slgo

Decentralised LTL monitoring

Andreas Bauer[1] Yliès Falcone[2]

[1]NICTA Canberra Research Lab and Australian National University
[2]Laboratoire d'Informatique de Grenoble, UJF University of Grenoble I, France

Work presented originally at FM'12

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)."

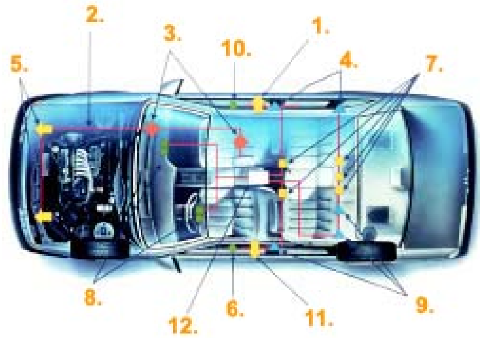
Could be formalised using LTL:

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

and then monitored as usual...

An introductory example

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

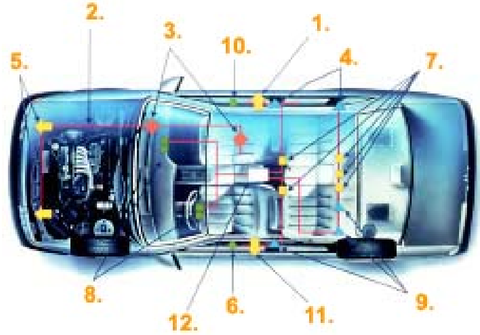


Legend:

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!

Related work (i.e., what others are doing about it)

Related work (i.e., what others are doing about it)

- Sen et al.: **Decentralized runtime analysis of multithreaded applications** (IPDPS'06)
  - Custom logic, MtTL, for specifying properties of "agents" (similar to LTL).
  - Monitoring problem: Matching of partially ordered traces against MtTL property (i.e., central collection point).
  - Restrictions: Safety properties only.

Related work (i.e., what others are doing about it)

- Sen et al.: **Decentralized runtime analysis of multithreaded applications** (IPDPS'06)
  - Custom logic, MtTL, for specifying properties of "agents" (similar to LTL).
  - Monitoring problem: Matching of partially ordered traces against MtTL property (i.e., central collection point).
  - Restrictions: Safety properties only.
- Genon et al.: **Monitoring distributed controllers** (FM'06)
  - LTL model checking of partially ordered traces (i.e., central collection point).
  - Main contribution lies in state-space reduction.

Related work (i.e., what others are doing about it)

- Sen et al.: **Decentralized runtime analysis of multithreaded applications** (IPDPS'06)
  - Custom logic, MtTL, for specifying properties of "agents" (similar to LTL).
  - Monitoring problem: Matching of partially ordered traces against MtTL property (i.e., central collection point).
  - Restrictions: Safety properties only.
- Genon et al.: **Monitoring distributed controllers** (FM'06)
  - LTL model checking of partially ordered traces (i.e., central collection point).
  - Main contribution lies in state-space reduction.
- And quite a few more works along those lines...

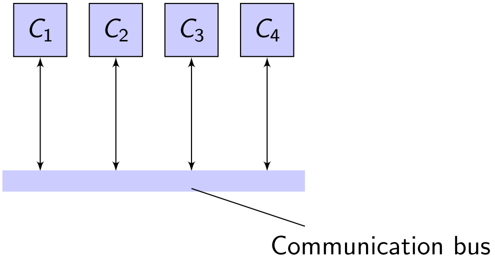
Related work (i.e., what others are doing about it)

- Sen et al.: **Decentralized runtime analysis of multithreaded applications** (IPDPS'06)
  - Custom logic, MtTL, for specifying properties of "agents" (similar to LTL).
  - Monitoring problem: Matching of partially ordered traces against MtTL property (i.e., central collection point).
  - Restrictions: Safety properties only.
- Genon et al.: **Monitoring distributed controllers** (FM'06)
  - LTL model checking of partially ordered traces (i.e., central collection point).
  - Main contribution lies in state-space reduction.
- And quite a few more works along those lines...

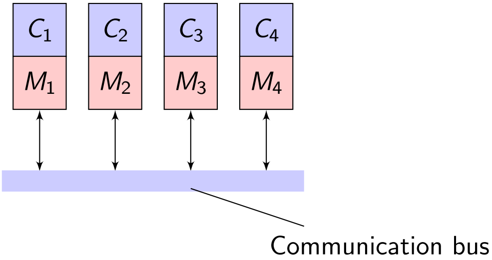
Ylies and I wanted to know...

- What happens if you can't collect a trace centrally?
- Can we monitor a system in a truly distributed manner?

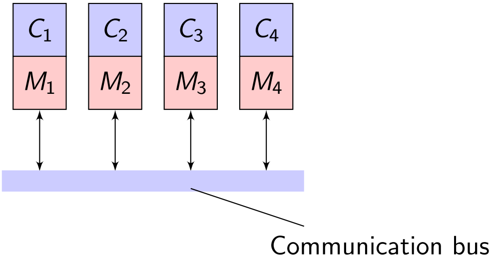
The setting (I)



The setting (I)

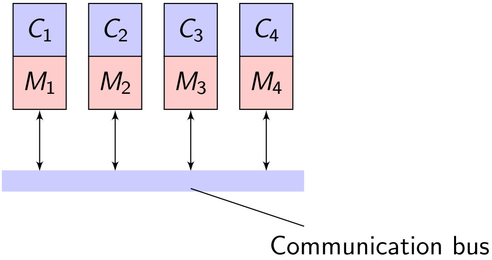


The setting (I)



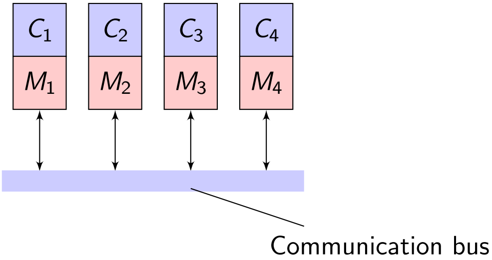
- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.

The setting (I)



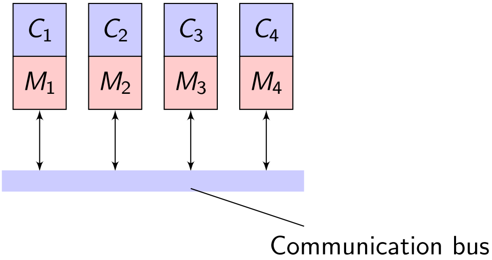
- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.
- Let $\Sigma = 2^{AP}$. Set of all system events, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_4$, where $\Sigma_j \cap \Sigma_i = \emptyset$ for all $i \neq j$.

The setting (I)



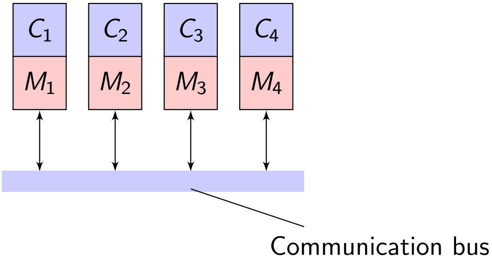
- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.
- Let $\Sigma = 2^{AP}$. Set of all system events, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_4$, where $\Sigma_j \cap \Sigma_i = \emptyset$ for all $i \neq j$.
- Let $\vec{u} = (u_1, \ldots, u_n)$ be the global trace of length t.

The setting (I)



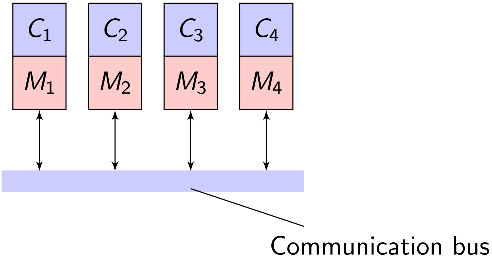
- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.
- Let $\Sigma = 2^{AP}$. Set of all system events, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_4$, where $\Sigma_j \cap \Sigma_i = \emptyset$ for all $i \neq j$.
- Let $\vec{u} = (u_1, \ldots, u_n)$ be the global trace of length t.
- Monitors, like components, communicate via the bus.

The setting (I)



- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.
- Let $\Sigma = 2^{AP}$. Set of all system events, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_4$, where $\Sigma_j \cap \Sigma_i = \emptyset$ for all $i \neq j$.
- Let $\vec{u} = (u_1, \ldots, u_n)$ be the global trace of length t.
- Monitors, like components, communicate via the bus.
- Each monitor monitors its own specification at any time t, φ_i^t. The specification changes depending on the trace and communication.

The setting (I)



- Let $\varphi \in \mathrm{LTL}(AP)$ be the global system specification to be monitored.
- Let $\Sigma = 2^{AP}$. Set of all system events, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_4$, where $\Sigma_j \cap \Sigma_i = \emptyset$ for all $i \neq j$.
- Let $\vec{u} = (u_1, \ldots, u_n)$ be the global trace of length t.
- Monitors, like components, communicate via the bus.
- Each monitor monitors its own specification at any time t, φ_i^t. The specification changes depending on the trace and communication.
- If $\varphi_i^t = \top$ (resp. *bot*) at C_i, then $\vec{u} \in \mathrm{good}(\varphi)$ (resp. $\mathrm{bad}(\varphi)$).

The setting (II)

- Bus is synchronous, i.e., at each time t a component/monitor may send (and receive) a message.
- At $t + 1$ this message is received by the recipient.
- That is, computation takes no time.
- Arguably, matches the **X**-semantics of LTL.
  - There are stutter-free variants of LTL. We do not consider this here.

A note on perfect synchrony

– *"Is perfect synchrony realistic?"*

A note on perfect synchrony

– *"Is perfect synchrony realistic?"*
– *"Not always, but many safety critical systems use it."*

A note on perfect synchrony

– *"Is perfect synchrony realistic?"*
– *"Not always, but many safety critical systems use it."*

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

FlexRay™

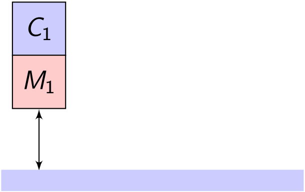
Examples: Steer-by-wire, brake-by-wire, engine management, etc.

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

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

Monitoring by progression (central case)

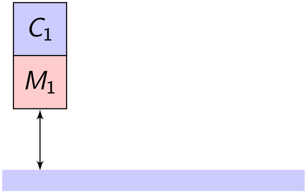
Let's assume our system looks like this:[1]



[1] Central monitoring is, but a special case of distributed monitoring.

Monitoring by progression (central case)

Let's assume our system looks like this:[1]

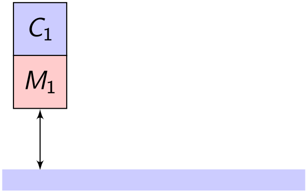


We could easily use progression to monitor a specification $\varphi \in \mathrm{LTL}$.

[1] Central monitoring is, but a special case of distributed monitoring.

Monitoring by progression (central case)

Let's assume our system looks like this:[1]



We could easily use progression to monitor a specification $\varphi \in \mathrm{LTL}$.

Definition (*Progression function* $P : \mathrm{LTL}(AP) \times 2^{AP} \rightarrow \mathrm{LTL}(AP)$)

Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}(AP)$, and $\sigma \in 2^{AP}$ be an event.

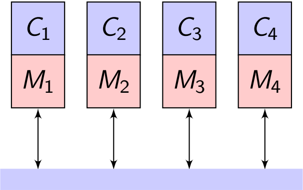
$$\begin{array}{lcl}
P(p \in AP, \sigma) & = & \top, \text{ if } p \in \sigma, \bot \text{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) & = & P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) & = & P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1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

$$\begin{array}{lcl}
P(\top, \sigma) & = & \top \\
P(\bot, \sigma) & = & \bot \\
P(\neg\varphi, \sigma) & = & \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) & = & \varphi
\end{array}$$

[1] Central monitoring is, but a special case of distributed monitoring.

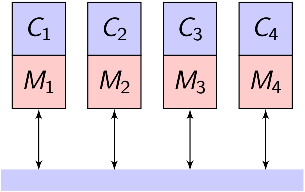
Monitoring by progression (distributed case)

But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.



Monitoring by progression (distributed case)

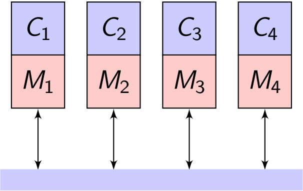
But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.



- Let's also assume that, initially, $\varphi_i^0 = \varphi$ for $i \in [1, 4]$, and that $\Sigma_i = \{p_i, \emptyset\}$.
- At time 0, let $u_1 = p_1$, $u_2 = p_2$, $u_3 = u_4 = \emptyset$.

Monitoring by progression (distributed case)

But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.

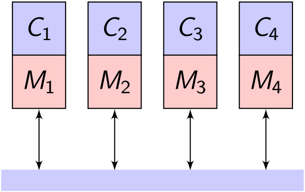


- Let's also assume that, initially, $\varphi_i^0 = \varphi$ for $i \in [1, 4]$, and that $\Sigma_i = \{p_i, \emptyset\}$.
- At time 0, let $u_1 = p_1$, $u_2 = p_2$, $u_3 = u_4 = \emptyset$.

Observe: φ is a safety property

Monitoring by progression (distributed case)

But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.



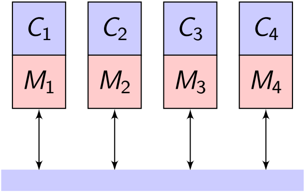
- Let's also assume that, initially, $\varphi_i^0 = \varphi$ for $i \in [1, 4]$, and that $\Sigma_i = \{p_i, \emptyset\}$.
- At time 0, let $u_1 = p_1$, $u_2 = p_2$, $u_3 = u_4 = \emptyset$.

Observe: φ is a safety property

- Is φ violated by $\vec{u}$?

Monitoring by progression (distributed case)

But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.



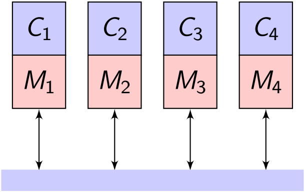
- Let's also assume that, initially, $\varphi_i^0 = \varphi$ for $i \in [1, 4]$, and that $\Sigma_i = \{p_i, \emptyset\}$.
- At time 0, let $u_1 = p_1$, $u_2 = p_2$, $u_3 = u_4 = \emptyset$.

Observe: φ is a safety property

- Is φ violated by $\vec{u}$?
- How could any of the monitors know?

Monitoring by progression (distributed case)

But we really care for distribution! Let's assume that $\varphi = \mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4)$.



- Let's also assume that, initially, $\varphi_i^0 = \varphi$ for $i \in [1, 4]$, and that $\Sigma_i = \{p_i, \emptyset\}$.
- At time 0, let $u_1 = p_1$, $u_2 = p_2$, $u_3 = u_4 = \emptyset$.

Observe: φ is a safety property

- Is φ violated by $\vec{u}$?
- How could any of the monitors know?

Monitors need to communicate outstanding obligations.

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) =$$

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) =$$
$$(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi =$$

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$
\begin{aligned}
&P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) = \\
&(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi
\end{aligned}
$$

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$\begin{aligned}
&P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) = \\
&(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi
\end{aligned}$$

If we continue with standard progression, we get $\bot$.

Although $\vec{u}$ is not a bad prefix!

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) =$$
$$(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi =$$
$$(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi$$

If we continue with standard progression, we get $\bot$.

Although $\vec{u}$ is not a bad prefix!

Rewrite into the past!

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) =$$
$$(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi =$$
$$(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi$$

If we continue with standard progression, we get $\bot$.

Although $\vec{u}$ is not a bad prefix!

Rewrite into the past!

$$(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi =$$

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$\begin{aligned}
&P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) = \\
&(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi
\end{aligned}$$

If we continue with standard progression, we get $\bot$.

Although $\vec{u}$ is not a bad prefix!

Rewrite into the past!

$$\begin{aligned}
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(\overline{\mathbf{X}}p_2 \vee \overline{\mathbf{X}}p_3 \wedge \overline{\mathbf{X}}p_4) \wedge \varphi
\end{aligned}$$

Monitoring by progression (distributed case)

Let's take a closer look at C_1/M_1:

$$
\begin{aligned}
&P_1(\mathbf{G}(p_1 \wedge p_2 \vee p_3 \wedge p_4), p_1) = \\
&(P_1(p_1, p_1) \wedge P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi
\end{aligned}
$$

If we continue with standard progression, we get $\bot$.

Although $\vec{u}$ is not a bad prefix!

Rewrite into the past!

$$
\begin{aligned}
&(P_1(p_2, p_1) \vee P_1(p_3, p_1) \wedge P_1(p_4, p_1)) \wedge \varphi = \\
&(\overline{\mathbf{X}}p_2 \vee \overline{\mathbf{X}}p_3 \wedge \overline{\mathbf{X}}p_4) \wedge \varphi
\end{aligned}
$$

(Do the same for the other monitors.)

Our first change to the progression function

Definition

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

In other words

- We need to distinguish why σ does not satisfy the proposition.
- Therefore, we add a third argument to progression function (i.e., the local alphabet)

Our second change to the progression function

Definition (Progression of past formula)

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

where Π is a projection function onto the local alphabet, and $\sigma(-m)$ the system event which occurred at time $t - m$.

Note

- Each monitor is now assumed to have a *bounded* buffer of past events!
- Since we do not allow $\overline{\mathbf{X}}$ for the specification of a global system monitoring property, our definitions will ensure that the local monitoring goals, φ_i^t, will never be of the form $\overline{\mathbf{X}}\mathbf{XX}p$, which is equivalent to a future obligation, despite the initial $\overline{\mathbf{X}}$.

Inter-monitor communication

Inter-monitor communication

What is a monitor to do with a formula $\overline{\mathbf{X}}\varphi$?

Inter-monitor communication

What is a monitor to do with a formula $\overline{\mathbf{X}}\varphi$?

Idea

Monitors send "urgent" obligations to respective co-monitors via communication bus.

Inter-monitor communication

What is a monitor to do with a formula $\overline{\mathbf{X}}\varphi$?

Idea

Monitors send "urgent" obligations to respective co-monitors via communication bus.

Definition (Urgency of formula)

Let φ be an LTL formula, and $\Upsilon : \mathrm{LTL} \to \mathbb{N}^{\geq 0}$ be an inductively defined function assigning a level of *urgency* to an LTL formula as follows.

$$
\begin{array}{rcll}
\Upsilon(\varphi) & = & \text{match } \varphi \text{ with } \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 & \to \max(\Upsilon(\varphi_1), \Upsilon(\varphi_2)) \\
 & & \mid \overline{\mathbf{X}}\varphi' & \to 1 + \Upsilon(\varphi') \\
 & & \mid \_ & \to 0.
\end{array}
$$

A formula φ is said to be *more urgent* than formula ψ, if and only if $\Upsilon(\varphi) > \Upsilon(\psi)$ holds. A formula φ where $\Upsilon(\varphi) = 0$ holds is said to be not urgent.

Communication policy

Consider M_1 again: $(\overline{\mathbf{X}}p_2 \vee \overline{\mathbf{X}}p_3 \wedge \overline{\mathbf{X}}p_4) \wedge \varphi$

- Who should M_1 send the formula to?

- Could send it to all M_2, M_3 and M_4.[2]
- But then the communication overhead for monitoring competes with the communication of the application under scrutiny. :-(

Monitor communication policy

- Send most urgent obligation first.
- If no such obligation exists, send to one monitor according to a linear order, e.g., $M_1 < \ldots < M_4$. (Order is arbitrary but fixed.)
- That is, M_1 sends the formula to M_2.

[2] In fact, the first version of this work did just that.

Handling sent obligations

Definition

If M_i, at time t, sends an obligation to another monitor, then M_i's new obligation is defined as $\varphi_i^{t+1} = \#$.

Handling sent obligations

Definition

If M_i, at time t, sends an obligation to another monitor, then M_i's new obligation is defined as $\varphi_i^{t+1} = \#$.

That is, it has nothing more to do, the other monitor now needs to check the formula. M_i may just have to progress $\#$:

Handling sent obligations

Definition

If M_i, at time t, sends an obligation to another monitor, then M_i's new obligation is defined as $\varphi_i^{t+1} = \#$.

That is, it has nothing more to do, the other monitor now needs to check the formula. M_i may just have to progress $\#$:

Definition (Our third change to progression)

$P(\#, \sigma, \mathrm{AP_i}) = \#$

Handling sent obligations

What happens when a monitor already has its own obligation?

Definition

- Let φ_j^{t+1} be M_j's obligation to be checked at time $t+1$.
- It receives from M_i, φ_i^{t+1}.
- Hence, M_j sets $\varphi_j^{t+1} = \varphi_j^{t+1} \wedge \varphi_i^{t+1}$.
- $(\# \wedge \varphi = \varphi.)$

Putting it all together

Algorithm L (*Local monitor*). Let φ be a global system specification, and $\mathcal{M} = \{M_1, \ldots, M_n\}$ be the set of component monitors. The algorithm Local Monitor, executed on each M_i, returns $\top$ (resp. $\bot$), if $\sigma \models_D \varphi_i^t$ (resp. $\sigma \not\models_D \varphi_i^t$) holds, where $\sigma \in \Sigma_i$ is the projection of an event to the observable set of actions of the respective monitor, and φ_i^t the monitor's current local obligation.

L1. [Next goal.] Let $t \in \mathbb{N}^{\geq 0}$ denote the current time step and φ_i^t be the monitor's current local obligation. If $t = 0$, then set $\varphi_i^t := \varphi$.

L2. [Receive event.] Read next σ.

L3. [Receive messages.] Let $\{\varphi_j\}_{j \in [1,n], j \neq i}$ be the set of received obligations at time t from other monitors. Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j \in [1,n], j \neq i} \varphi_j$.

L4. [Progress.] Determine $P(\varphi_i^t, \sigma, \mathrm{AP_i})$ and store the result in φ_i^{t+1}.

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

L6. [Communicate.] Let $\Psi \subseteq \mathit{sus}(\varphi_i^{t+1})$ be the set of most urgent obligations of φ_i^{t+1}. Send φ_i^{t+1} to respective monitor M_j.

L7. [Replace goal.] If in step L6 a message was sent at all, set $\varphi_i^{t+1} := \#$. Then go back to step L1.

A quick word on semantics...

We now know the behaviour/semantics of an individual monitor:

A quick word on semantics...

We now know the behaviour/semantics of an individual monitor:

Theorem

In a single component-system, decentralised progression is equivalent to "standard progression."

A quick word on semantics. . .

We now know the behaviour/semantics of an individual monitor:

Theorem

In a single component-system, decentralised progression is equivalent to "standard progression."

Proof.

Follows straight from the definitions. □

A quick word on semantics...

We now know the behaviour/semantics of an individual monitor:

Theorem

In a single component-system, decentralised progression is equivalent to "standard progression."

Proof.

Follows straight from the definitions. □

Definition

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of system components, $\varphi \in \mathrm{LTL}$ be a global goal, and $\mathcal{M} = \{M_1, \ldots, M_n\}$ be the set of component monitors. Further, let $\vec{u} = u_1(0) \cup \ldots \cup u_n(0) \cdot u_1(1) \cup \ldots \cup u_n(1) \cdots u_1(t) \cup \ldots \cup u_n(t)$ be the global behavioural trace, at time $t \in \mathbb{N}^{\geq 0}$. If for some component C_i, with $i \leq n$, containing a local obligation φ_i^t, M_i reports $P(\varphi_i^t, u_i(t), \mathrm{AP_i}) = \top$ (resp. $\bot$), then $\vec{u} \models_D \varphi = \top$ (resp. $\bot$). Otherwise, $\vec{u} \models_D \varphi = ?$.

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:				
M_A:				
M_B:				
M_C:				

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$			
M_A:				
M_B:				
M_C:				

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$			
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_B:				
M_C:				

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$			
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_C:				

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$			
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$			

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$		
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$			

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$		
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$			
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$			

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$		
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$			

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$		
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$		

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$		

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$		
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$		

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$		

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B})$ $= \#$	
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C})$ $= \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$	$\varphi_A^4 = P(\varphi_C^3 \wedge \#, \sigma, \mathrm{AP_A})$ $= \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}} b \wedge \overline{\mathbf{X}} c \vee \varphi)$
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B})$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B})$ $= \#$	
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C})$ $= \overline{\mathbf{X}} a \wedge \overline{\mathbf{X}} b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C})$ $= \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^4 = P(\varphi_C^3 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	$\varphi_B^4 = P(\varphi_A^3 \wedge \#, \sigma, \mathrm{AP_B}) = \top$
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^4 = P(\varphi_C^3 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	$\varphi_B^4 = P(\varphi_A^3 \wedge \#, \sigma, \mathrm{AP_B}) = \top$
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	$\varphi_C^4 = P(\#, \sigma, \mathrm{AP_C}) = \#$

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^4 = P(\varphi_C^3 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	$\varphi_B^4 = P(\varphi_A^3 \wedge \#, \sigma, \mathrm{AP_B}) = \top$
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	$\varphi_C^4 = P(\#, \sigma, \mathrm{AP_C}) = \#$

Thus, $\{a, b\}\{a, b, c\}\emptyset\emptyset \models_D \varphi$.

Example—the algorithm at work

Decentralised prog. of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

t:	0	1	2	3
σ:	$\{a, b\}$	$\{a, b, c\}$	$\emptyset$	$\emptyset$
M_A:	$\varphi_A^1 = P(\varphi, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 = P(\varphi_B^1 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 = P(\varphi_C^2 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^4 = P(\varphi_C^3 \wedge \#, \sigma, \mathrm{AP_A}) = \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$
M_B:	$\varphi_B^1 = P(\varphi, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 = P(\varphi_A^1 \wedge \#, \sigma, \mathrm{AP_B}) = \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 = P(\#, \sigma, \mathrm{AP_B}) = \#$	$\varphi_B^4 = P(\varphi_A^3 \wedge \#, \sigma, \mathrm{AP_B}) = \top$
M_C:	$\varphi_C^1 = P(\varphi, \sigma, \mathrm{AP_C}) = \varphi$	$\varphi_C^2 = P(\varphi, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 = P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, \mathrm{AP_C}) = \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	$\varphi_C^4 = P(\#, \sigma, \mathrm{AP_C}) = \#$

Thus, $\{a, b\}\{a, b, c\}\emptyset\emptyset \models_D \varphi$.

(Well, in fact, we'd have to show that our definition of semantics implies this result. But we have: it is a ca. 10 page proof in the paper.)

How much does a monitor need to remember?

Theorem

Let, for any $p \in AP$, $\overline{\mathbf{X}}^m p$ be a local obligation obtained by Algorithm L executed on some monitor $M_i \in \mathcal{M}$. At any time $t \in \mathbb{N}^{\geq 0}$, $m \leq min(|\mathcal{M}|, t+1)$.

How much does a monitor need to remember?

Theorem

Let, for any $p \in AP$, $\overline{\mathbf{X}}^m p$ be a local obligation obtained by Algorithm L executed on some monitor $M_i \in \mathcal{M}$. At any time $t \in \mathbb{N}^{\geq 0}$, $m \leq min(|\mathcal{M}|, t+1)$.

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

How much does a monitor need to remember?

Theorem

Let, for any $p \in AP$, $\overline{\mathbf{X}}^m p$ be a local obligation obtained by Algorithm L executed on some monitor $M_i \in \mathcal{M}$. At any time $t \in \mathbb{N}^{\geq 0}$, $m \leq min(|\mathcal{M}|, t+1)$.

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

However

Unless, we rule this out, there could be an infinite delay not due to communication:

- $true\mathbf{U}(\mathbf{G}b \vee \mathbf{F}\neg b)$
- As long as we don't see $\neg b$, the monitor doesn't know it's a tautology!

How much does a monitor need to remember?

Theorem

Let, for any $p \in AP$, $\overline{\mathbf{X}}^m p$ be a local obligation obtained by Algorithm L executed on some monitor $M_i \in \mathcal{M}$. At any time $t \in \mathbb{N}^{\geq 0}$, $m \leq min(|\mathcal{M}|, t+1)$.

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

However

Unless, we rule this out, there could be an infinite delay not due to communication:

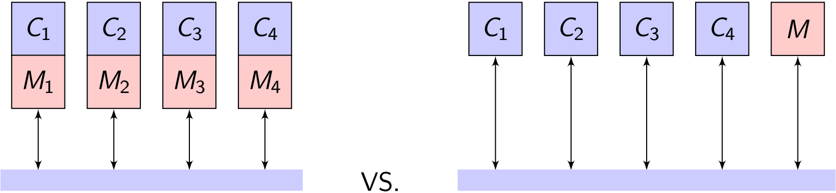
- $true\mathbf{U}(\mathbf{G}b \vee \mathbf{F}\neg b)$
- As long as we don't see $\neg b$, the monitor doesn't know it's a tautology!

Corollary

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

Implementation of the approach

We have implemented our approach (DecentMon) and compared it empirically against a centralised approach (right picture):



Evaluation—random formulae

- Three monitors, A, B, C, each see actions a, b, c, respectively.
- DecentMon generates 1000 random LTL formulae, and monitors random traces:

	centralised		decentralised		*diff. ratio*	
$\lvert\varphi\rvert$	|trace|	#msg.	|trace|	#msg.	|trace|	#msg.
1	1.369	4.107	1.634	0.982	1.1935	0.2391
2	2.095	6.285	2.461	1.647	1.1747	0.262
3	3.518	10.554	4.011	2.749	1.1401	0.2604
4	5.889	17.667	6.4	4.61	1.0867	0.2609
5	9.375	28.125	9.935	7.879	1.0597	0.2801
6	11.808	35.424	12.366	9.912	1.0472	0.2798

- First column: all formulae of size $\lvert n\rvert$.
- $\lvert trace\rvert$ column: length of trace until verdict was reached.
- $\#msg$. column: how many messages were exchanged.

Evaluation—a note on heuristics

A quick word on formula length: Normally, $|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 7$.

Evaluation—a note on heuristics

A quick word on formula length: Normally, $|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 7$.

But not representative for effort to monitor in distributed manner! E.g., $(a \wedge \ldots \wedge z)$ is of size 25, yet can be monitored almost instantly.

Evaluation—a note on heuristics

A quick word on formula length: Normally, $|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 7$.

But not representative for effort to monitor in distributed manner! E.g., $(a \wedge \ldots \wedge z)$ is of size 25, yet can be monitored almost instantly.

Here: formula size = number of temp. operators

$|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 2$

Evaluation—a note on heuristics

A quick word on formula length: Normally, $|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 7$.

But not representative for effort to monitor in distributed manner! E.g., $(a \wedge \ldots \wedge z)$ is of size 25, yet can be monitored almost instantly.

Here: formula size = number of temp. operators

$|\mathbf{G}(a \wedge b) \vee \mathbf{F}c| = 2$

Formula length = heuristic for communication effort

Evaluation—spec patterns

- Specification Patterns (Dwyer et al.) describe frequently occurring requirements in software specification (absence, existence, etc.)
- We generated 1000 LTL formulae, corresponding to each such requirement.

	centralised		decentralised		*diff. ratio*	
pattern	|trace|	#msg.	|trace|	#msg.	|trace|	#msg.
absence	156.17	468.51	156.72	37.94	1.0035	0.0809
existence	189.90	569.72	190.42	44.41	1.0027	0.0779
bounded existence	171.72	515.16	172.30	68.72	1.0033	0.1334
universal	97.03	291.09	97.66	11.05	1.0065	0.0379
precedence	224.11	672.33	224.72	53.703	1.0027	0.0798
response	636.28	1,908.86	636.54	360.33	1.0004	0.1887
precedence chain	200.23	600.69	200.76	62.08	1.0026	0.1033
response chain	581.20	1,743.60	581.54	377.64	1.0005	0.2165
constrained chain	409.12	1,227.35	409.62	222.84	1.0012	0.1815

Thank you!

That's Canberra:

View onto Lake Burley Griffin from Mount Ainslie (in winter).