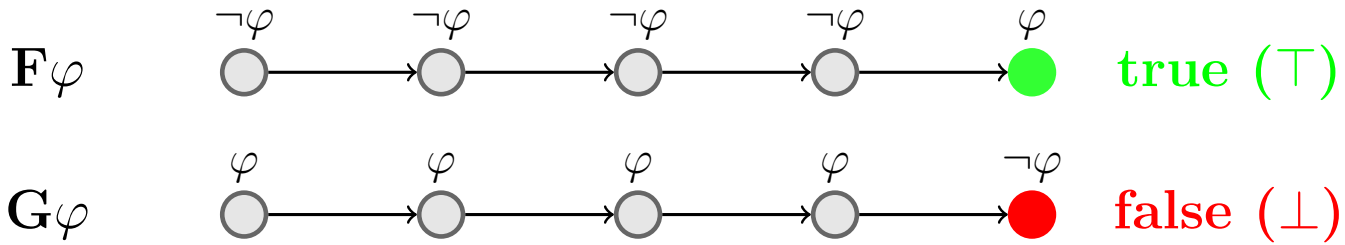
- liveness properties
- we do not "know" the future



LTL for monitoring: LTL_3 - Bauer et al.

LTL has mostly been used in validation techniques such as model-checking
The semantics needs to be adapted for monitoring
2 issues with a semantics over infinite sequences:

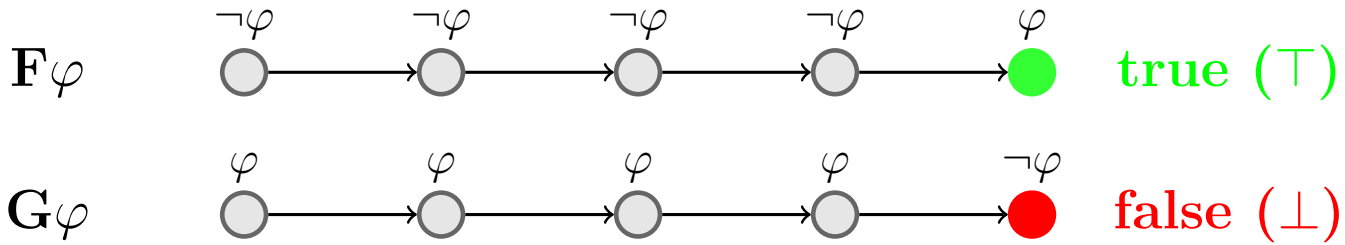
- liveness properties
- we do not "know" the future



LTL for monitoring: LTL_3 - Bauer et al.

LTL has mostly been used in validation techniques such as model-checking
The semantics needs to be adapted for monitoring
2 issues with a semantics over infinite sequences:

- liveness properties
- we do not "know" the future



Definition (LTL_3 semantics for a formula φ)

- $\text{good}(\varphi) = \{u \in \Sigma^* \mid u \cdot \Sigma^\omega \subseteq \mathcal{L}(\varphi)\}$
- $\text{bad}(\varphi) = \{u \in \Sigma^* \mid u \cdot \Sigma^\omega \subseteq \Sigma^\omega \setminus \mathcal{L}(\varphi)\}$
- Given $u \in \Sigma^*$:

$$u \models_3 \varphi \begin{cases} \top & \text{if } u \in \text{good}(\varphi) \\ \bot & \text{if } u \in \text{bad}(\varphi) \\ ? & \text{otherwise} \end{cases}$$

Outline – Motivations

An introductory example

Most modern cars realise the following abstract requirement:

> "Issue warning if one of the passengers is not wearing a seat belt (when the car has reached a certain speed)."

An introductory example

Most modern cars realise the following abstract requirement:

"Issue warning if one of the passengers is not wearing a seat belt (when the car has reached a certain speed)."

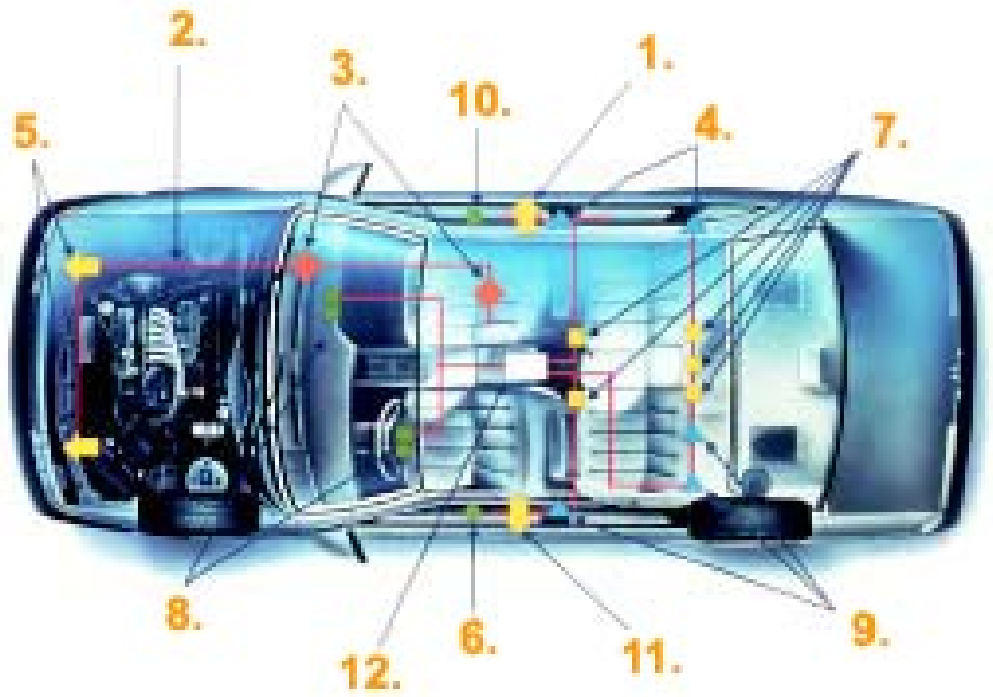
Could be formalised using LTL:

$$\varphi := \mathbf{G}\big(speed\_low \quad \vee \quad ((pressure\_sensor\_1\_high \Rightarrow seat\_belt\_1\_on) \wedge \ldots \wedge (pressure\_sensor\_n\_high \Rightarrow seat\_belt\_n\_on))\big)$$

and then monitored as usual...

An introductory example

However, cars are nowadays highly distributed systems ($\geq$ 130 CPUs):

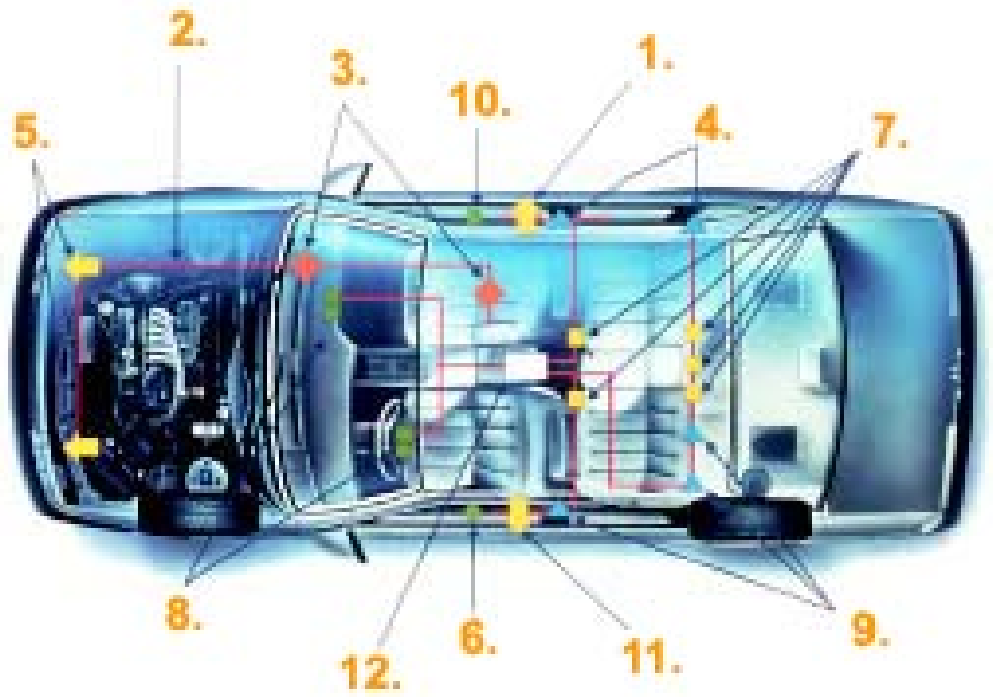


Legend:

3. Occupant sensing system (only one shown)
7. Seat-belt buckle sensors

An introductory example

However, cars are nowadays highly distributed systems ($\geq$ 130 CPUs):



Legend:

3. Occupant sensing system (only one shown)
7. Seat-belt buckle sensors

You can't easily monitor φ without central observation point!

Outline – Decentralised Monitoring of LTL formulae

Outline – Decentralised Monitoring of LTL formulae

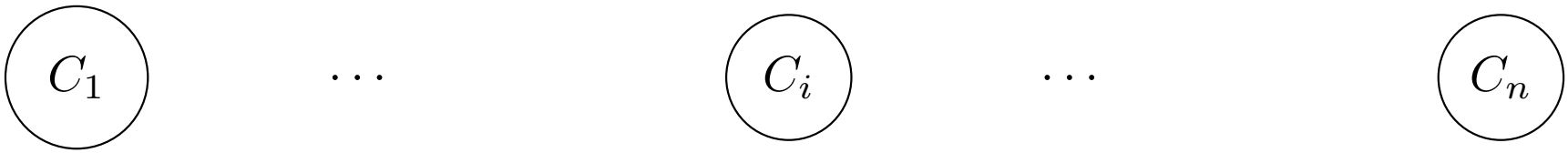
Decentralised monitoring – Our setting

Distributed system operating under a global clock:

Decentralised monitoring – Our setting

Distributed system operating under a global clock:

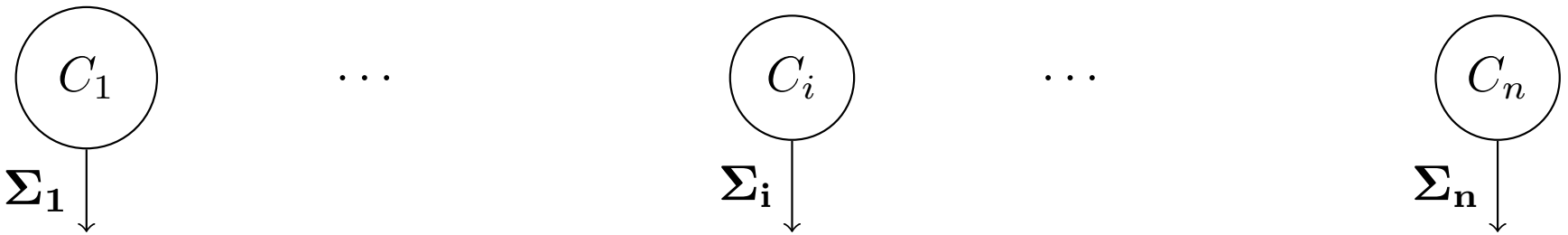
- A set of "components" $C_1, \ldots, C_n$



Decentralised monitoring – Our setting

Distributed system operating under a global clock:

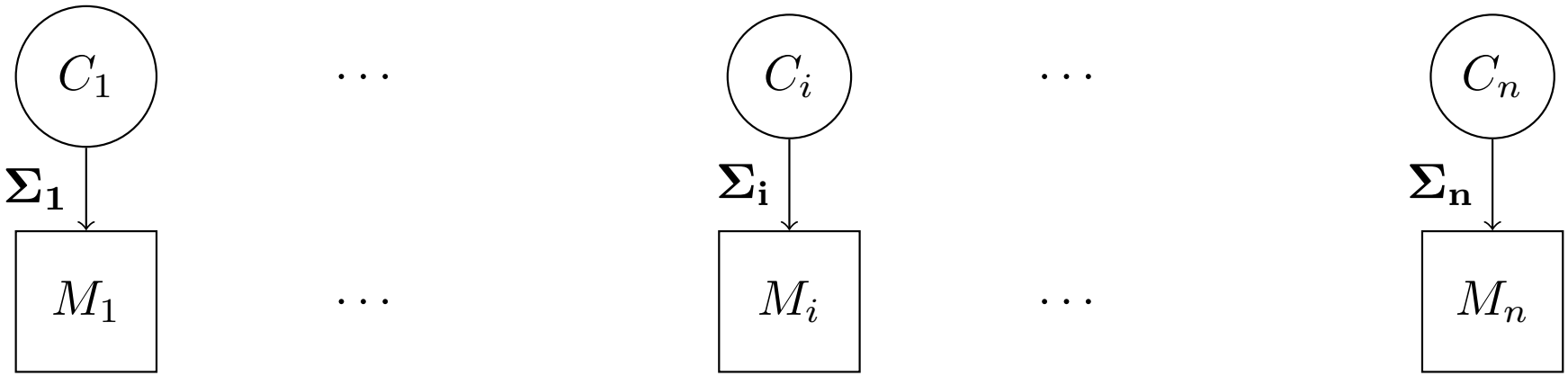
- A set of "components" $C_1, \ldots, C_n$
- $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$: all system events (where $\forall i, j : i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \emptyset$)



Decentralised monitoring – Our setting

Distributed system operating under a global clock:

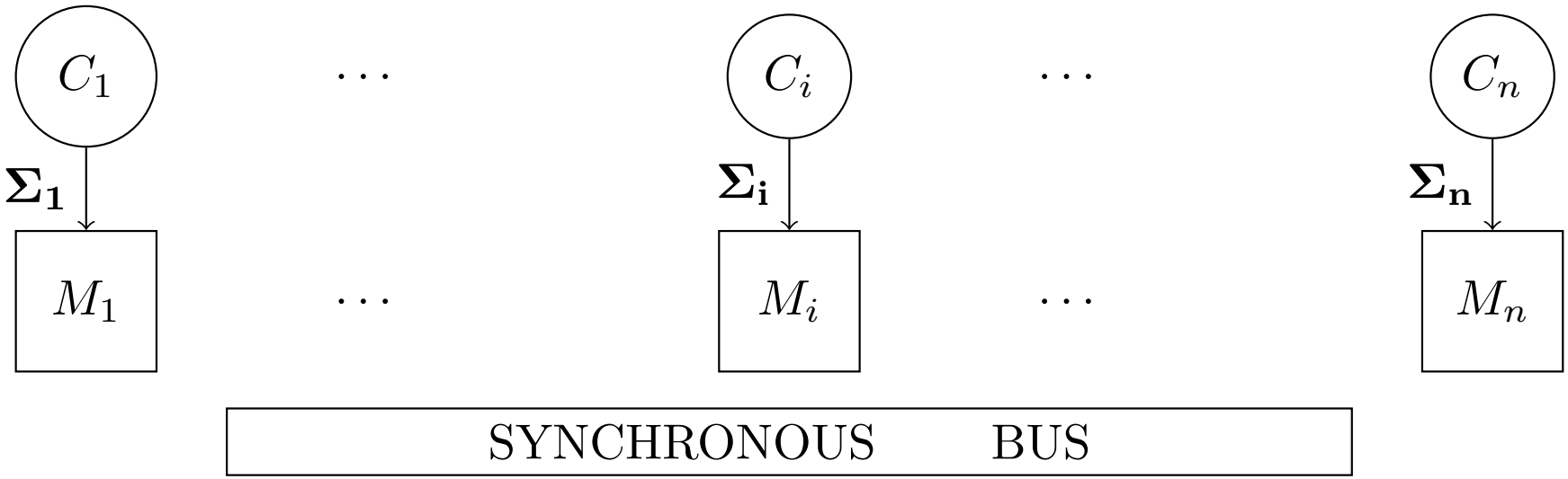
- A set of "components" $C_1, \ldots, C_n$
- $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$: all system events (where $\forall i, j : i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \emptyset$)
- No central observation point
- but monitors $M_1, \ldots, M_n$ are attached to components



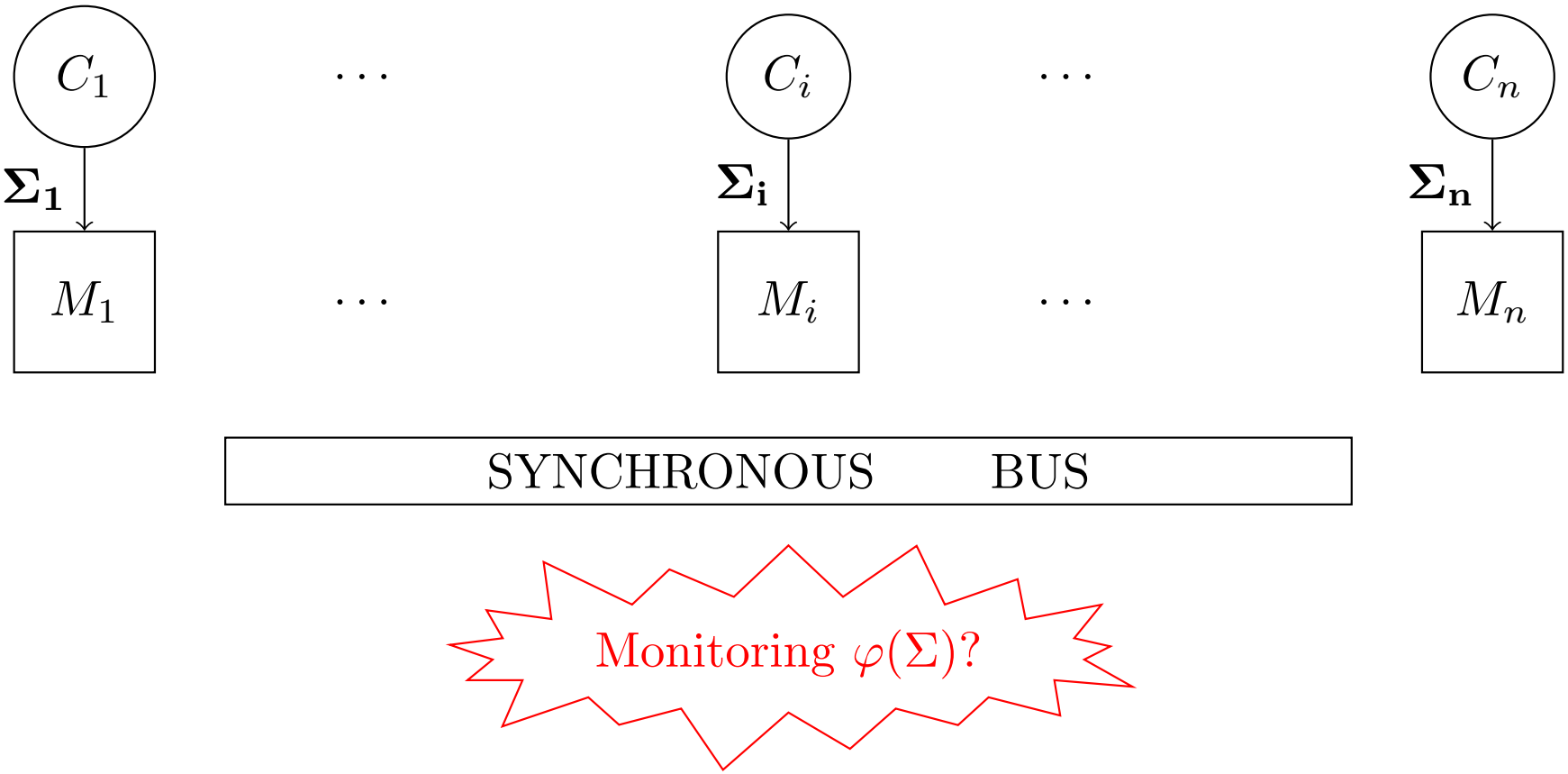
Decentralised monitoring – Our setting

Distributed system operating under a global clock:

- A set of "components" $C_1, \ldots, C_n$
- $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$: all system events (where $\forall i, j : i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \emptyset$)
- No central observation point
- but monitors $M_1, \ldots, M_n$ are attached to components
- Synchronous bus: at time t a monitor may send/receive a message:
  - At $t + 1$ this message is received by the recipient.
  - That is, computation takes no time.



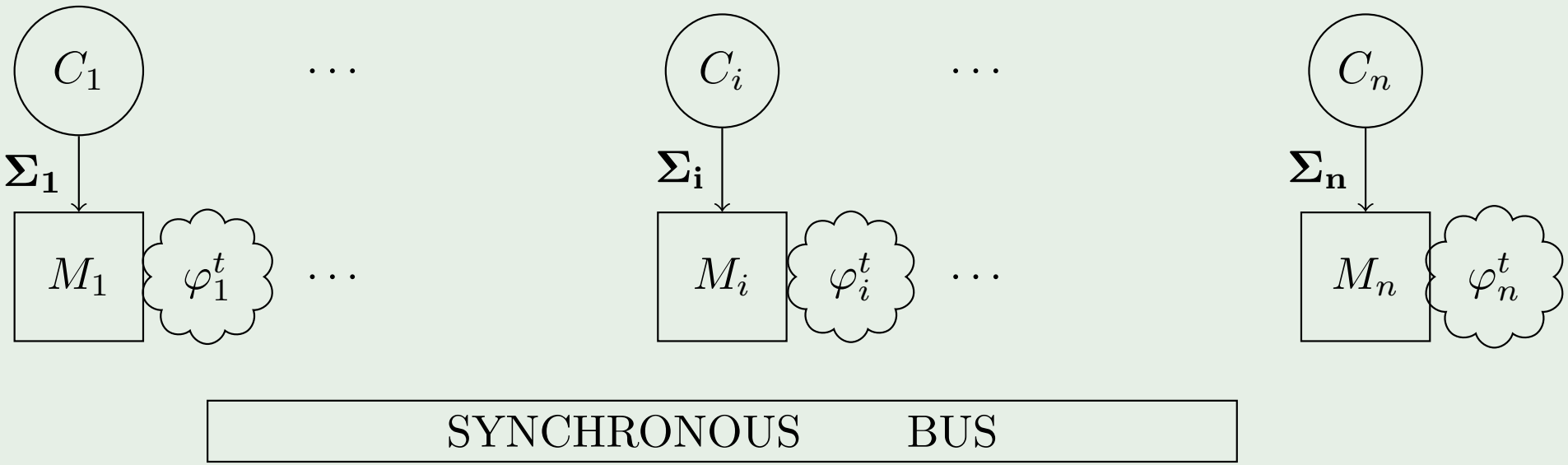
Decentralised monitoring – the idea



Decentralised monitoring – the idea

Distribute φ's evaluation & exchange obligations

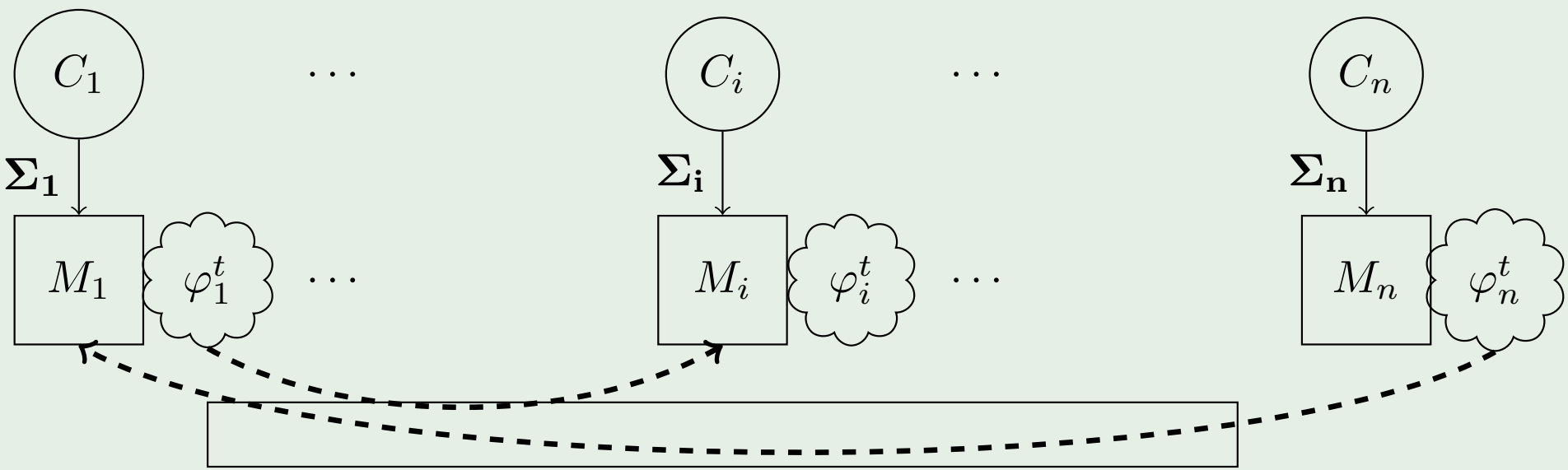
Proposed Solution:



Decentralised monitoring – the idea

Distribute φ's evaluation & exchange obligations

Proposed Solution:



Three organizations of monitors: orchestration, migration, and choreography (borrowing terminology from Francalanza et al.)

A note on the global clock and synchrony

– *"Is a global clock realistic?"*

A note on the global clock and synchrony

– *"Is a global clock realistic?"*
– *"Not always, but many safety critical systems use it."*

A note on the global clock and synchrony

– *"Is a global clock realistic?"*
– *"Not always, but many safety critical systems use it."*

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

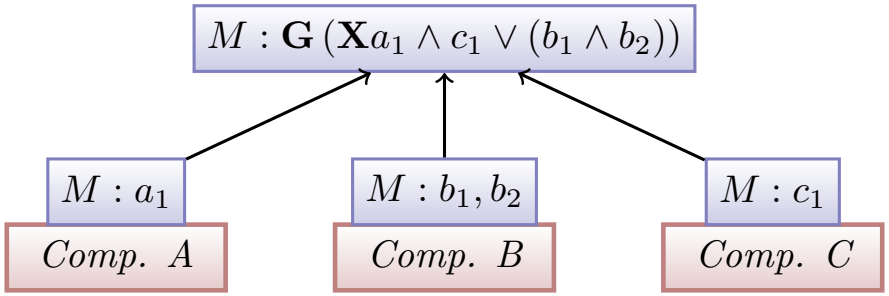
Examples: Steer-by-wire, brake-by-wire, engine management, etc.

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

Examples for implementation/verification systems used in this domain: SIGNAL, Lustre, Astrée verifier, etc.

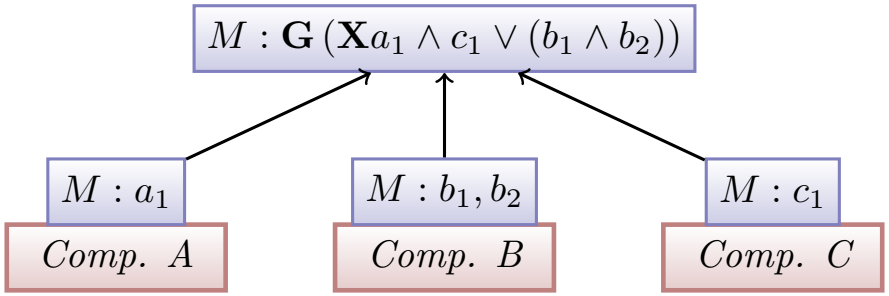
Outline – Decentralised Monitoring of LTL formulae

Orchestration (simplified)



- Central point monitoring the global formula.
- Several communication "protocols" can be used to forward local observations.

Orchestration (simplified)

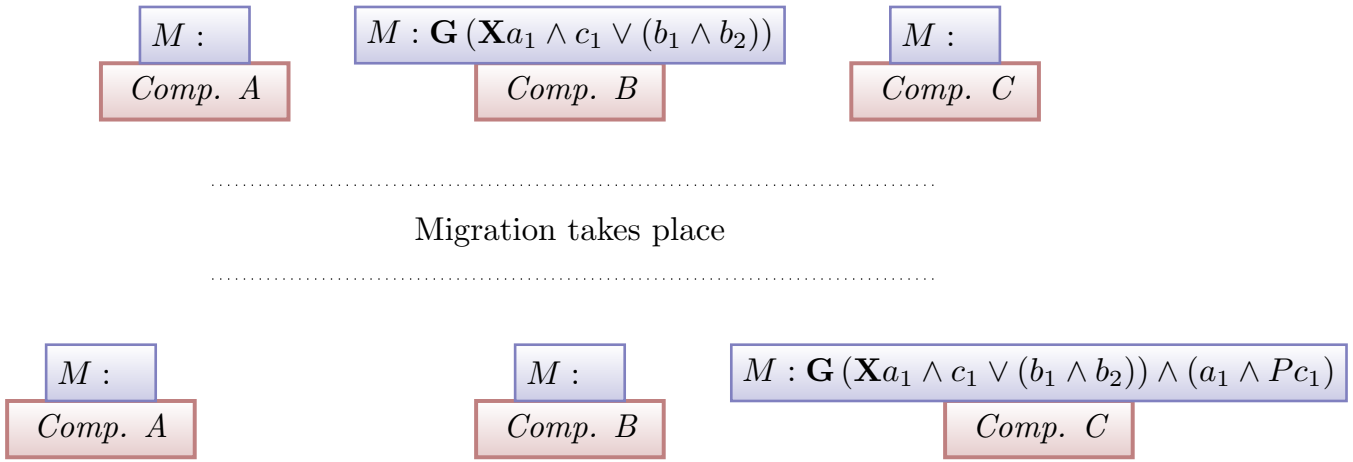


- Central point monitoring the global formula.
- Several communication "protocols" can be used to forward local observations.

At the central site, at each time step, when globally monitoring φ:

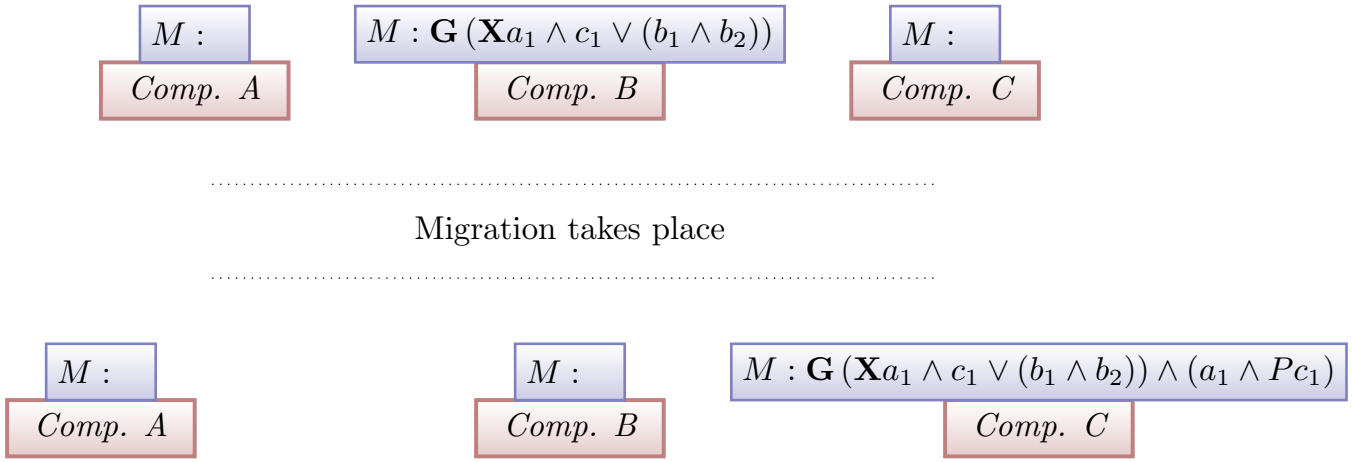
1. Wait for all observations to arrive from the remote components.
2. Merge all observations to form an event.
3. Progress φ with the event and simplify the progressed formula.
4. If a verdict is reached, stop monitoring and report result.

Migration (simplified)



- Monitor state encoded by a formula traversing the network.
- Formula to be satisfied given the local observations of traversed components.
- Formula may contain references to past time instants.

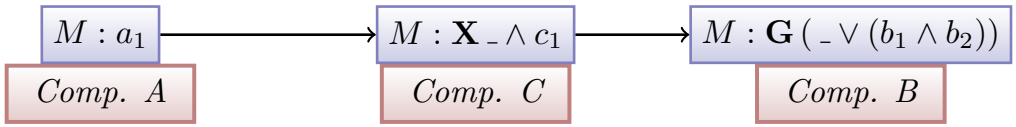
Migration (ctd)



At each component with a formula φ to process, at each time step:

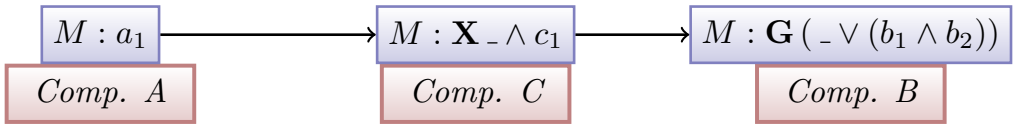
1. Use the current local observations to resolve relevant propositions.
2. Use the local history to resolve any past references to local observations.
3. Progress φ using "obligations" to earlier observations when not locally available.
4. If a verdict is reached, stop monitoring and report result.
5. Otherwise, select the component which can resolve the "oldest" obligation and send the formula to this component.

Choreography (simplified)



- Breaking down the formula across the network (following its syntax tree).
- *Tree structure* where results from subformulae flow up to the parent formula.

Choreography (simplified)



- Breaking down the formula across the network (following its syntax tree).
- *Tree structure* where results from subformulae flow up to the parent formula.

At each time instant, on each component:

1. If a verdict from a child is received:
   1. Substitute the verdict for the corresponding place holder in the local formula;
   2. Apply simplification rules to the local formula.
2. Progress the local formula using the local observation.
3. If the local formula reaches a verdict, send the verdict to the parent (if any).
4. If the formula at the root of the tree reaches a verdict, stop monitoring and report result.

Outline – Decentralised Monitoring of LTL formulae

Monitoring by progression

Definition (*Progression function* $P : LTL \times \Sigma \rightarrow LTL$)

Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}$, and $\sigma \in \Sigma$ be an event.

$$
\begin{array}{lcl}
P(p \in AP, \sigma) & = & \top, \text{ if } p \in \sigma, \bot \text{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) & = & P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) & = & P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \\
P(\mathbf{G}\varphi, \sigma) & = & P(\varphi, \sigma) \wedge \mathbf{G}(\varphi) \\
P(\mathbf{F}\varphi, \sigma) & = & P(\varphi, \sigma) \vee \mathbf{F}(\varphi)
\end{array}
\qquad
\begin{array}{lcl}
P(\top, \sigma) & = & \top \\
P(\bot, \sigma) & = & \bot \\
P(\neg\varphi, \sigma) & = & \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) & = & \varphi
\end{array}
$$

Monitoring by progression

Definition (*Progression function* $P : LTL \times \Sigma \rightarrow LTL$)

Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}$, and $\sigma \in \Sigma$ be an event.

$$
\begin{array}{lcl}
P(p \in AP, \sigma) & = & \top, \text{ if } p \in \sigma, \bot \text{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) & = & P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) & = & P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \\
P(\mathbf{G}\varphi, \sigma) & = & P(\varphi, \sigma) \wedge \mathbf{G}(\varphi) \\
P(\mathbf{F}\varphi, \sigma) & = & P(\varphi, \sigma) \vee \mathbf{F}(\varphi)
\end{array}
\qquad
\begin{array}{lcl}
P(\top, \sigma) & = & \top \\
P(\bot, \sigma) & = & \bot \\
P(\neg\varphi, \sigma) & = & \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) & = & \varphi
\end{array}
$$

Example (Progression)

- Let $\varphi = \mathbf{G}(a \wedge b \vee c)$
- At time $t = 0$, let $u = \{a\}$

Monitoring by progression

Definition (*Progression function* $P : LTL \times \Sigma \rightarrow LTL$)

Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}$, and $\sigma \in \Sigma$ be an event.

$$
\begin{array}{lcl}
P(p \in AP, \sigma) & = & \top, \text{ if } p \in \sigma, \bot \text{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) & = & P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) & = & P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \\
P(\mathbf{G}\varphi, \sigma) & = & P(\varphi, \sigma) \wedge \mathbf{G}(\varphi) \\
P(\mathbf{F}\varphi, \sigma) & = & P(\varphi, \sigma) \vee \mathbf{F}(\varphi)
\end{array}
\qquad
\begin{array}{lcl}
P(\top, \sigma) & = & \top \\
P(\bot, \sigma) & = & \bot \\
P(\neg\varphi, \sigma) & = & \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) & = & \varphi
\end{array}
$$

Example (Progression)

- Let $\varphi = \mathbf{G}(a \wedge b \vee c)$
- At time $t = 0$, let $u = \{a\}$

$$
\begin{aligned}
P(\varphi, u) &= P(a \wedge b \vee c, u) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \big(P(a, u) \wedge P(b, u) \vee P(c, u)\big) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \bot \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \bot
\end{aligned}
$$

Monitoring by progression

Definition (*Progression function* $P : LTL \times \Sigma \rightarrow LTL$)

Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}$, and $\sigma \in \Sigma$ be an event.

$$
\begin{array}{lcl}
P(p \in AP, \sigma) & = & \top, \text{ if } p \in \sigma, \bot \text{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) & = & P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) & = & P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \\
P(\mathbf{G}\varphi, \sigma) & = & P(\varphi, \sigma) \wedge \mathbf{G}(\varphi) \\
P(\mathbf{F}\varphi, \sigma) & = & P(\varphi, \sigma) \vee \mathbf{F}(\varphi)
\end{array}
\qquad
\begin{array}{lcl}
P(\top, \sigma) & = & \top \\
P(\bot, \sigma) & = & \bot \\
P(\neg\varphi, \sigma) & = & \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) & = & \varphi
\end{array}
$$

Example (Progression)

- Let $\varphi = \mathbf{G}(a \wedge b \vee c)$
- At time $t = 0$, let $u = \{a, c\}$

$$
\begin{aligned}
P(\varphi, u) &= P(a \wedge b \vee c, u) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \big(P(a, u) \wedge P(b, u) \vee P(c, u)\big) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \top \wedge \mathbf{G}(a \wedge b \vee c) \\
&= \mathbf{G}(a \wedge b \vee c)
\end{aligned}
$$

Monitoring by progression

Progression provides a monitoring algorithm

$$P(P(\ldots P(\varphi, u(0))\ldots, u(n-1)), u(n)) = \top \implies u \in \mathrm{good}(\varphi)$$
$$P(P(\ldots P(\varphi, u(0))\ldots, u(n-1)), u(n)) = \bot \implies u \in \mathrm{bad}(\varphi)$$

Monitoring by progression

Progression provides a monitoring algorithm

$$P(P(\ldots P(\varphi, u(0))\ldots, u(n-1)), u(n)) = \top \implies u \in \mathrm{good}(\varphi)$$
$$P(P(\ldots P(\varphi, u(0))\ldots, u(n-1)), u(n)) = \bot \implies u \in \mathrm{bad}(\varphi)$$

Observe:

- Efficiency does not depend on length of trace, but
- Potential "formula explosion" problem
  $\hookrightarrow$ continuous syntactic simplification

Is (classical) progression adequate for migration?

Example (Non-adequacy of (classical) progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$

Is (classical) progression adequate for migration?

Example (Non-adequacy of (classical) progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply progression on each component in separation (with their local observation)

Is (classical) progression adequate for migration?

Example (Non-adequacy of (classical) progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply progression on each component in separation (with their local observation)
- Let's take a look at what happens on M_A:

$$\begin{aligned} \text{“}P_A(\varphi, u)\text{”} &= P_A(\varphi, \{a\}) \\ &= P_A(a \wedge b \vee c, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\ &= (\top \wedge \bot \vee \bot) \wedge \mathbf{G}(a \wedge b \vee c) \\ &= \bot \end{aligned}$$

Is (classical) progression adequate for migration?

Example (Non-adequacy of (classical) progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply progression on each component in separation (with their local observation)
- Let's take a look at what happens on M_A:

$$\begin{aligned} \text{“}P_A(\varphi, u)\text{”} &= P_A(\varphi, \{a\}) \\ &= P_A(a \wedge b \vee c, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\ &= (\top \wedge \bot \vee \bot) \wedge \mathbf{G}(a \wedge b \vee c) \\ &= \bot \end{aligned}$$

- However, u is not a bad prefix!

Decentralising progression on some component C_i

Not much changes except for atomic propositions...

Definition (Decentralised progression for atomic propositions)

On some component C_i with atomic propositions $\mathrm{AP_i}$

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

Decentralising progression on some component C_i

Not much changes except for atomic propositions...

Definition (Decentralised progression for atomic propositions)

On some component C_i with atomic propositions $\mathrm{AP_i}$

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

Definition (Decentralised progression for past goals)

On some component C_i with atomic propositions $\mathrm{AP_i}$

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

where $\Pi_i(\sigma(-m))$ is the event observed m times ago on C_i

Back to the example

Example (Adequacy of decentralised progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply *decentralised* progression on each component in separation (with their local observation)

Back to the example

Example (Adequacy of decentralised progression)

- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply *decentralised* progression on each component in separation (with their local observation)
- Let's take a look at what happens on M_A:

$$
\begin{aligned}
\text{“}P_A(\varphi, u)\text{”} &= P_A(\varphi, \{a\}) \\
&= P_A(a \wedge b \vee c, \{a\}, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= P_A(a \wedge b \vee c, \{a\}, \{a\}) \wedge P_A(a \wedge b \vee c, \{b\}, \{a\}) \\
&\quad \wedge P_A(a \wedge b \vee c, \{c\}, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= (\top \wedge \overline{\mathbf{X}} b \vee \overline{\mathbf{X}} c) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= (\overline{\mathbf{X}} b \vee \overline{\mathbf{X}} c) \wedge \mathbf{G}(a \wedge b \vee c)
\end{aligned}
$$

Back to the example

Example (Adequacy of decentralised progression)

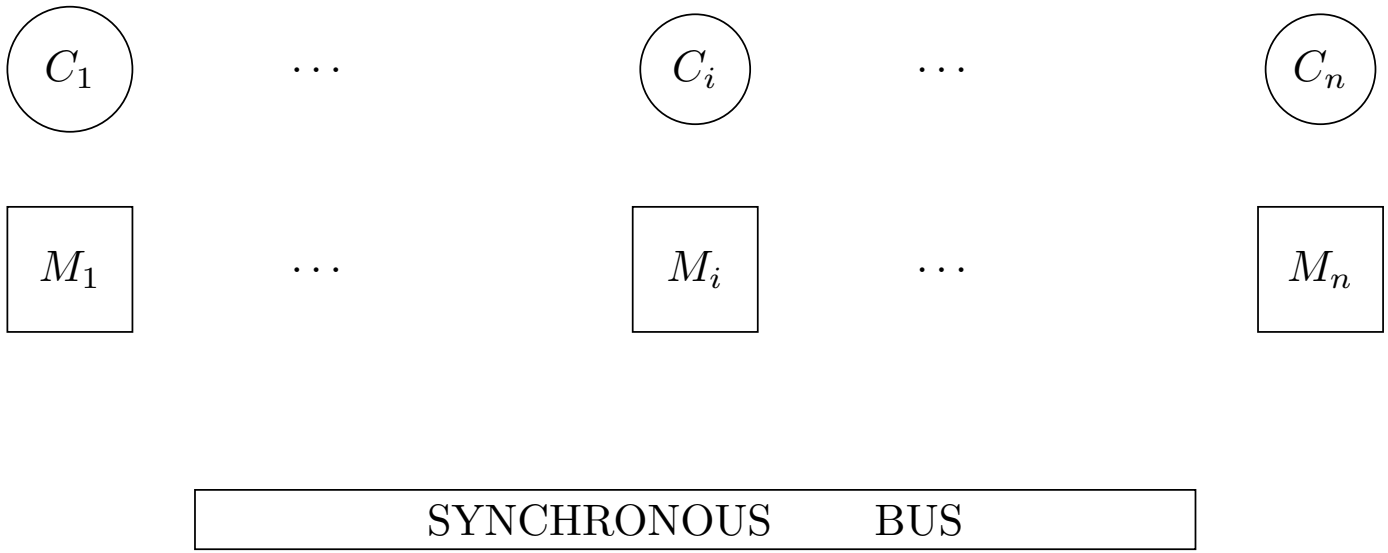
- Architecture with components A, B, C, resp. observing propositions a, b, c
- At time $t = 0$, $u = \{a, c\}$ and $\varphi = \mathbf{G}(a \wedge b \vee c)$
- We apply *decentralised* progression on each component in separation (with their local observation)
- Let's take a look at what happens on M_A:

$$
\begin{aligned}
\text{“}P_A(\varphi, u)\text{”} &= P_A(\varphi, \{a\}) \\
&= P_A(a \wedge b \vee c, \{a\}, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= P_A(a \wedge b \vee c, \{a\}, \{a\}) \wedge P_A(a \wedge b \vee c, \{b\}, \{a\}) \\
&\quad \wedge P_A(a \wedge b \vee c, \{c\}, \{a\}) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= (\top \wedge \overline{\mathbf{X}} b \vee \overline{\mathbf{X}} c) \wedge \mathbf{G}(a \wedge b \vee c) \\
&= (\overline{\mathbf{X}} b \vee \overline{\mathbf{X}} c) \wedge \mathbf{G}(a \wedge b \vee c)
\end{aligned}
$$

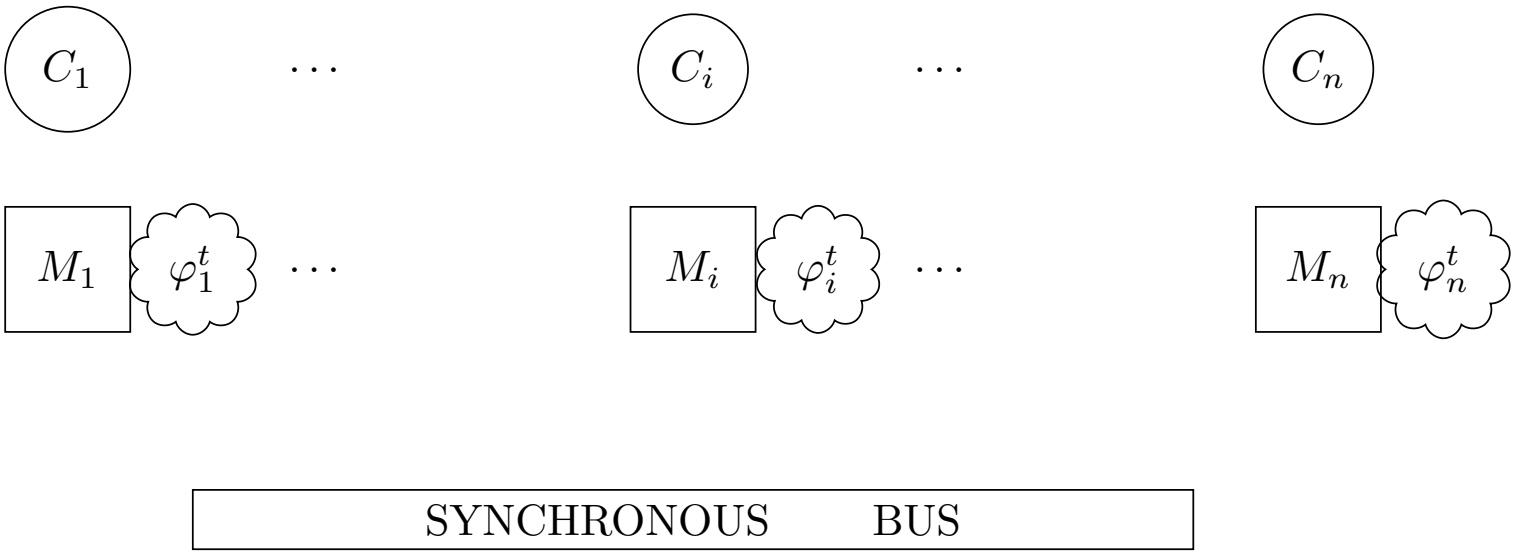
- Monitoring can continue :-)

Outline – Decentralised Monitoring of LTL formulae

Decentralised Monitoring: local algorithm at time t

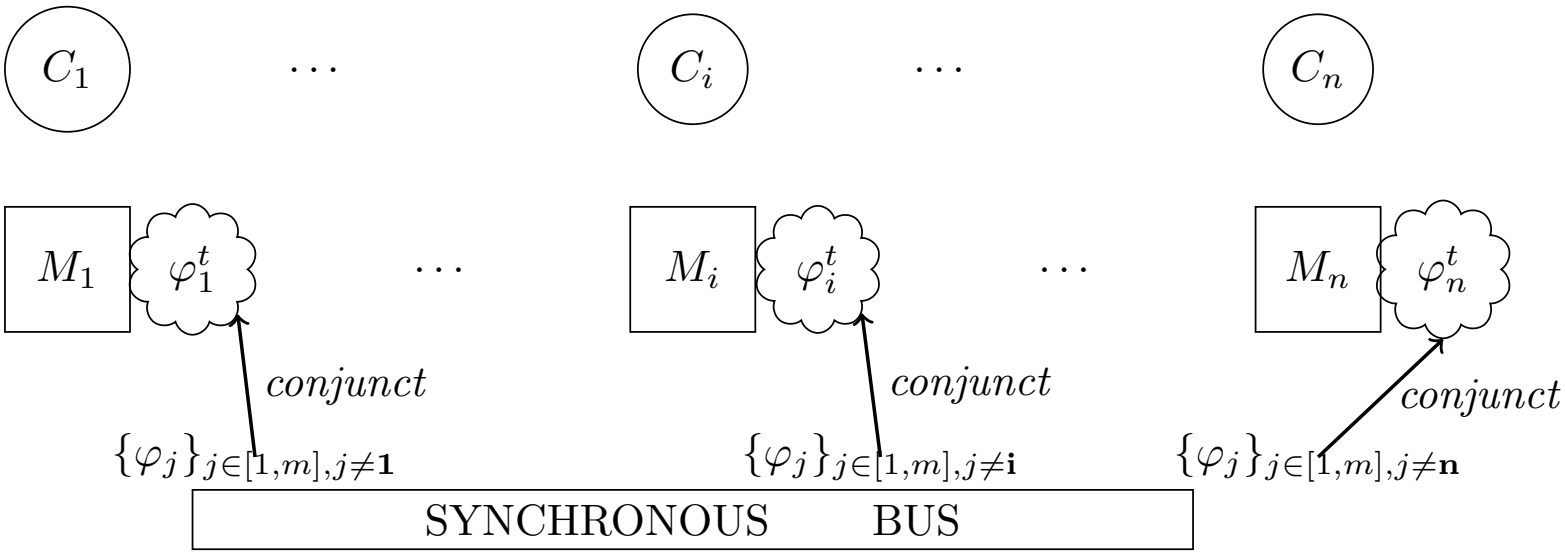


Decentralised Monitoring: local algorithm at time t



L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

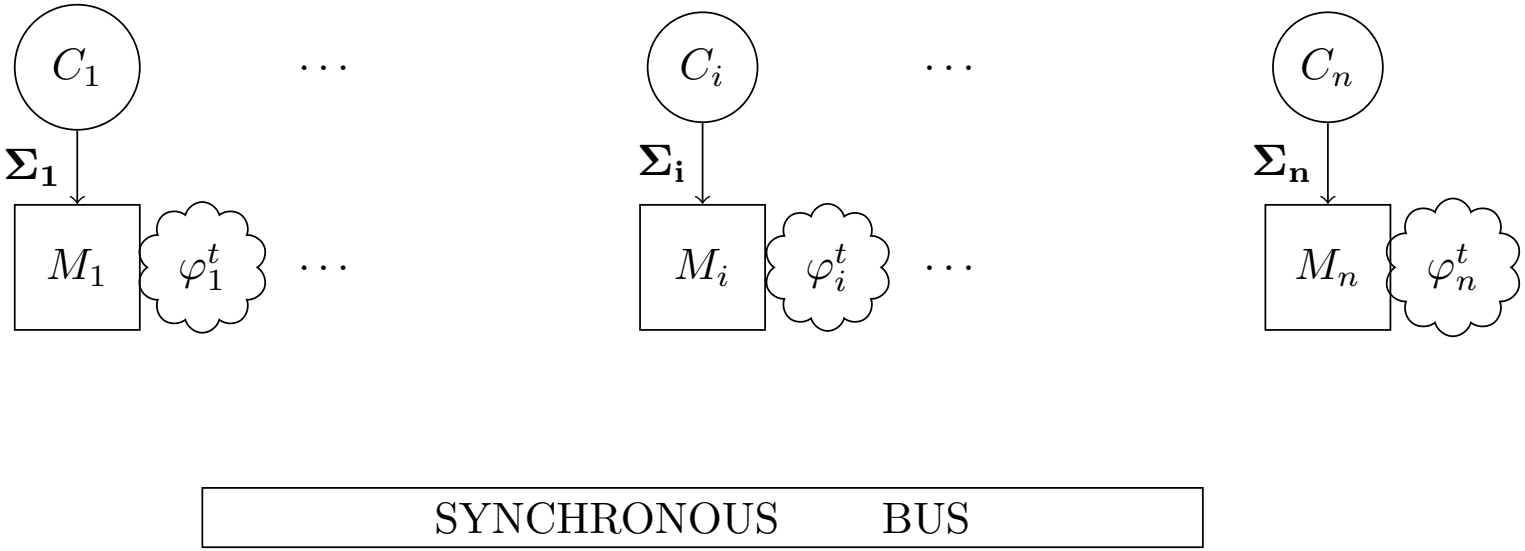
Decentralised Monitoring: local algorithm at time t



L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

L2. [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

Decentralised Monitoring: local algorithm at time t

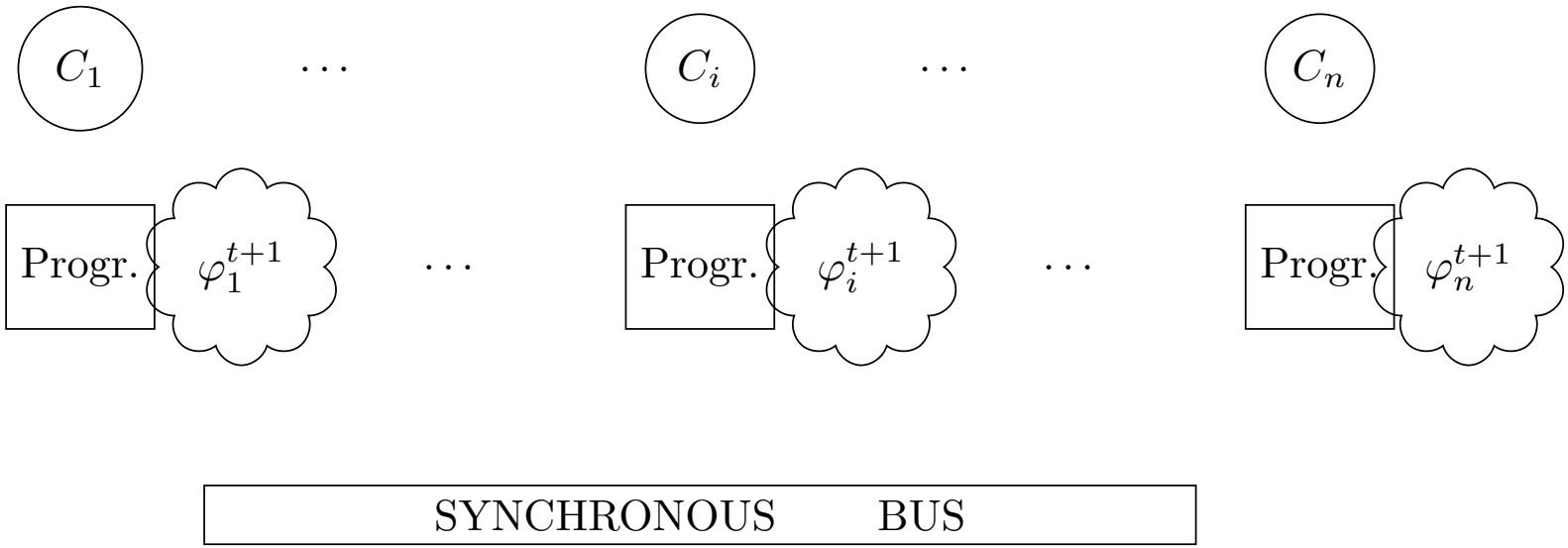


L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

L2. [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

L3. [Receive event.] Read next σ

Decentralised Monitoring: local algorithm at time t



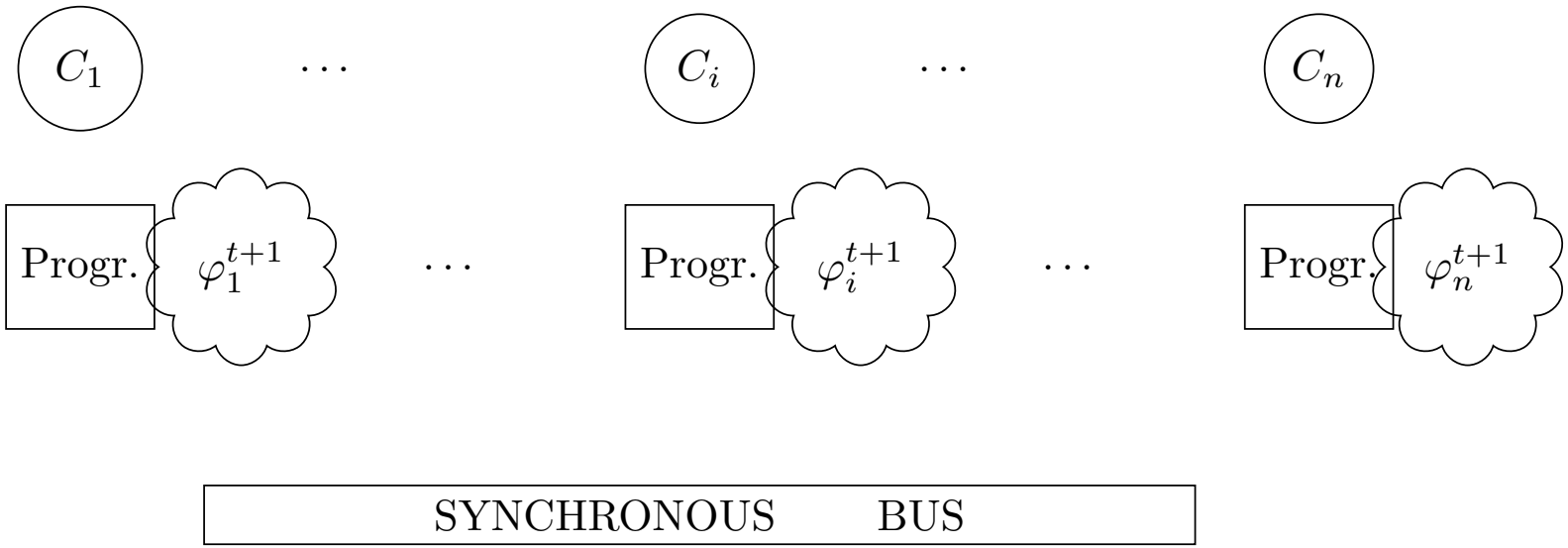
L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

L2. [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

L3. [Receive event.] Read next σ

L4. [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

Decentralised Monitoring: local algorithm at time t



L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

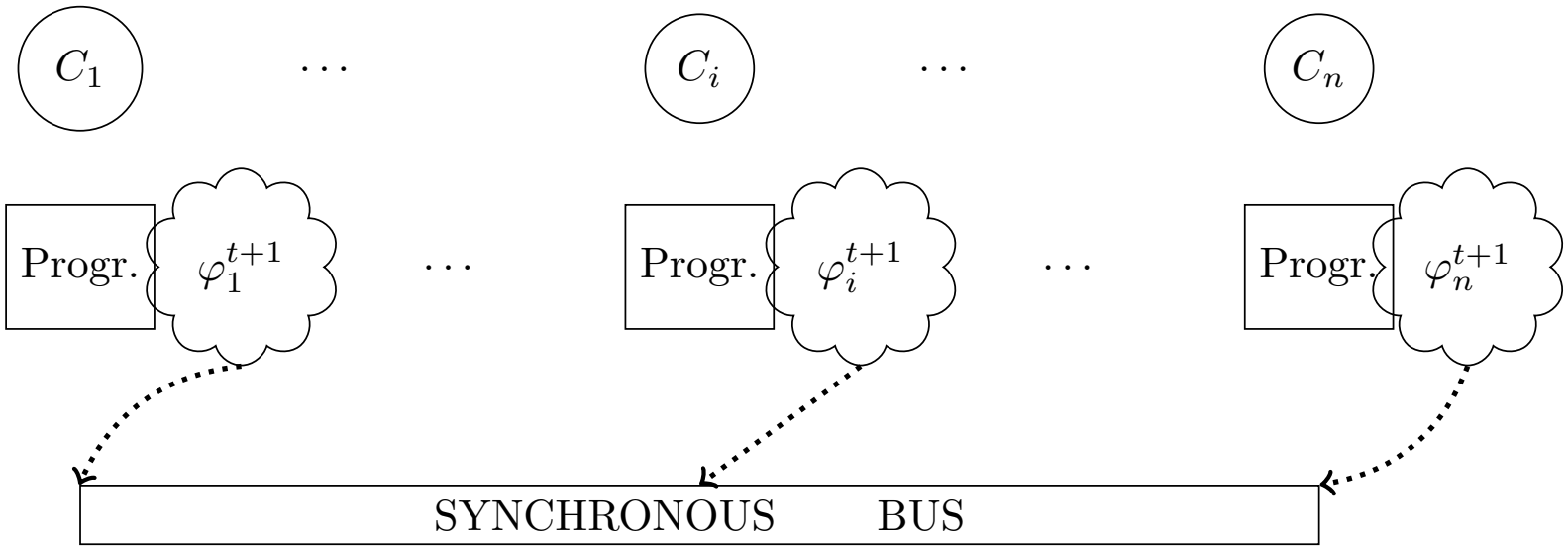
L2. [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

L3. [Receive event.] Read next σ

L4. [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

L5. [Evaluate and return.] If $\varphi_i^{t+1} = \top$ return $\top$, if $\varphi_i^{t+1} = \bot$ return $\bot$

Decentralised Monitoring: local algorithm at time t



L1. [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

L2. [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

L3. [Receive event.] Read next σ

L4. [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

L5. [Evaluate and return.] If $\varphi_i^{t+1} = \top$ return $\top$, if $\varphi_i^{t+1} = \bot$ return $\bot$

L6. [Communicate.] If φ_i^{t+1} is *urgent* send it to the most "relevant" monitor

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

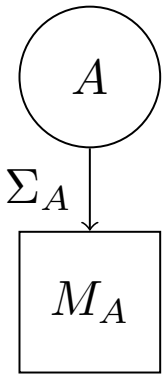
Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

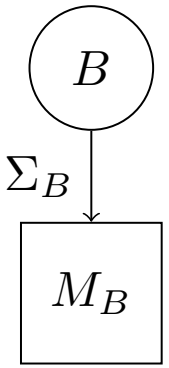
- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP}_{\mathrm{A}} = \{a\}, \mathrm{AP}_{\mathrm{B}} = \{b\}, \mathrm{AP}_{\mathrm{C}} = \{c\}$

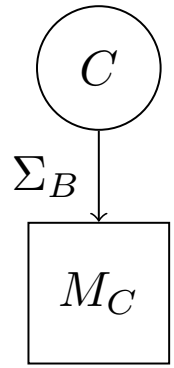
Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$





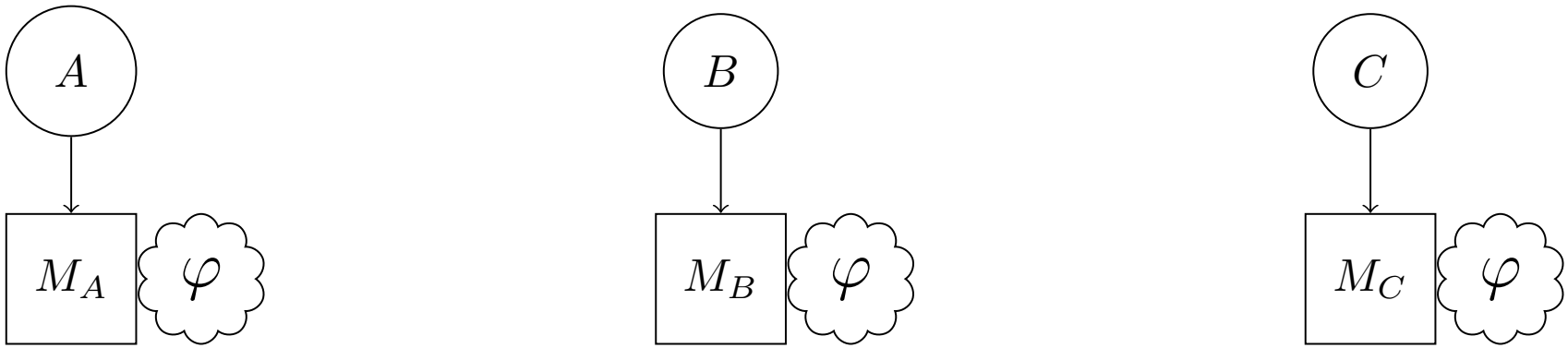


Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



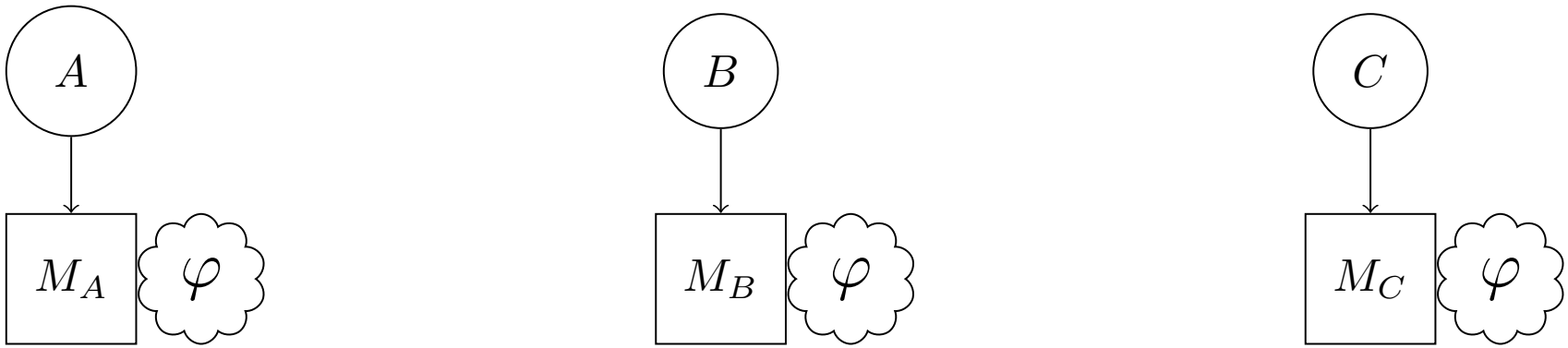
[L1.] [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\boxed{\mathrm{t} = 0}$



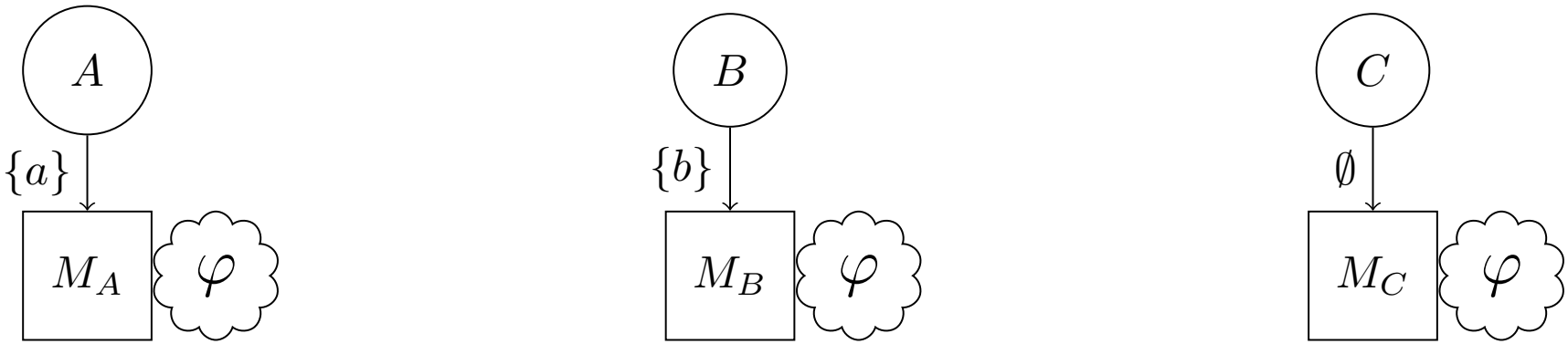
[L2.] [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



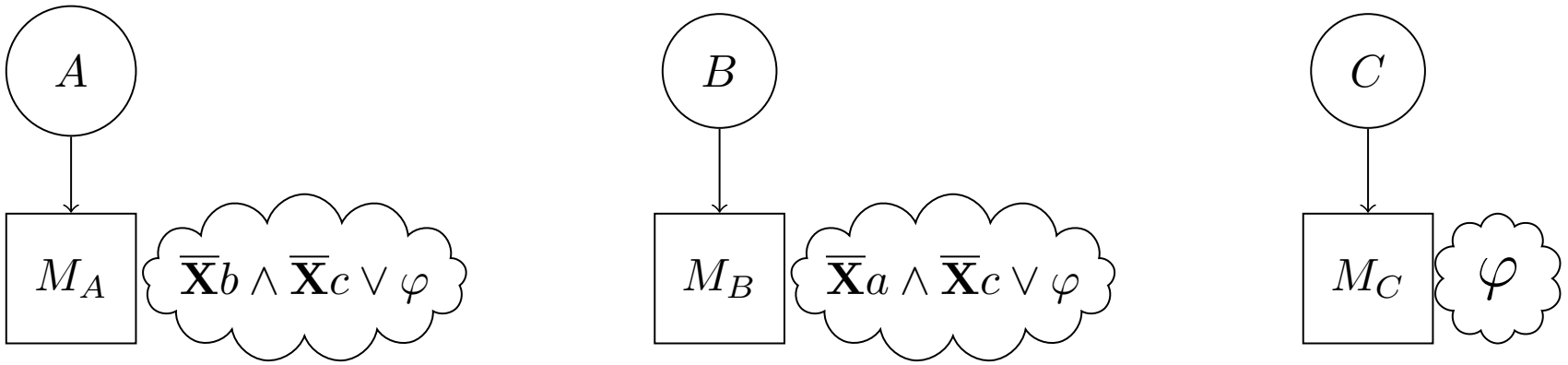
[L3.] [Receive event.] Read next σ

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

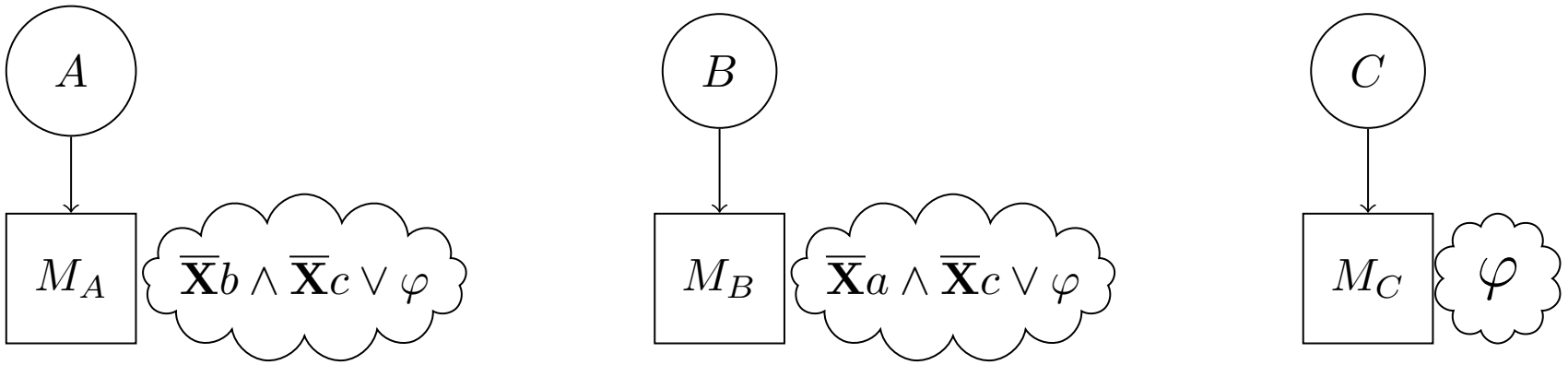
- $\varphi_A^1 := P(\varphi, \{a\}, \mathrm{AP_A}) = P(a \wedge b \wedge c, \{a\}, \mathrm{AP_A}) \vee \varphi = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

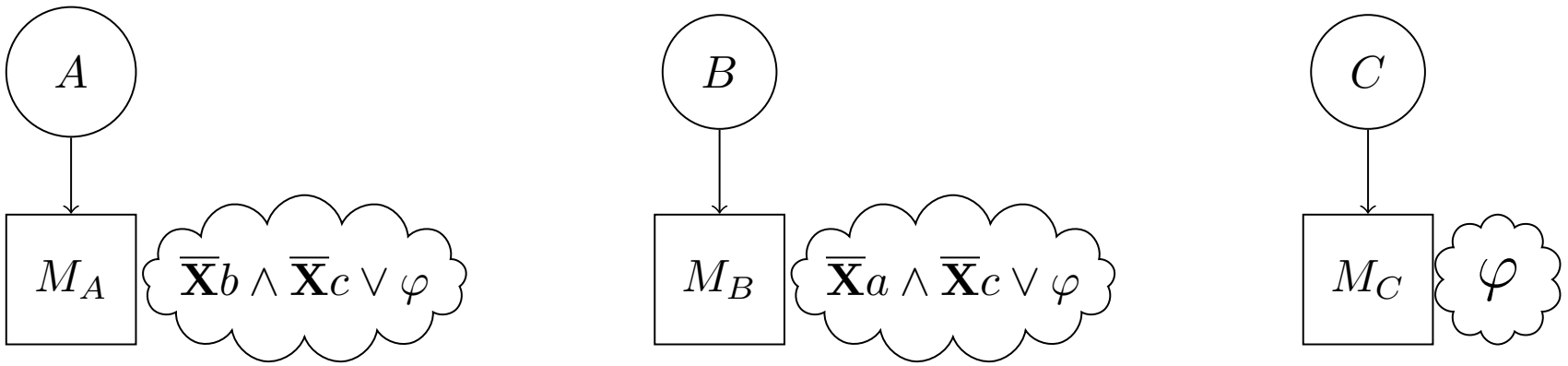
- $\varphi_A^1 := P(\varphi, \{a\}, \mathrm{AP_A}) = P(a \wedge b \wedge c, \{a\}, \mathrm{AP_A}) \vee \varphi = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$
- $\varphi_B^1 := P(\varphi, \{b\}, \mathrm{AP_B}) = P(a \wedge b \wedge c, \{b\}, \mathrm{AP_B}) \vee \varphi = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

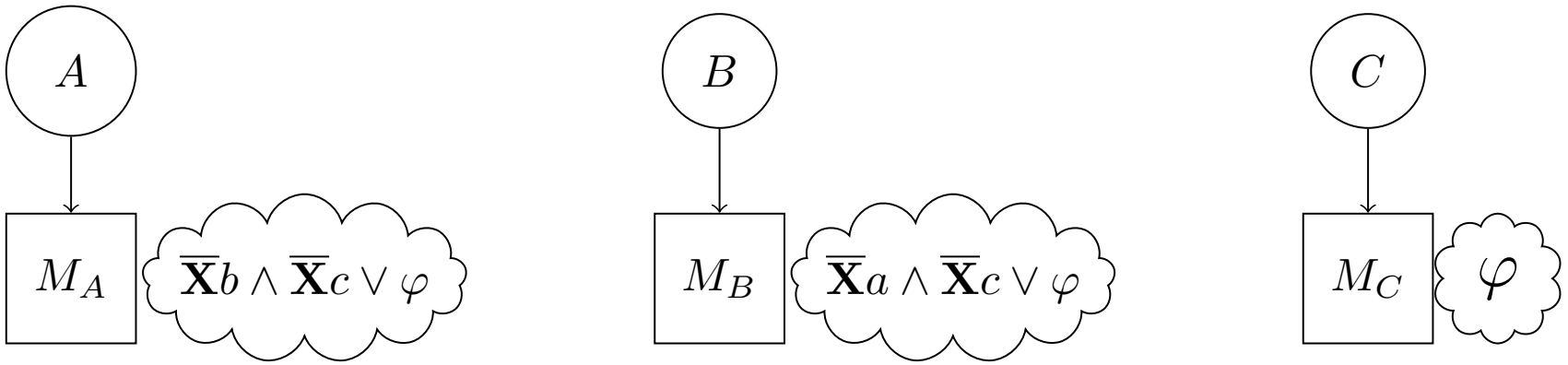
- $\varphi_A^1 := P(\varphi, \{a\}, \mathrm{AP_A}) = P(a \wedge b \wedge c, \{a\}, \mathrm{AP_A}) \vee \varphi = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$
- $\varphi_B^1 := P(\varphi, \{b\}, \mathrm{AP_B}) = P(a \wedge b \wedge c, \{b\}, \mathrm{AP_B}) \vee \varphi = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$
- $\varphi_C^1 := P(\varphi, \emptyset, \mathrm{AP_C}) = P(a \wedge b \wedge c, \emptyset, \mathrm{AP_C}) \vee \varphi = \varphi$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



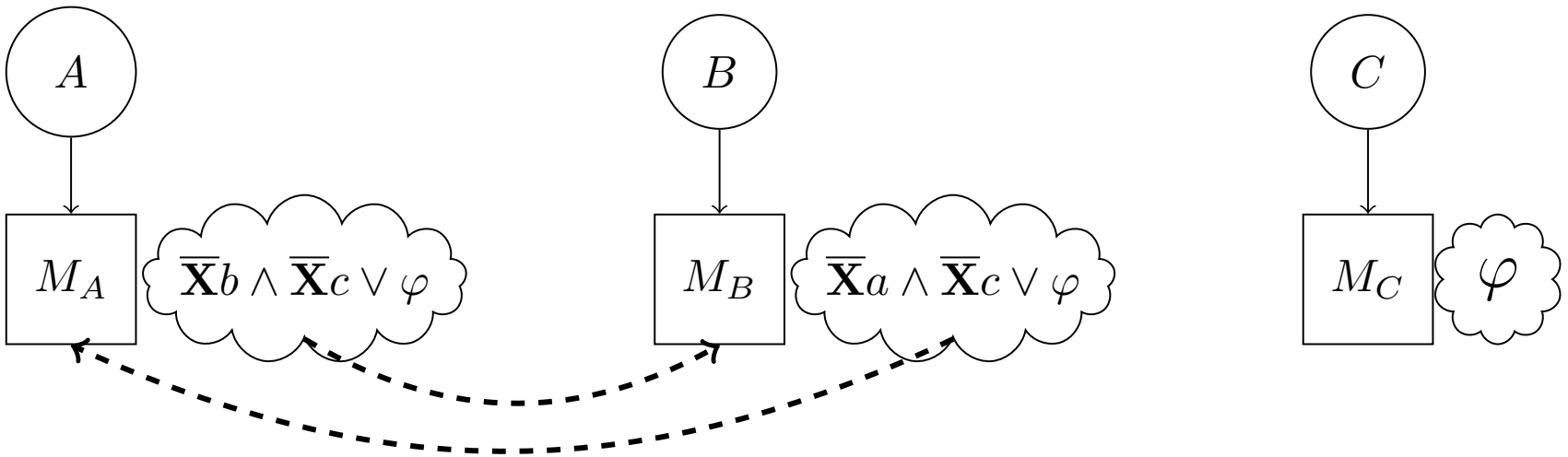
[L5.] [Evaluate and return.] If $\varphi_i^{t+1} = \top$ return $\top$, if $\varphi_i^{t+1} = \bot$ return $\bot$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 0}$



[L6.] [Communicate.] If φ_i^{t+1} is *urgent* send it to the most "relevant" monitor

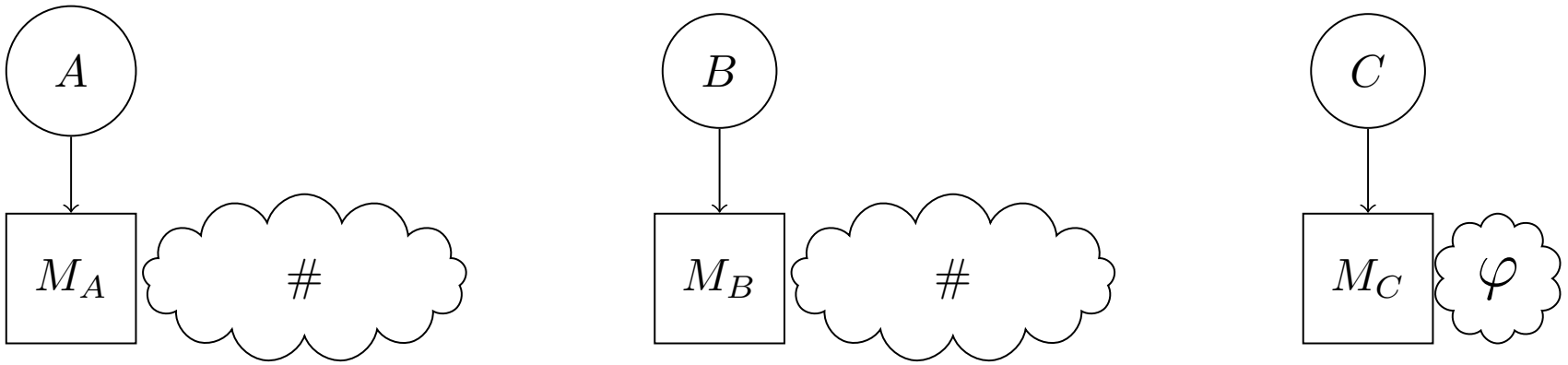
- $\mathrm{urgency}(\varphi_{\mathrm{A}}^1) = \mathrm{urgency}(\overline{\mathbf{X}}\mathrm{b} \wedge \overline{\mathbf{X}}\mathrm{c} \vee \varphi) = 1 \rightsquigarrow M_B$
- $\mathrm{urgency}(\varphi_{\mathrm{B}}^1) = \mathrm{urgency}(\overline{\mathbf{X}}\mathrm{a} \wedge \overline{\mathbf{X}}\mathrm{c} \vee \varphi) = 1 \rightsquigarrow M_A$
- $\mathrm{urgency}(\varphi_{\mathrm{C}}^1) = \mathrm{urgency}(\varphi) = 0$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



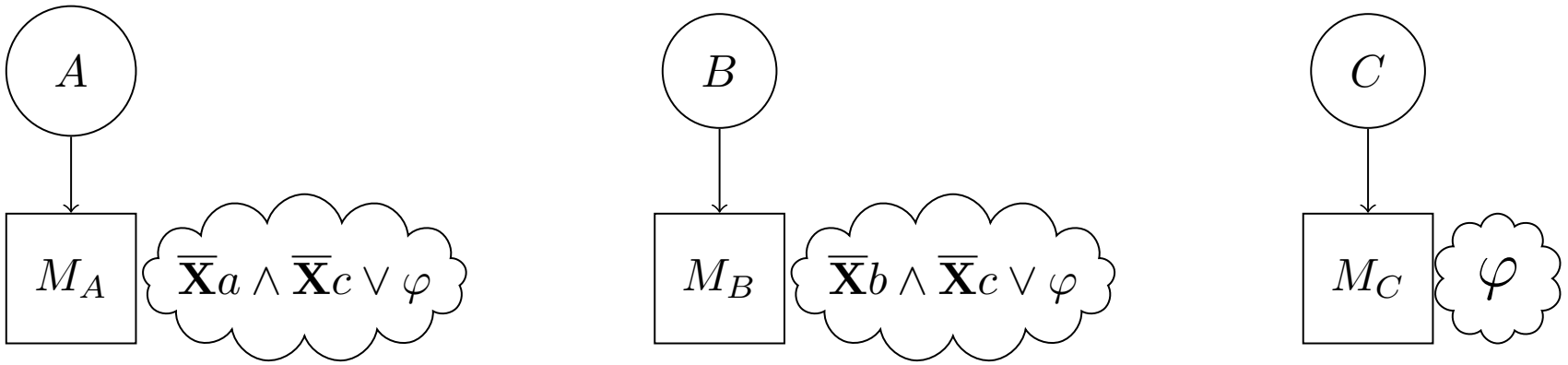
[L1.] [Next goal.] Let φ_i^t be the monitor's current local obligation ($\varphi_i^0 := \varphi$)

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathtt{t} = 1$



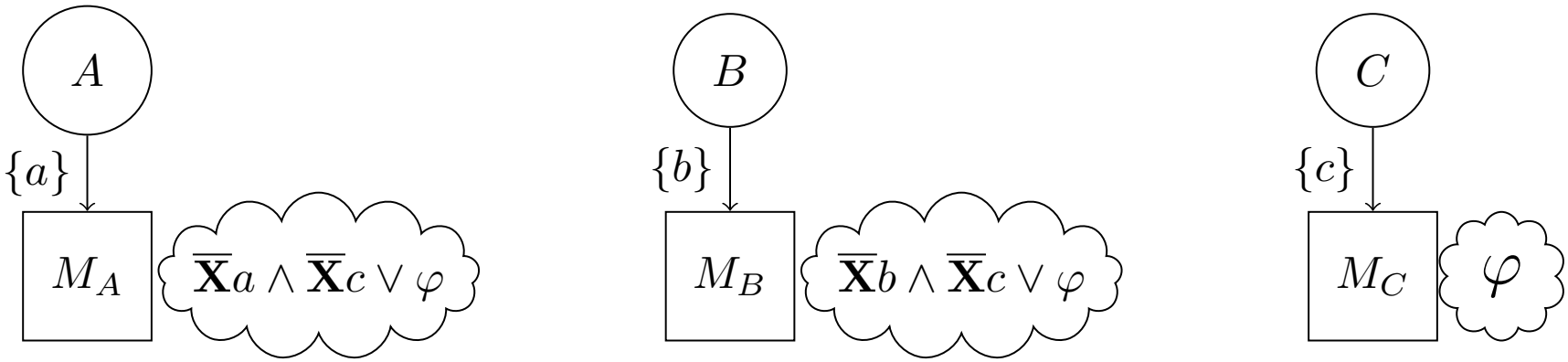
[L2.] [Receive messages.] ($\{\varphi_j\}_{j\in[1,m],j\neq i}$: received obligations)
Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j\in[1,m],j\neq i} \varphi_j$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



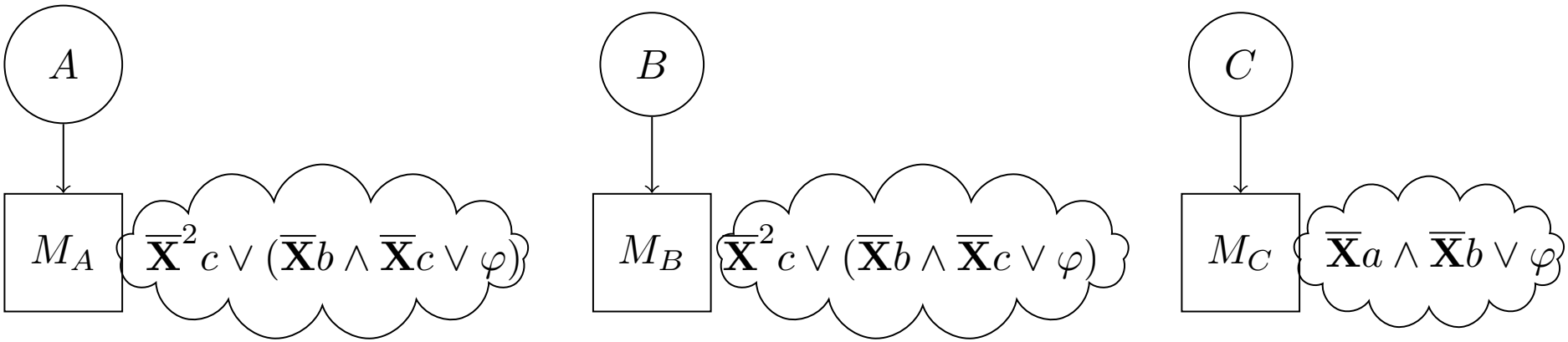
[L3.] [Receive event.] Read next σ

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\boxed{\mathrm{t} = 1}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

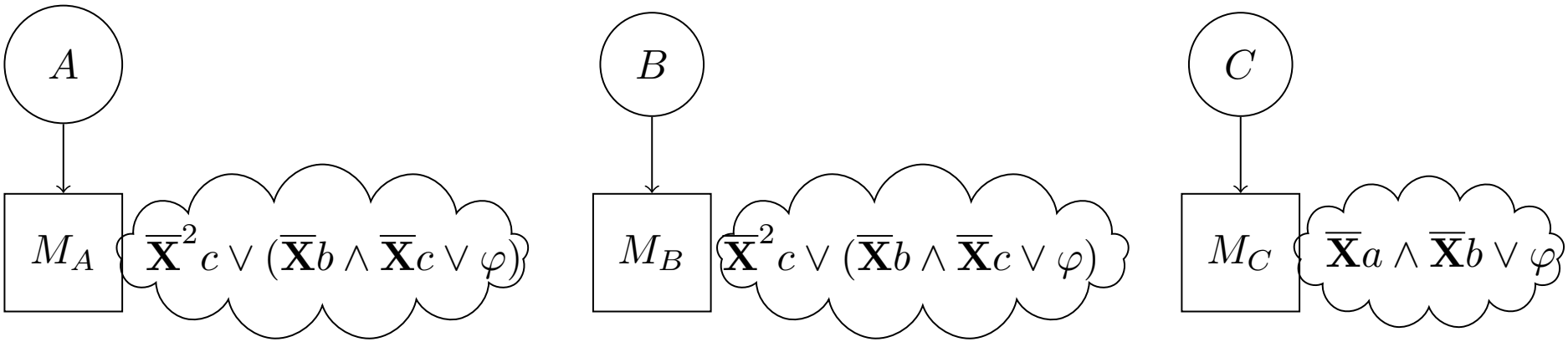
- $\varphi_A^2 := P\big(\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi \wedge \#, \{a\}, \mathrm{AP_A}\big) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

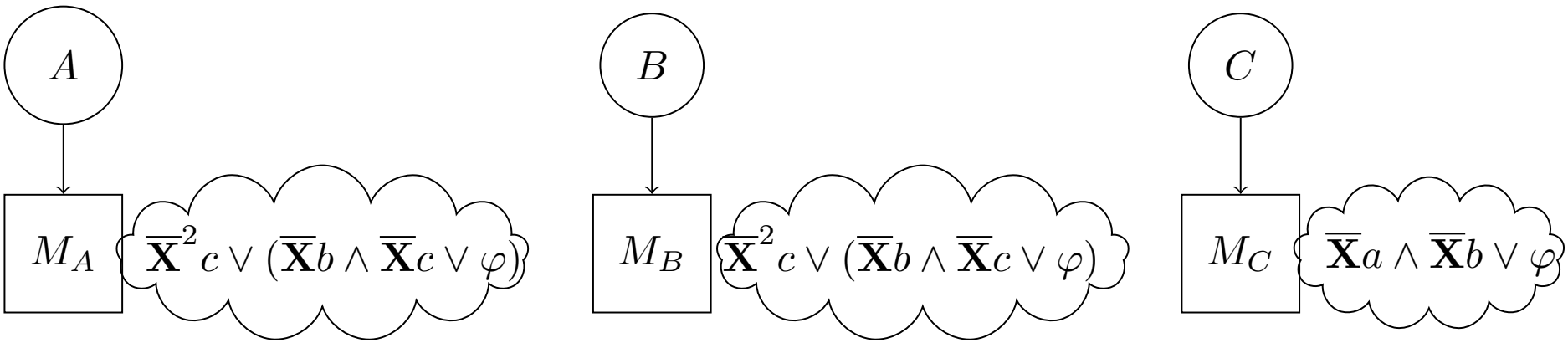
- $\varphi_A^2 := P\big(\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi \wedge \#, \{a\}, \mathrm{AP_A}\big) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
- $\varphi_B^2 := P(\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi \wedge \#, \{b\}, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi)$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



[L4.] [Progress.] Let the rewriting engine determine $\varphi_i^{t+1} := P(\varphi_i^t, \sigma, \mathrm{AP_i})$

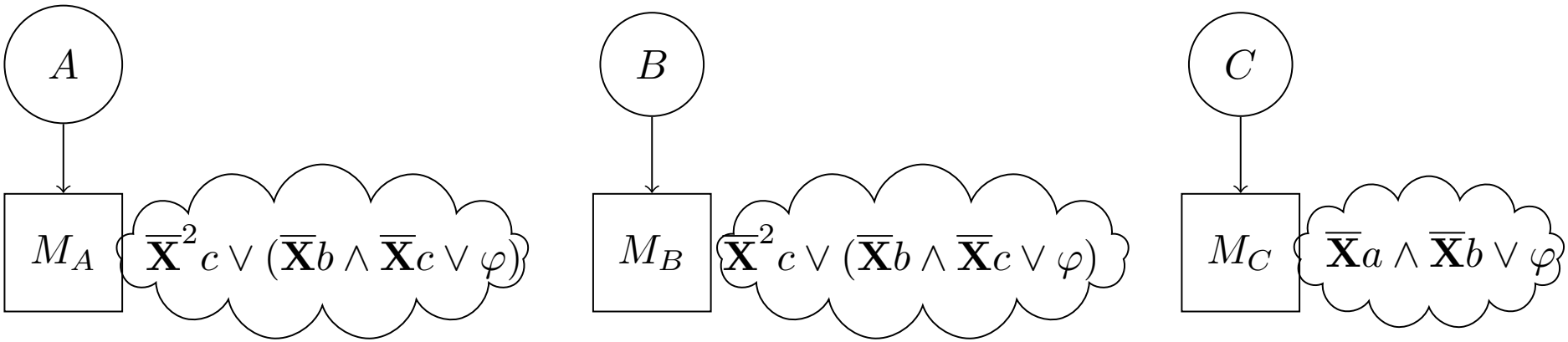
- $\varphi_A^2 := P\big(\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi \wedge \#, \{a\}, \mathrm{AP_A}\big) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
- $\varphi_B^2 := P(\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi \wedge \#, \{b\}, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi)$
- $\varphi_C^2 := P(\varphi, \{c\}, \mathrm{AP_C}) = \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} b \vee \varphi$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



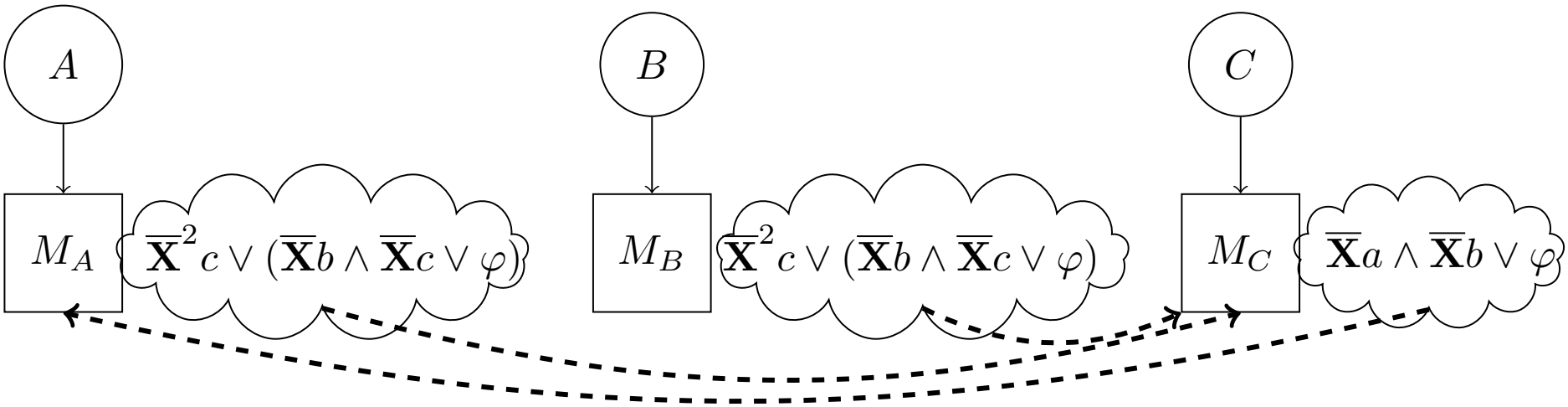
[L5.] [Evaluate and return.] If $\varphi_i^{t+1} = \top$ return $\top$, if $\varphi_i^{t+1} = \bot$ return $\bot$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$

- over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
- with
  $\mathrm{AP_A} = \{a\}, \mathrm{AP_B} = \{b\}, \mathrm{AP_C} = \{c\}$

$\mathbf{t = 1}$



[L6.] [Communicate.] If φ_i^{t+1} is *urgent* send it to the most "relevant" monitor

- $\mathrm{urgency}(\overline{\mathbf{X}}^2 \mathrm{c} \vee (\overline{\mathbf{X}} \mathrm{a} \wedge \overline{\mathbf{X}} \mathrm{c} \vee \varphi)) = 2 \rightsquigarrow M_C$
- $\mathrm{urgency}(\overline{\mathbf{X}}^2 \mathrm{c} \vee (\overline{\mathbf{X}} \mathrm{a} \wedge \overline{\mathbf{X}} \mathrm{c} \vee \varphi)) = 2 \rightsquigarrow M_C$
- $\mathrm{urgency}(\overline{\mathbf{X}} \mathrm{a} \wedge \overline{\mathbf{X}} \mathrm{b} \vee \varphi) = 1 \rightsquigarrow M_A$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$ over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
with $\Sigma_A = \{a\}, \Sigma_B = \{b\}, \Sigma_C = \{c\}$

t:	0	1
σ:	$\{a, b\}$	$\{a, b, c\}$
M_A:	$\varphi_A^1 := P(\varphi, \{a\}, \mathrm{AP_A}) = P(a \wedge b \wedge c, \{a\}, \mathrm{AP_A}) \vee \varphi$ $= \overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_A^2 := P(\varphi_B^1 \wedge \#, \{a\}, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_B:	$\varphi_B^1 := P(\varphi, \{b\}, \mathrm{AP_B}) = P(a \wedge b \wedge c, \{b\}, \mathrm{AP_B}) \vee \varphi$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_B^2 := P(\varphi_A^1 \wedge \#, \{b\}, \mathrm{AP_B})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_C:	$\varphi_C^1 := P(\varphi, \{c\}, \mathrm{AP_C}) = P(a \wedge b \wedge c, \emptyset, \mathrm{AP_C}) \vee \varphi$ $= \varphi$	$\varphi_C^2 := P(\varphi, \{c\}, \mathrm{AP_C})$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} b \vee \varphi$

Decent. progress. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$, 3 components

Monitoring $\varphi = \mathbf{F}(a \wedge b \wedge c)$ over $\{a, b\} \cdot \{a, b, c\} \cdot \emptyset \cdot \emptyset$
with $\Sigma_A = \{a\}, \Sigma_B = \{b\}, \Sigma_C = \{c\}$

t:	0	1
σ:	$\{a, b\}$	$\{a, b, c\}$
M_A:	$\varphi_A^1 := P(\varphi, \{a\}, \mathrm{AP_A}) = P(a \wedge b \wedge c, \{a\}, \mathrm{AP_A}) \vee \varphi$ $= \overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_A^2 := P(\varphi_B^1 \wedge \#, \{a\}, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_B:	$\varphi_B^1 := P(\varphi, \{b\}, \mathrm{AP_B}) = P(a \wedge b \wedge c, \{b\}, \mathrm{AP_B}) \vee \varphi$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_B^2 := P(\varphi_A^1 \wedge \#, \{b\}, \mathrm{AP_B})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_C:	$\varphi_C^1 := P(\varphi, \{c\}, \mathrm{AP_C}) = P(a \wedge b \wedge c, \emptyset, \mathrm{AP_C}) \vee \varphi$ $= \varphi$	$\varphi_C^2 := P(\varphi, \{c\}, \mathrm{AP_C})$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} b \vee \varphi$

t:	2	3
σ:	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^3 := P(\varphi_C^2 \wedge \#, \emptyset, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$	$\varphi_A^4 := P(\varphi_C^3 \wedge \#, \emptyset, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_B:	$\varphi_B^3 := P(\#, \emptyset, \mathrm{AP_B})$ $= \#$	$\varphi_B^4 := P(\varphi_A^3 \wedge \#, \emptyset, \mathrm{AP_B})$ $= \top$
M_C:	$\varphi_C^3 := P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \emptyset, \mathrm{AP_C})$ $= \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	$\varphi_C^4 := P(\#, \emptyset, \mathrm{AP_C})$ $= \#$

Some properties of the algorithm

Let $\varphi \in \mathrm{LTL}$ and $u \in \Sigma^*$

What is the link between:

- $\models_3$: centralised LTL_3 semantics
- $\models_D$: decentralised LTL_3 semantics

Some properties of the algorithm

Let $\varphi \in \mathrm{LTL}$ and $u \in \Sigma^*$

What is the link between:

- $\models_3$: centralised LTL_3 semantics
- $\models_D$: decentralised LTL_3 semantics

Theorem (Soundness)

$$u \models_D \varphi = \top/\bot \quad \Rightarrow \quad u \models_3 \varphi = \top/\bot$$
$$u \models_3 \varphi = ? \quad \Rightarrow \quad u \models_D \varphi = ?$$

Some properties of the algorithm

Let $\varphi \in \mathrm{LTL}$ and $u \in \Sigma^*$

What is the link between:

- $\models_3$: centralised LTL_3 semantics
- $\models_D$: decentralised LTL_3 semantics

Theorem (Soundness)

$$u \models_D \varphi = \top/\bot \quad \Rightarrow \quad u \models_3 \varphi = \top/\bot$$
$$u \models_3 \varphi = ? \quad \Rightarrow \quad u \models_D \varphi = ?$$

Theorem (Completeness)

$$u \models_3 \varphi = \top/\bot \Rightarrow \exists u' \in \Sigma^*.\ |u'| \leq |\mathcal{M}| \wedge u \cdot u' \models_D \varphi = \top/\bot$$

How much a monitor has to remember?

Theorem (Maximum delay)

Let $\overline{\mathbf{X}}^m p \in \mathrm{LTL}$ *be a local obligation on some monitor* $M_i \in \mathcal{M}$
In the worst case, $m \leq \min(|\mathcal{M}|, t+1)$ *at any time* $t \in \mathbb{N}^{\geq 0}$

How much a monitor has to remember?

Theorem (Maximum delay)

Let $\overline{\mathbf{X}}^m p \in \mathrm{LTL}$ *be a local obligation on some monitor* $M_i \in \mathcal{M}$
In the worst case, $m \leq \min(|\mathcal{M}|, t+1)$ *at any time* $t \in \mathbb{N}^{\geq 0}$

This, at the same time, reflects the communication delay by which a decentralised monitor may come to a verdict!

How much a monitor has to remember?

Theorem (Maximum delay)

Let $\overline{\mathbf{X}}^m p \in \mathrm{LTL}$ *be a local obligation on some monitor* $M_i \in \mathcal{M}$
In the worst case, $m \leq \min(|\mathcal{M}|, t+1)$ *at any time* $t \in \mathbb{N}^{\geq 0}$

This, at the same time, reflects the communication delay by which a decentralised monitor may come to a verdict!

However

Unless, there could be a (possibly infinite) delay not due to communication:

- $\mathbf{XX}\mathit{true}$ and $\mathbf{G}(\mathit{true}\mathbf{U}(\mathbf{G}b \vee \mathbf{F}\neg b))$

How much a monitor has to remember?

Theorem (Maximum delay)

Let $\overline{\mathbf{X}}^m p \in \mathrm{LTL}$ *be a local obligation on some monitor* $M_i \in \mathcal{M}$
In the worst case, $m \leq \min(|\mathcal{M}|, t+1)$ *at any time* $t \in \mathbb{N}^{\geq 0}$

This, at the same time, reflects the communication delay by which a decentralised monitor may come to a verdict!

However

Unless, there could be a (possibly infinite) delay not due to communication:

- $\mathbf{XX}\mathit{true}$ and $\mathbf{G}(\mathit{true}\mathbf{U}(\mathbf{G}b \vee \mathbf{F}\neg b))$

Corollary

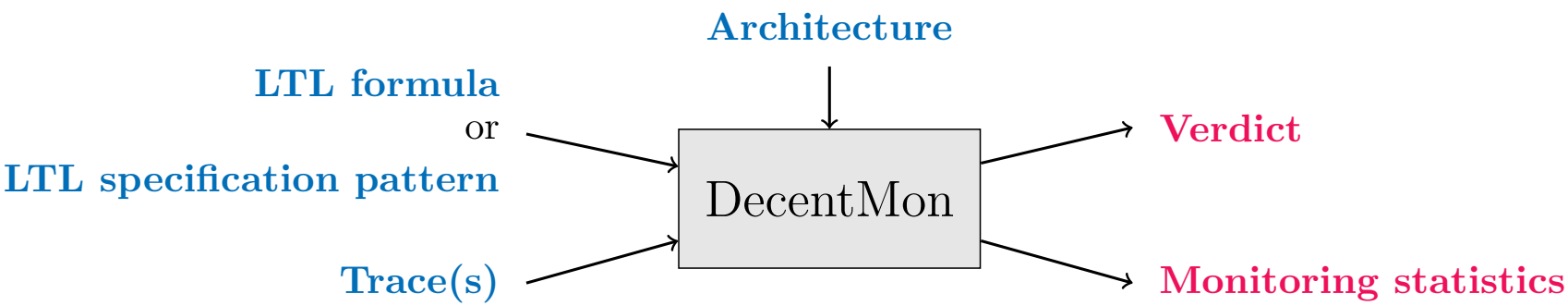
Given a "clean input": communication delay = memory requirements = verdict delay. (Otherwise, we can't say much at all.)

Outline – Decentralised Monitoring of LTL formulae

DecentMon: an OCaml benchmark

DecentMon: an OCaml benchmark simulating the decentralised algorithm

`http://decentmon.forge.imag.fr/`

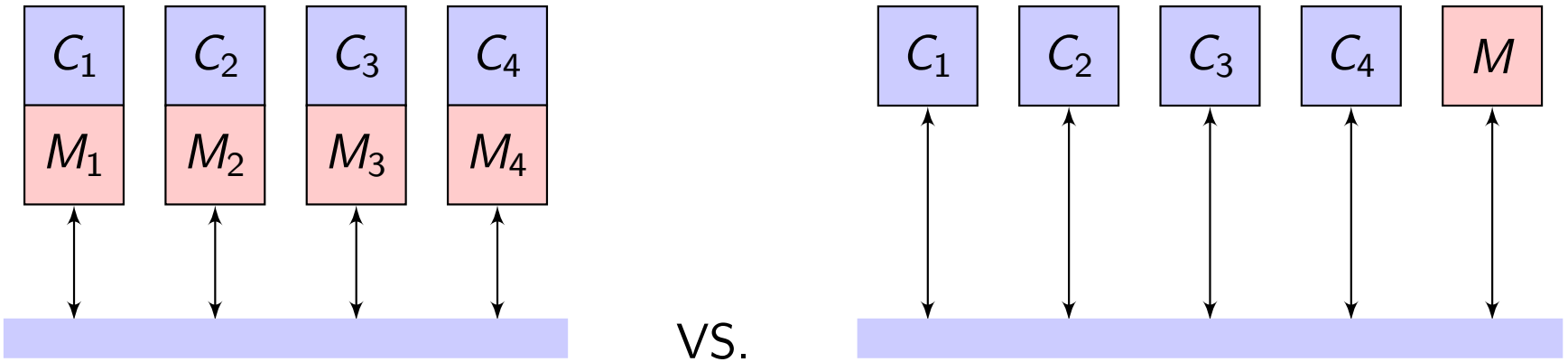


Occurrences of atomic propositions can be parameterised according to several *probability distributions*

What we wanted to compare

Two monitoring modes:

- **decentralised** mode (i.e., each trace is read by a separate monitor)
- **centralised** mode by merging the traces and using a "central monitor"

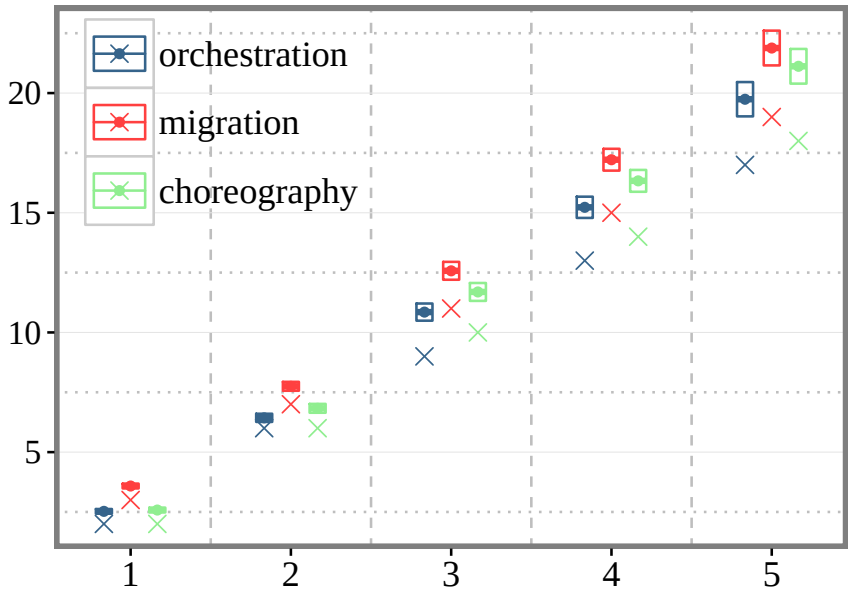


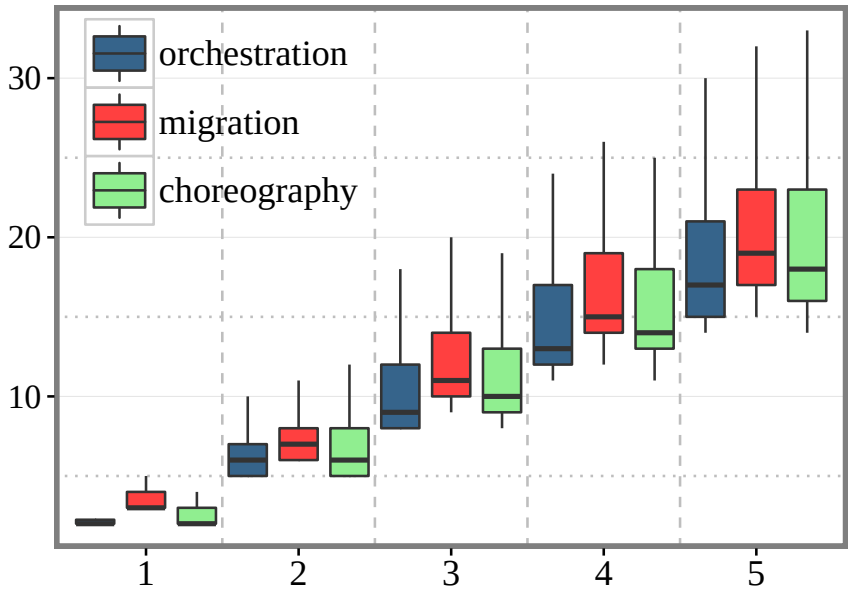
Four metrics:

- **length of the trace** needed to reach a verdict
- **number and size of messages** exchanged between monitors
- **number of progressions** performed by local monitors

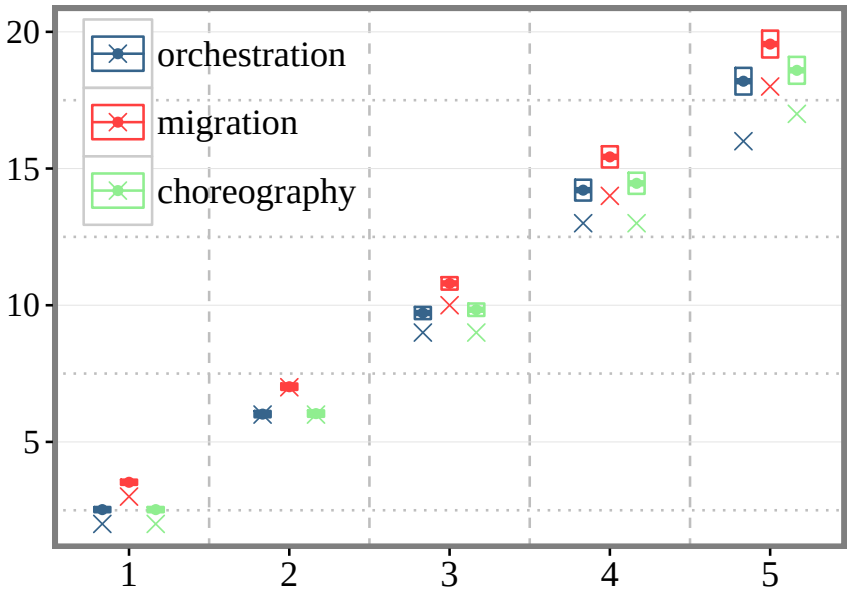
Experimental Results - trace length

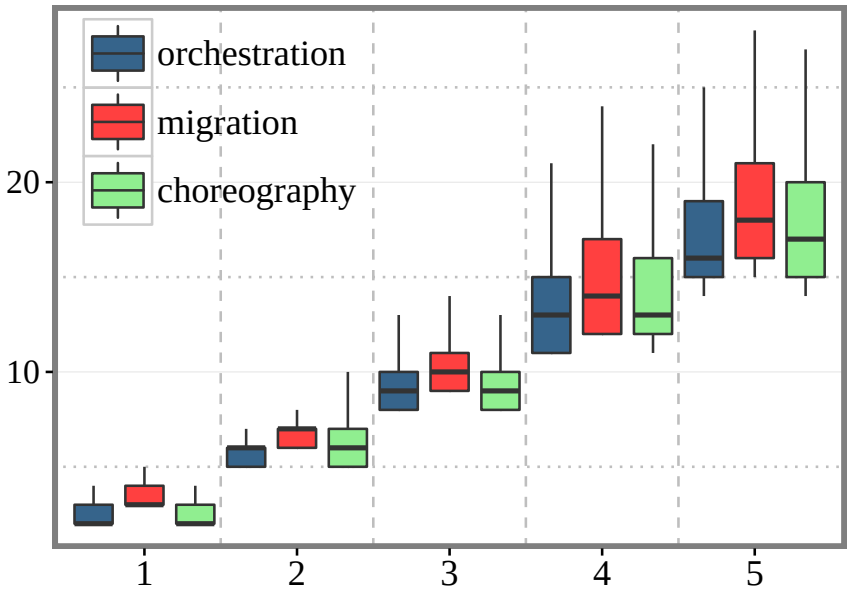
random formula generation





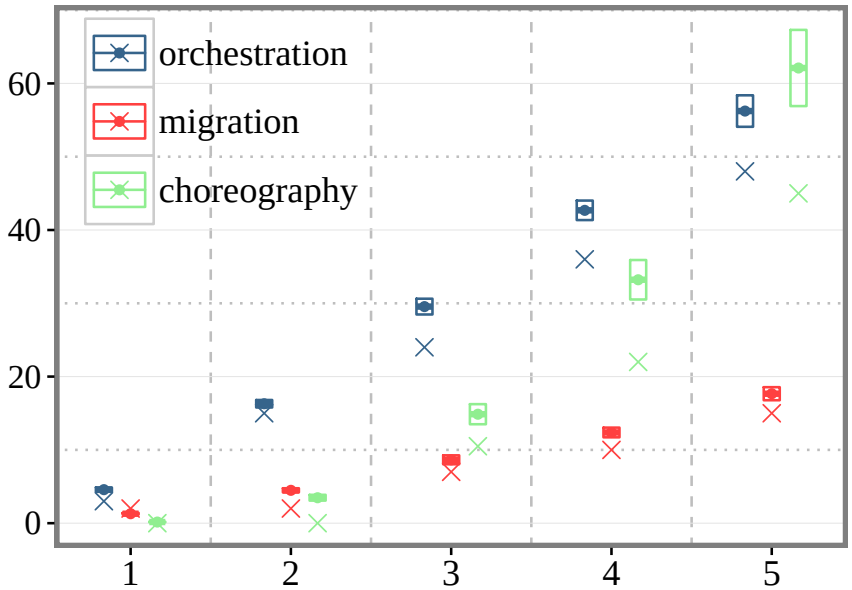
biased formula generation

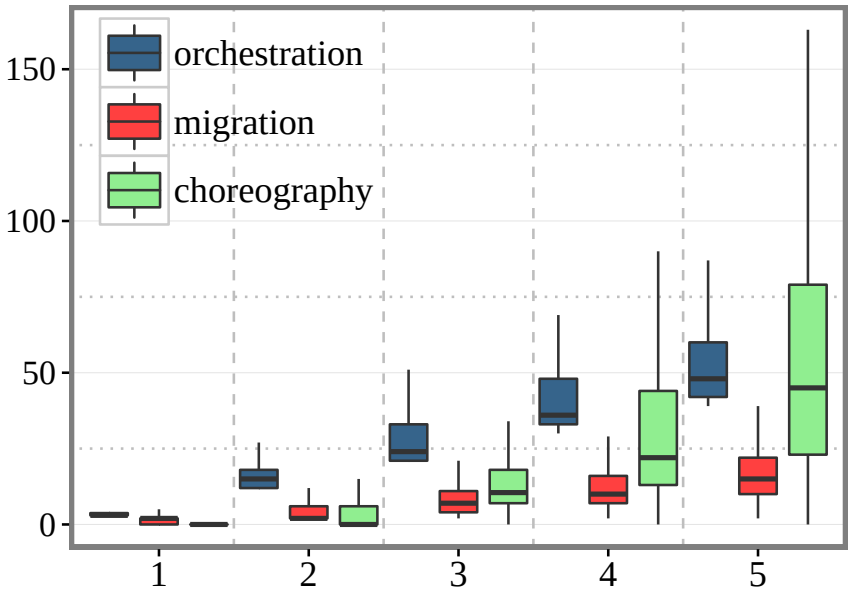




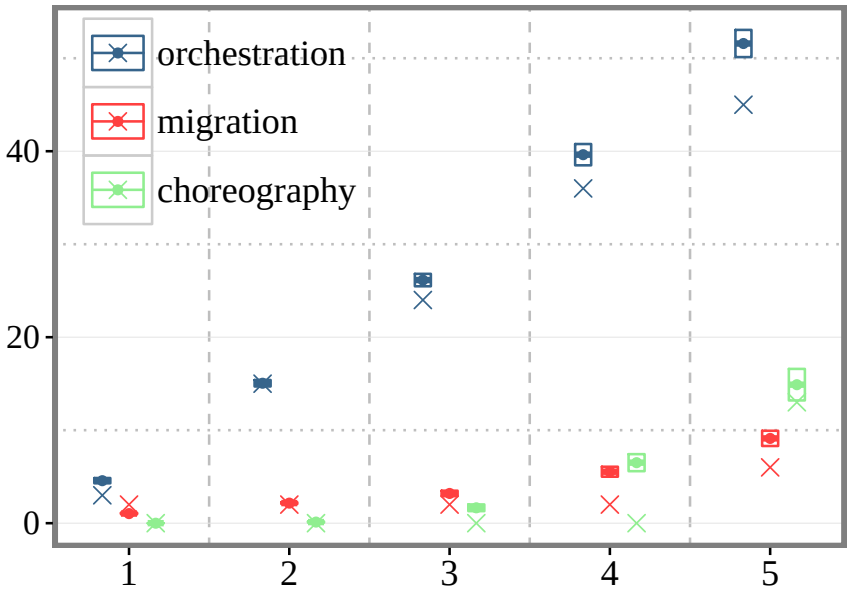
Experimental Results - number of messages

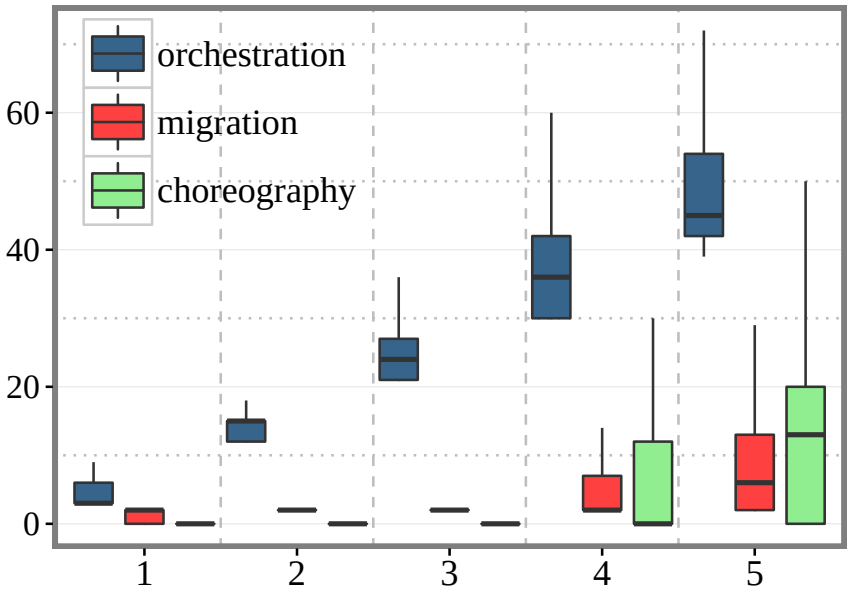
random formula generation





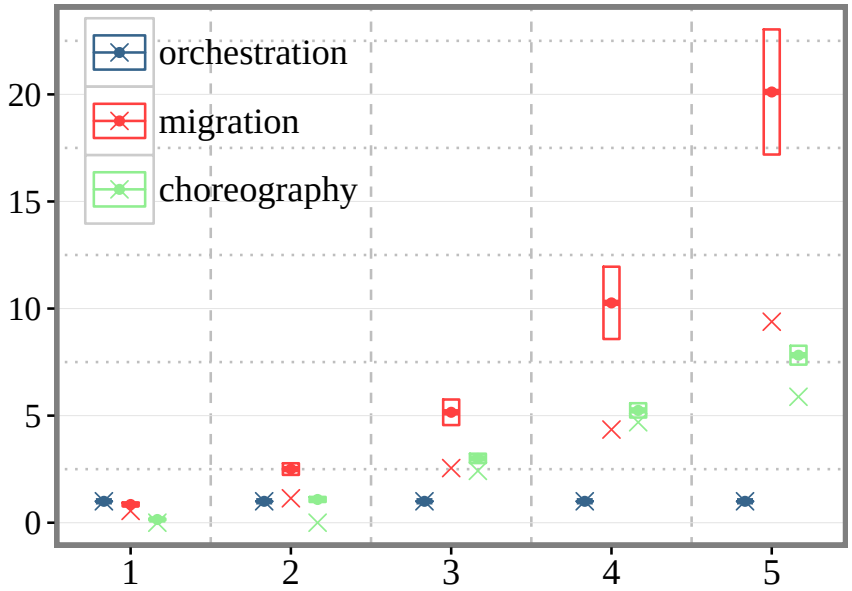
biased formula generation

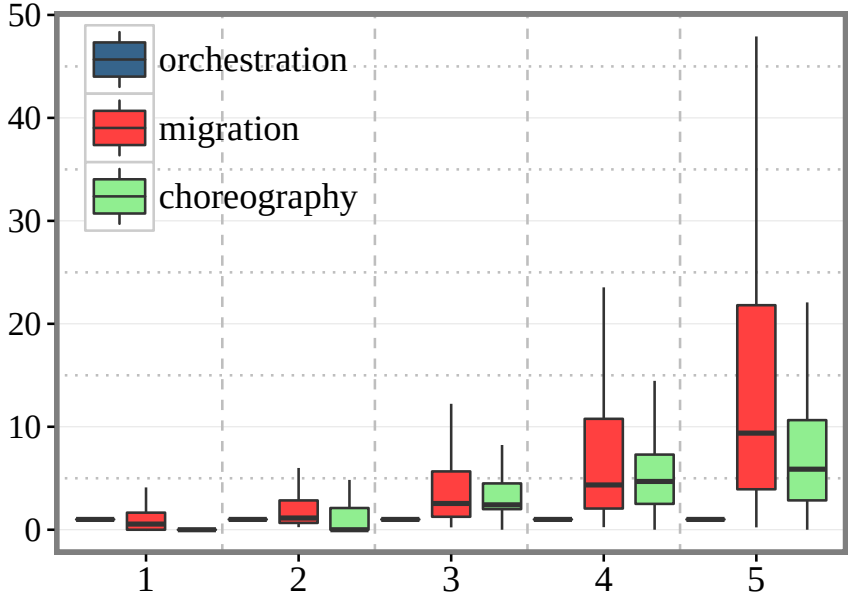




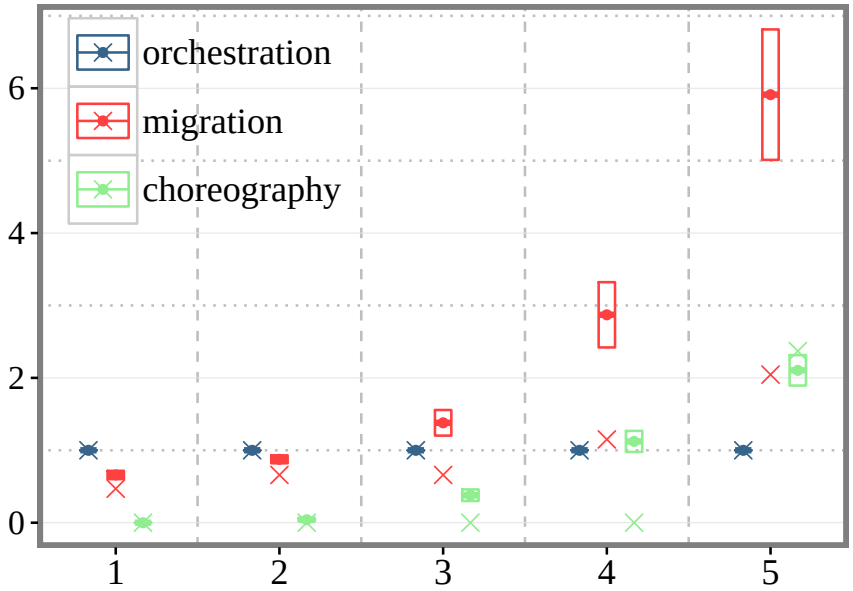
Experimental Results - size of messages

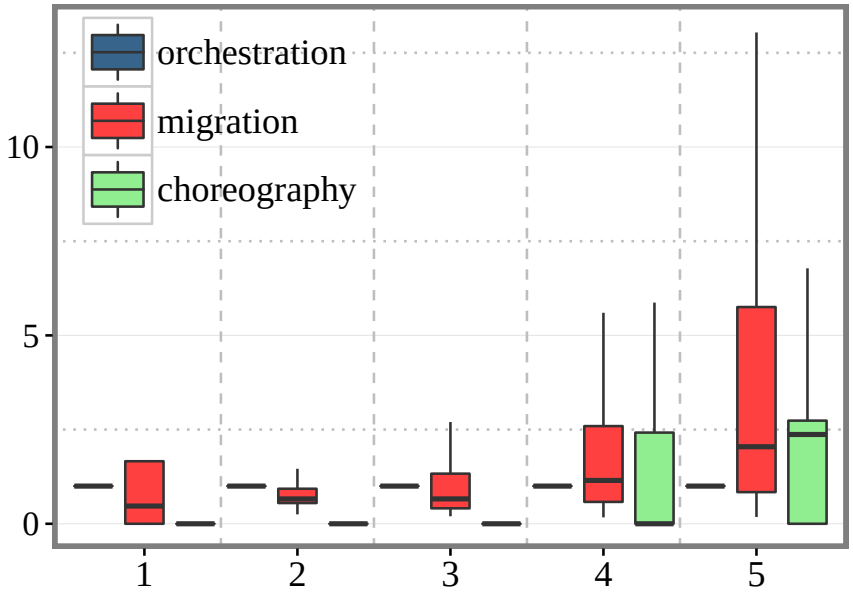
random formula generation





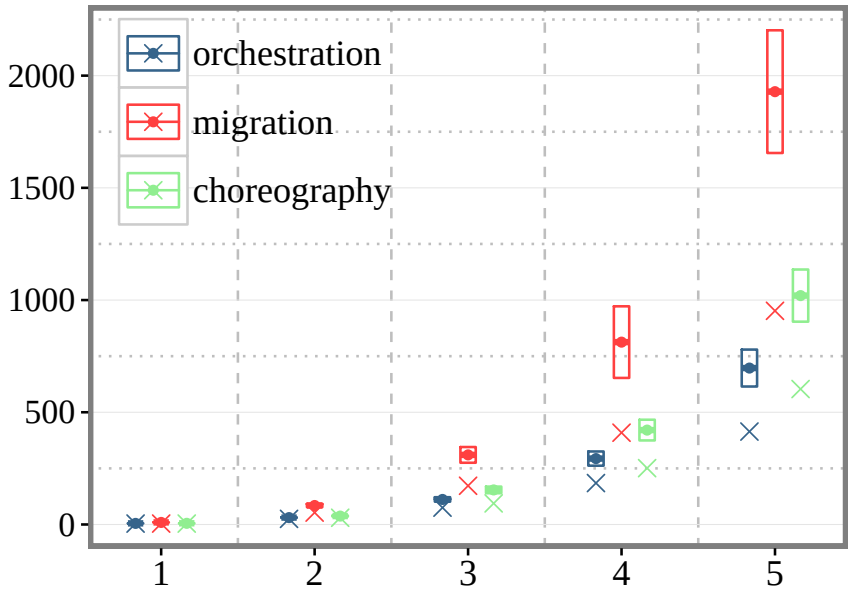
biased formula generation

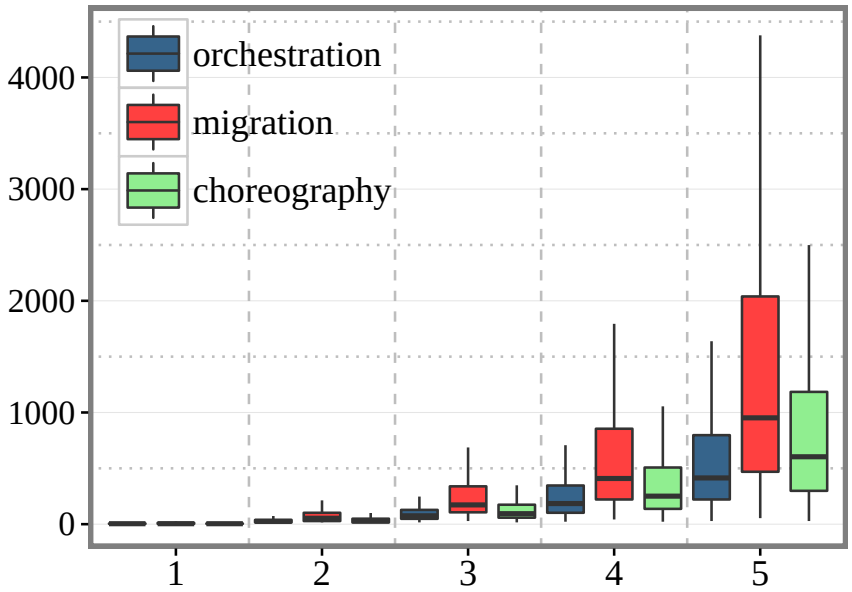




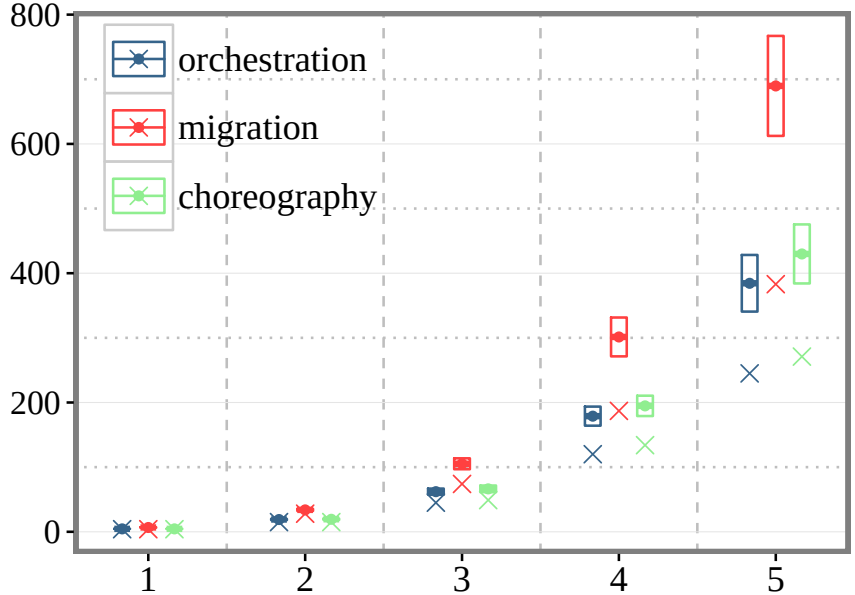
Experimental Results - number of progressions

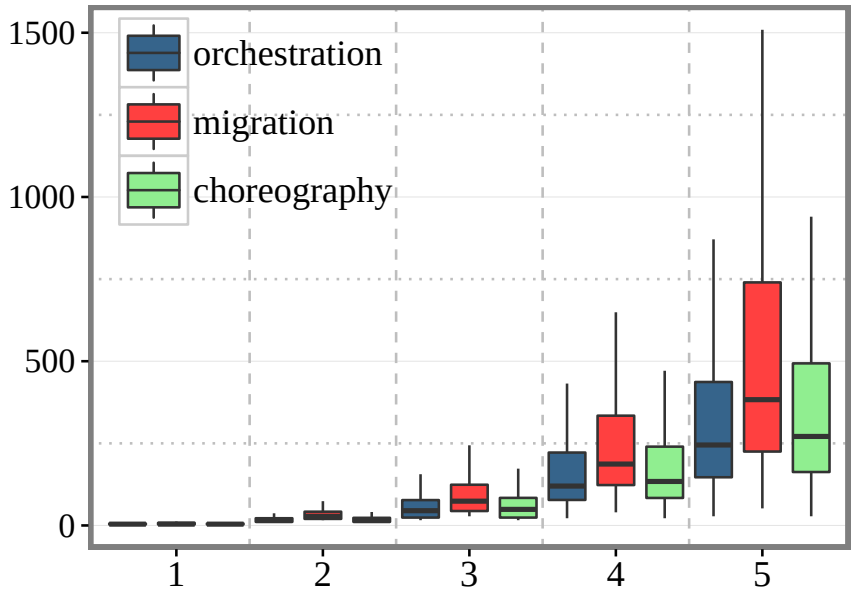
random formula generation





biased formula generation





Outline – Conclusions

Conclusions

Summary [FM12, RV14, FMSD16a, FMSD16b]

- Monitoring of (off the shelf) LTL specifications in a decentralised fashion
- No central observation point
- Keeping the *communication at a minimum* with *negligible delay*
- Validated by experimental results

Future Work

- Operational description of specifications (e.g. automata).
- Heuristics based on syntactic criteria to determine the organisation of monitor.
- Rigorous analysis of the cost of decentralised monitoring.

Please consider submitting to RV 2016 :-)!

The 16th International Conference on Runtime Verification,
September 23-30 2016, Madrid, Spain

`http://rv2016.imag.fr`

- Abstract deadline: May 20, 2016
- Paper and tutorial deadline: May 27, 2016
- COST ARVI Summer school on Runtime Verification: September 23-25, 2016
- Workshops and tutorials: September 26-27, 2016
- Conference: September 28-30, 2016

References I


Andreas Klaus Bauer and Yliès Falcone.
Decentralised LTL monitoring.
In *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, pages 85–100, 2012.


Andreas Bauer and Yliès Falcone.
Decentralised LTL monitoring.
Formal Methods in System Design, 2016.
To appear. Online version at Springer.


Christian Colombo and Yliès Falcone.
Organising LTL monitors over distributed systems with a global clock.
Formal Methods in System Design, 2016.
To appear. Online version at Springer.


Christian Colombo and Yliès Falcone.
Organising LTL monitors over distributed systems with a global clock.
In *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 140–155, 2014.

Related Work

Diagnosis of DES

- detect the occurrence of a fault after a finite number of discrete steps
- *diagnosability*: a system model is *diagnosable* if it is always the case that the occurrence of a fault can be detected after a finite number of discrete steps
- Uses the model of a system (usually contains faulty + nominal behaviours)

Decentralised observability

- Various degrees of observability depending on available memory of local observers
- Combine the local observers' states after reading some trace to a truthful verdict w.r.t. the monitored property
- Comparison with our approach:
  - No central-observation point
  - Observability is taken for granted
  - Minimisation of communication overhead

Related Work (ctd)

Monitoring

- MtTL monitoring properties of *asynchronous systems* [Sen et al.]
  - systems operating concurrently
  - partially ordered traces
  - LTL + modalities about the distributed nature of the system
  - Comparison with our approach:
    - synchronous systems
    - not restricted to safety properties
    - no collection of global behavior
- Monitoring distributed controllers [Genon et al.]
  - partially ordered traces (asynchronous systems)
  - exploration of execution interleavings
  - restricted to bad prefixes