Failure-aware Runtime Verification of Distributed Systems

David Basin[1] Felix Klaedtke[2]
Eugen Zălinescu[3]

[1] Institute of Information Security
ETH Zürich

[2] NEC Europe Ltd.

[3] TUM, I02

TUM, Garching
March 9th, 2016

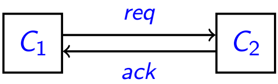
Runtime Verification

- check *at runtime* whether a system's behaviour satisfies a property
- lightweight alternative to model checking
  (e.g. when the model is not available or it is too large)
- verify correctness of the actual observed system behavior
- wide range of approaches and applications

Runtime Verification

- check *at runtime* whether a system's behaviour satisfies a property
- lightweight alternative to model checking
  (e.g. when the model is not available or it is too large)
- verify correctness of the actual observed system behavior
- wide range of approaches and applications

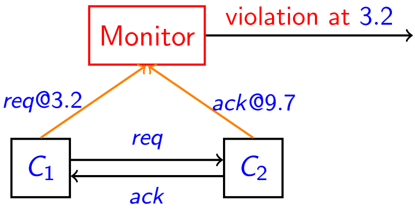
- Example: every request must be acknowledged in 5 milliseconds



Runtime Verification

- check *at runtime* whether a system's behaviour satisfies a property
- lightweight alternative to model checking
  (e.g. when the model is not available or it is too large)
- verify correctness of the actual observed system behavior
- wide range of approaches and applications

- Example: every request must be acknowledged in 5 milliseconds



Runtime Verification of Distributed Systems

In a **distributed system**

- components might crash
- components communicate asynchronously
- network failures may occur

$\leadsto$ message delays, out-of-order message receipt, message loss

Current approaches are limited:

- no real-time constraints (e.g. deadlines are met)
- no message loss
- no out-of-order messages
  - naive solution: buffer messages $\leadsto$ delayed reports

Overview

Contribution: a monitoring approach for distributed systems

- It accounts for message loss and delays.
- Observations can arrive at the monitor in any order.
- It supports real-time temporal properties.
- Monitoring itself can be distributed.

Ingredients

- the real-time logic MTL (Metric Temporal Logic)
- three valued MTL semantics
- AND/OR graph-like data structure

Time Model

- Components use their clocks to timestamp observations.
- Timestamps *totally* order the observations.
  - This is in contrast to a time-free model.
- **Note**: monitor reports are valid as long as timestamps are accurate.
  - Real clocks are imprecise.
  - In practice, timestamps are "accurate enough".
    (e.g. NTP maintains synchronization over LANs within 1 millisecond)
  - Imprecision can be taken into account in the formalization.

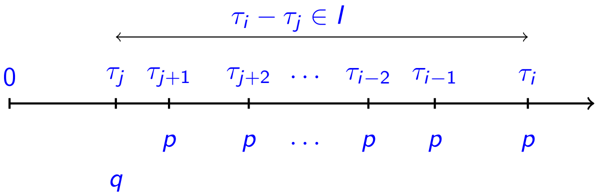
Metric Temporal Logic

- **Syntax**: $p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, \mathsf{S}_I \, \varphi \mid \varphi \, \mathsf{U}_I \, \varphi$
  where $p \in P$ and I is a non-empty interval over $\mathbb{Q}_+$

Metric Temporal Logic

- **Syntax**: $p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, \mathsf{S}_I \, \varphi \mid \varphi \, \mathsf{U}_I \, \varphi$
  where $p \in P$ and I is a non-empty interval over $\mathbb{Q}_+$

- **Semantics**:
  - *Models*: timed words $w = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$
    where $\sigma_i \in \{\mathsf{t}, \mathsf{f}\}^P$ and $\tau_i \in \mathbb{Q}_+$
  - $[\![w, i \models \varphi]\!] \in \{\mathsf{t}, \mathsf{f}\}$

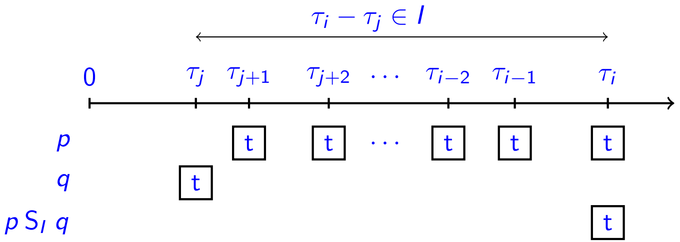


$\varphi = p \, \mathsf{S}_I \, q$

Metric Temporal Logic

- **Syntax**: $p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, \mathsf{S}_I \, \varphi \mid \varphi \, \mathsf{U}_I \, \varphi$
  where $p \in P$ and I is a non-empty interval over $\mathbb{Q}_+$

- **Semantics**:
  * *Models*: timed words $w = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$
    where $\sigma_i \in \{\mathrm{t}, \mathrm{f}\}^P$ and $\tau_i \in \mathbb{Q}_+$
  * $[\![w, i \models \varphi]\!] \in \{\mathrm{t}, \mathrm{f}\}$



Three-valued MTL

- Additional truth value $\perp$ ('unknown') represents a knowledge gap.
  * truth tables given by strong Kleene logic

$\neg$	
t	f
f	t
$\perp$	$\perp$

$\vee$	t	f	$\perp$
t	t	t	t
f	t	f	$\perp$
$\perp$	t	$\perp$	$\perp$

$\wedge$	t	f	$\perp$
t	t	f	$\perp$
f	f	f	f
$\perp$	$\perp$	f	$\perp$

Three-valued MTL

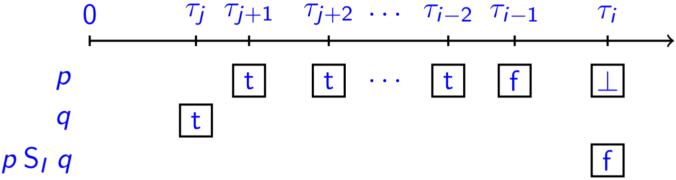
- Additional truth value $\perp$ ('unknown') represents a knowledge gap.
  - ∗ truth tables given by strong Kleene logic

$\neg$	
t	f
f	t
$\perp$	$\perp$

$\vee$	t	f	$\perp$
t	t	t	t
f	t	f	$\perp$
$\perp$	t	$\perp$	$\perp$

$\wedge$	t	f	$\perp$
t	t	f	$\perp$
f	f	f	f
$\perp$	$\perp$	f	$\perp$

- Satisfaction relation lifted to $\{\perp, \mathrm{f}, \mathrm{t}\}$
  - ∗ letters (σ, τ) with $\sigma \in \{\perp, \mathrm{f}, \mathrm{t}\}^P$
  - ∗ $[\![w, i \models \varphi]\!] \in \{\perp, \mathrm{f}, \mathrm{t}\}$, e.g. $[\![w, i \models \neg\varphi]\!] = \neg[\![w, i \models \varphi]\!]$



Monitoring Architecture

- The target system consists of one or more components.
- For monitoring, additional components are added to the system.
  - * Each monitor is responsible for some subformula.
  - * The monitor decomposition is system and application specific.

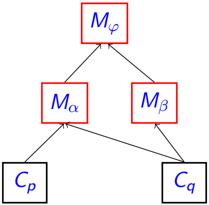
- Components communicate their observations over channels.
  - * Observations encoded and transmitted by messages: e.g. $\mathrm{report}(q, \mathrm{f}, 2.3)$

Example

$\varphi = \alpha \land \beta$

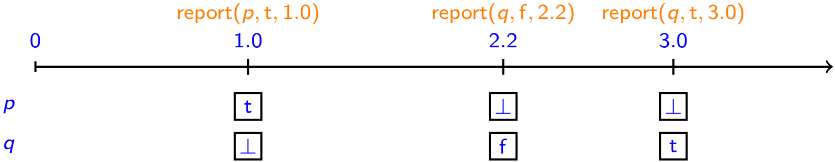
$\alpha = (\neg p \; \mathsf{S} \; q)$

$\beta = \Diamond_{[1,2]} \, q$



Requirements on Monitors

- System behavior captured by
  - the set O of messages sent by system components, or
  - a timed word w_O



Requirements on Monitors

- System behavior captured by
  - the set O of messages sent by system components, or
  - a timed word w_O
- A monitoring algorithm M:
  - iteratively receives messages $m_i \in O$,
  - at each iteration, it outputs a set of verdicts $M(m_i) \subseteq \mathbb{Q}_+ \times \{\mathrm{t}, \mathrm{f}\}$

Requirements on Monitors

- System behavior captured by
  - the set O of messages sent by system components, or
  - a timed word w_O
- A monitoring algorithm M:
  - iteratively receives messages $m_i \in O$,
  - at each iteration, it outputs a set of verdicts $M(m_i) \subseteq \mathbb{Q}_+ \times \{\mathsf{t}, \mathsf{f}\}$

Goal. Given formula φ and set O of sent messages, design M_φ:

- *Soundness*:

  If $(\tau, b) \in M_\varphi(m_i)$, then $[\![w_O, \tau \models \varphi]\!] = b$

Requirements on Monitors

- System behavior captured by
  - the set O of messages sent by system components, or
  - a timed word w_O
- A monitoring algorithm M:
  - iteratively receives messages $m_i \in O$,
  - at each iteration, it outputs a set of verdicts $M(m_i) \subseteq \mathbb{Q}_+ \times \{\mathrm{t}, \mathrm{f}\}$

Goal. Given formula φ and set O of sent messages, design M_φ:

- *Soundness*:

  If $(\tau, b) \in M_\varphi(m_i)$, then $[\![w_O, \tau \models \varphi]\!] = b$
- *Completeness*:

  If $[\![w_O, \tau \models \varphi]\!] = b \in \{\mathrm{f}, \mathrm{t}\}$,

  then exists $i \in \mathbb{N}$ such that $(\tau, b) \in M_\varphi(m_i)$

  when
  - all sent messages are received by the monitor
  - all temporal future connective in φ are bounded

(In)complete Intervals

- For completeness, the monitor should also know about the non-existence of observations (or time points) in an interval
  - Artifact of the MTL's point-wise semantics
  - Example: $[\![w, i \models \blacklozenge_{[1,2]}\, \mathsf{t}]\!]$ may be f

(In)complete Intervals

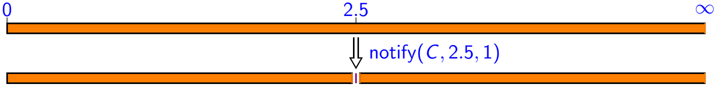
- For completeness, the monitor should also know about the non-existence of observations (or time points) in an interval
  - Artifact of the MTL's point-wise semantics
  - Example: $[\![w, i \models \blacklozenge_{[1,2]} \mathsf{t}]\!]$ may be f

- J is complete if M_φ knows of all letters (σ, τ) of w_O with $\tau \in J$
  - New message type: $\mathsf{notify}(C, \tau, s)$ — C has made an observation at τ
    - s is the sequence number of this observation
    - J is complete iff M_φ has received all $\mathsf{notify}(\_, \tau, \_)$ with $\tau \in J$
  - **Example** (2 components: C and D)

(In)complete Intervals

- For completeness, the monitor should also know about the non-existence of observations (or time points) in an interval
  - Artifact of the MTL's point-wise semantics
  - Example: $[\![w, i \models \blacklozenge_{[1,2]}\, \mathsf{t}]\!]$ may be f

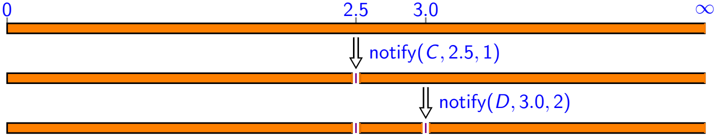
- J is complete if M_φ knows of all letters (σ, τ) of w_O with $\tau \in J$
  - New message type: $\mathsf{notify}(C, \tau, s)$ — C has made an observation at τ
    - s is the sequence number of this observation
    - J is complete iff M_φ has received all $\mathsf{notify}(\_, \tau, \_)$ with $\tau \in J$
  - **Example** (2 components: C and D)



(In)complete Intervals

- For completeness, the monitor should also know about the non-existence of observations (or time points) in an interval
  - Artifact of the MTL's point-wise semantics
  - Example: $[\![w, i \models \blacklozenge_{[1,2]}\, \mathsf{t}]\!]$ may be f

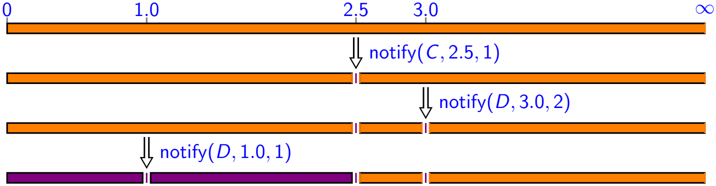
- J is complete if M_φ knows of all letters (σ, τ) of w_O with $\tau \in J$
  - New message type: notify(C, τ, s) — C has made an observation at τ
    - s is the sequence number of this observation
    - J is complete iff M_φ has received all notify$(_, \tau, _)$ with $\tau \in J$
  - **Example** (2 components: C and D)



(In)complete Intervals

- For completeness, the monitor should also know about the non-existence of observations (or time points) in an interval
  - Artifact of the MTL's point-wise semantics
  - Example: $[\![w, i \models \blacklozenge_{[1,2]}\, \mathsf{t}]\!]$ may be f

- J is complete if M_φ knows of all letters (σ, τ) of w_O with $\tau \in J$
  - New message type: $\mathsf{notify}(C, \tau, s)$ — C has made an observation at τ
    - s is the sequence number of this observation
    - J is complete iff M_φ has received all $\mathsf{notify}(\_, \tau, \_)$ with $\tau \in J$
  - **Example** (2 components: C and D)



i-words

- An *i-word* is a sequence $(\sigma_0, J_0)(\sigma_1, J_1)\ldots(\sigma_n, J_n)$ where
  * the sequence (J_i) is increasing and the intervals are non-overlapping
  * intuition on i-words
    - if $|J_i| > 1$ then $\sigma_i = \sigma_\perp$, where $\sigma_\perp(p) := \perp$ for each $p \in P$

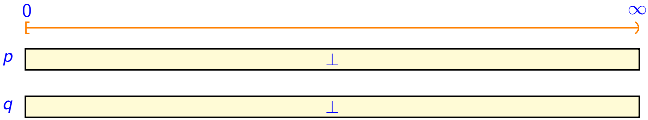
- Let $TW(u)$ be all the timed-words that match with the i-word u.
  * w matches u if for any letter (σ, τ) in w there is a letter (σ', J) in u such that $\tau \in J$ and $\sigma' \leq \sigma$.

- If $[\![u, J \models \varphi]\!] = b \in \{\mathrm{f}, \mathrm{t}\}$,
  then $[\![w, \tau \models \varphi]\!] = b$ for any $w \in TW(u)$ and any τ with $\tau \in J$.

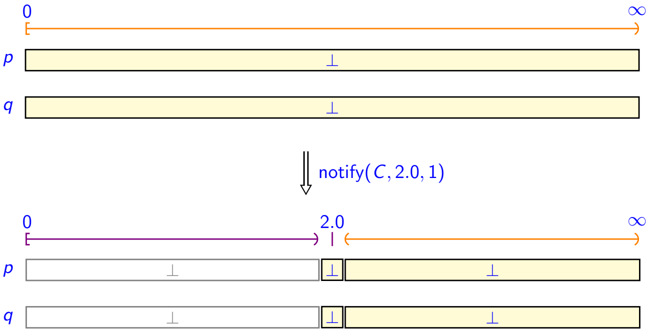
Refinement of i-words

- Given the received messages (m_i), we built the i-words (u_i)
  - $u_0 := (\sigma_\perp, [0, \infty))$
  - u_{i+1} is built from u_i based on m_i
  - If $m_i = \text{notify}(C, \tau, s)$
    - Let j be the index of the letter such that $\tau \in J_j$
    - Replace $(\sigma_\perp, J_j)$ by $(\sigma_\perp, J_j \cap [0, \tau))\,(\sigma_\perp, \{\tau\})\,(\sigma_\perp, J_j \cap (\tau, \infty))$
    - Infer the new complete intervals and delete the corresponding letters
  - If $m_i = \text{report}(p, b, \tau)$
    - Let j be the index of the letter such that $J_j = \{\tau\}$
    - Replace letter $(\sigma_j, \{\tau\})$ by $(\sigma_j[p \mapsto b], \{\tau\})$

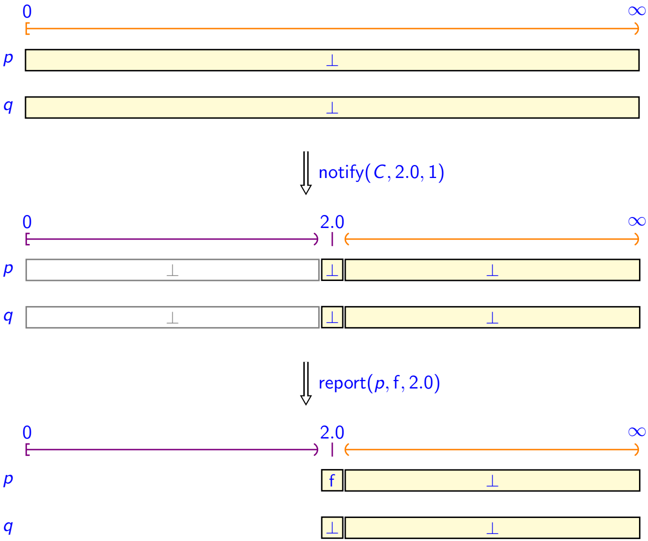
Example



Example



Example



Overview of Monitoring Algorithm

- $\{w_O\} \subseteq \cdots \subseteq TW(u_{\ell+1}) \subseteq TW(u_\ell) \subseteq \cdots \subseteq TW(u_0)$
- If $[\![u_\ell, J \models \varphi]\!] = b \in \{\mathsf{f}, \mathsf{t}\}$,
  then $[\![w_O, \tau \models \varphi]\!] = b$ for any τ with $\tau \in J$.

Overview of Monitoring Algorithm

- $\{w_O\} \subseteq \cdots \subseteq TW(u_{\ell+1}) \subseteq TW(u_\ell) \subseteq \cdots \subseteq TW(u_0)$
- If $[\![u_\ell, J \models \varphi]\!] = b \in \{\mathrm{f}, \mathrm{t}\}$,
  then $[\![w_O, \tau \models \varphi]\!] = b$ for any τ with $\tau \in J$.

- The monitor computes $[\![u_\ell, J \models \varphi]\!]$, for all letters (σ, J) in u_ℓ, by keeping state between iterations ℓ:
  - * It stores the unrolling of $[\![u_\ell, J \models \varphi]\!]$ as a graph-like data structure
  - * Common subformulas are shared
    - There is a node (ψ, J) for each subformula ψ, and each letter (σ, J) such that $[\![u_\ell, J \models \varphi]\!] = \perp$
  - * With each message m_ℓ, it updates the state:
    - If $m_\ell = \mathrm{notify}(C, \tau, s)$, then it replaces nodes (ψ, J) by nodes $(\psi, J \cap [0, \tau))$, $(\psi, \{\tau\})$, $(\psi, J \cap (0, \infty))$
    - If $m_\ell = \mathrm{report}(p, b, \tau)$, then it replaces $[\![u_\ell, \{\tau\} \models p]\!]$ by b and simplifies the formulas

Monitor State

- Nodes (ψ, J) are linked according to the formula defining the semantics of ψ

  $[\![u_\ell, J \models \alpha \vee \beta]\!] := [\![u_\ell, J \models \alpha]\!] \vee [\![u_\ell, J \models \beta]\!]$

  $[\![(\alpha \vee \beta, J)]\!] = [\![(\alpha, J)]\!] \vee [\![(\beta, J)]\!]$

Monitor State

- Nodes (ψ, J) are linked according to the formula defining the semantics of ψ

  $[\![u_\ell, J \models \alpha \vee \beta]\!] := [\![u_\ell, J \models \alpha]\!] \vee [\![u_\ell, J \models \beta]\!]$

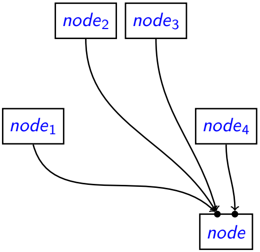
  $[\![(\alpha \vee \beta, J)]\!] = [\![(\alpha, J)]\!] \vee [\![(\beta, J)]\!]$

Terminology

- Nodes have guards.
- Guards have preconditions.

Intuition (such formulas are in DNF)

$[\![node]\!] = \bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} [\![node']\!]$



Monitor State

- Nodes (ψ, J) are linked according to the formula defining the semantics of ψ

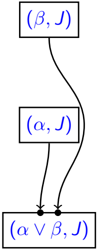
  $$[\![u_\ell, J \models \alpha \vee \beta]\!] := [\![u_\ell, J \models \alpha]\!] \vee [\![u_\ell, J \models \beta]\!]$$

  $$[\![(\alpha \vee \beta, J)]\!] = [\![(\alpha, J)]\!] \vee [\![(\beta, J)]\!]$$

  Terminology

  * Nodes have guards.
  * Guards have preconditions.

  Intuition (such formulas are in DNF)
  $[\![node]\!] = \bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} [\![node']\!]$



Monitor State

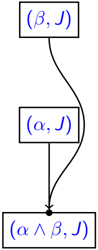
- Nodes (ψ, J) are linked according to the formula defining the semantics of ψ

  $$\llbracket u_\ell, J \models \alpha \wedge \beta \rrbracket := \llbracket u_\ell, J \models \alpha \rrbracket \wedge \llbracket u_\ell, J \models \beta \rrbracket$$

  $$\llbracket (\alpha \wedge \beta, J) \rrbracket = \llbracket (\alpha, J) \rrbracket \wedge \llbracket (\beta, J) \rrbracket$$

  Terminology
  - Nodes have guards.
  - Guards have preconditions.

  Intuition (such formulas are in DNF)
  $\llbracket node \rrbracket = \bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} \llbracket node' \rrbracket$



Monitor State

- Nodes (ψ, J) are linked according to the formula defining the semantics of ψ

$$[\![u_\ell, J \models \blacklozenge_I \beta]\!] := \bigvee_{K:(J-K)\cap I \neq \emptyset} [\![u_\ell, K \models \beta]\!]$$

$$[\![(\blacklozenge_I \beta, J)]\!] = \bigvee_{K:(J-K)\cap I \neq \emptyset} [\![(\beta, K)]\!]$$

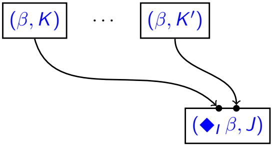
Terminology

- Nodes have guards.
- Guards have preconditions.

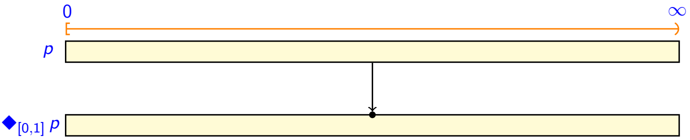
Intuition (such formulas are in DNF)

$$[\![node]\!] = \bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} [\![node']\!]$$

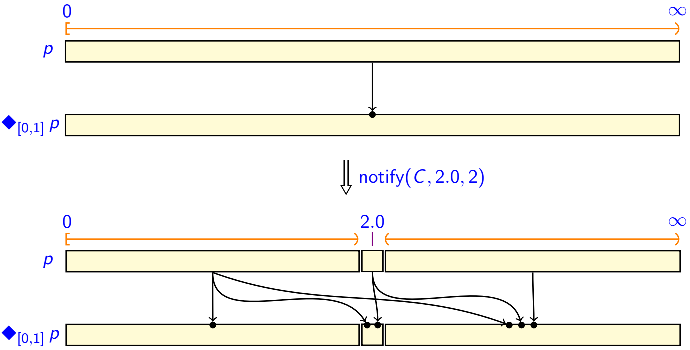
$(\beta, K) \rightsquigarrow (\blacklozenge_I \beta, J)$ iff there are $\tau \in J$ and $\kappa \in K$ such that $\tau - \kappa \in I$



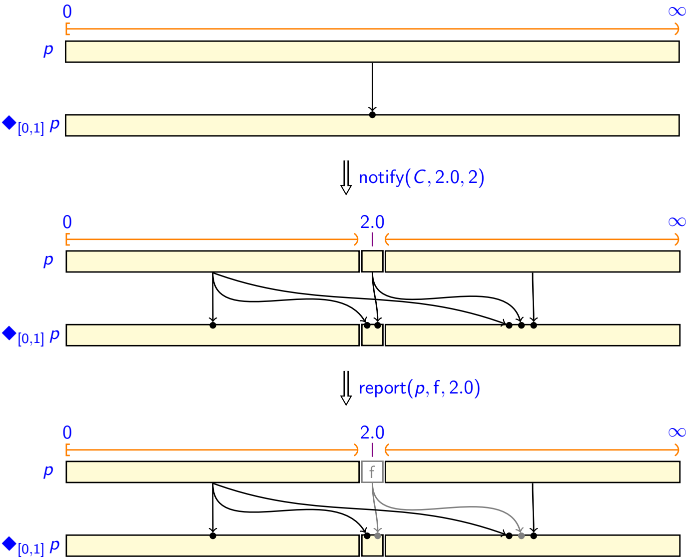
Example



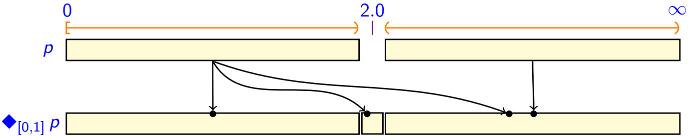
Example



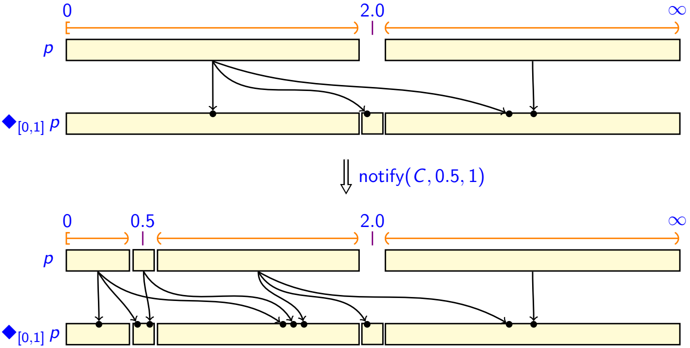
Example



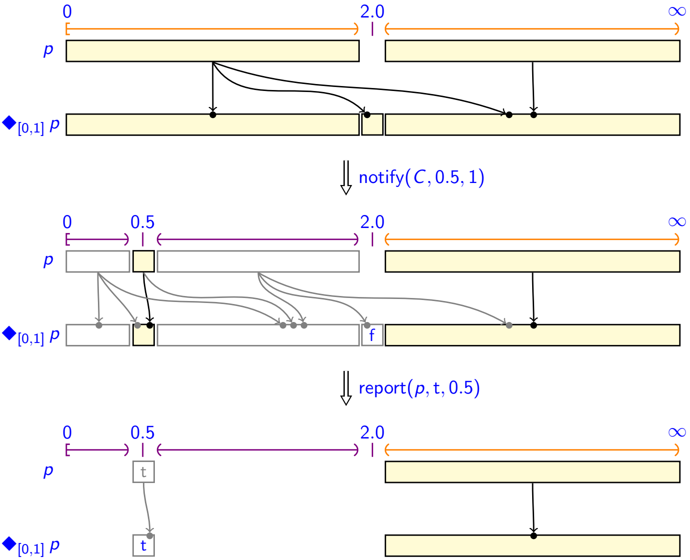
Example (cont.)



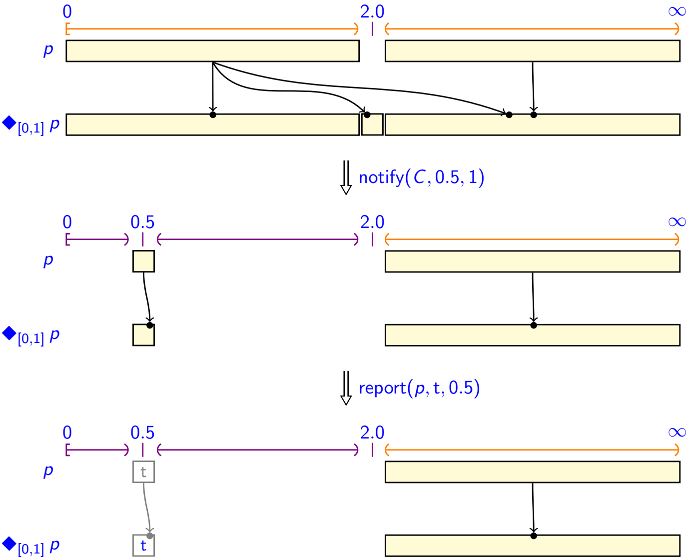
Example (cont.)



Example (cont.)

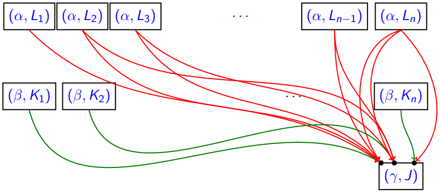


Example (cont.)



The Since and Until Cases

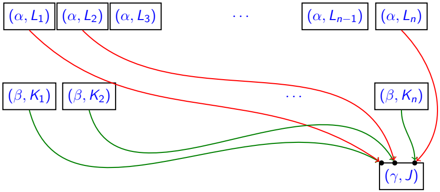
- From $\bigvee_{j \in \{\ell \in \mathbb{N} \mid \tau_i - \tau_\ell \in I\}} \left([\![w, j \models \beta]\!] \wedge \bigwedge_{j < k \leq i} [\![w, k \models \alpha]\!] \right)$:



$(\beta, K) \rightsquigarrow (\alpha \, \mathsf{S}_I \, \beta, J)$ iff there are $\tau \in J,\ \kappa \in K$ such that $\tau - \kappa \in I$

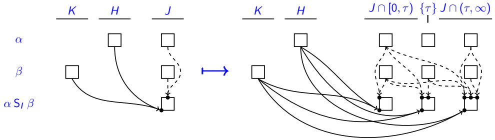
The Since and Until Cases

- From $\bigvee_{j \in \{\ell \in \mathbb{N} \mid \tau_i - \tau_\ell \in I\}} \left([\![w, j \models \beta]\!] \wedge \bigwedge_{j<k\leq i} [\![w, k \models \alpha]\!] \right)$:



$(\beta, K) \rightsquigarrow (\alpha \, \mathsf{S}_I \, \beta, J)$ iff there are $\tau \in J$, $\kappa \in K$ such that $\tau - \kappa \in I$

- **Key optimization**: only one link $(\alpha, \_) \rightsquigarrow (\alpha \, \mathsf{S}_I \, \beta, J)$ for each $(\beta, \_)$
  * Graph update and propagation become tricky



Conclusions

Summary

- distributed monitoring approach for distributed systems
  - accounts for failures and out-of-order message deliveries
  - with soundness and completeness guarantees

Current and future work

- implementation of the algorithm
- perform a case study
- extension to parametric properties (in half-order MTL)

$$\downarrow_r . \, req(r) \to \Diamond \, ack_{[0,5]}(r)$$

Thank you!

i-words

- An *i-word* is a sequence $(\sigma_0, J_0, o_0)(\sigma_1, J_1, o_1)\ldots(\sigma_n, J_n, o_n)$ where
  - ∗ the sequence (J_i) is "increasing" and partitions $[0, \infty)$
  - ∗ $o_i \in \{\bot, \mathrm{f}, \mathrm{t}\}$ represents whether there are observations in J_i
    - if $|J_i| = 1$ then $o_i = \mathrm{t}$
    - if $|J_i| > 1$ then $\sigma_i = \sigma_\bot$, where $\sigma_\bot(p) := \bot$ for each $p \in P$

- Lifting the MTL semantics to i-words
  - ∗ on timed-words:

$$[\![w, i \models \alpha \,\mathsf{S}_I\, \beta]\!] := \bigvee_{j \in \{\ell \in \mathbb{N} \mid \ell \leq i, \tau_i - \tau_\ell \in I\}} \left([\![w, j \models \beta]\!] \wedge \bigwedge_{j<k\leq i} [\![w, k \models \alpha]\!]\right)$$

or

$$[\![w, i \models \alpha \,\mathsf{S}_I\, \beta]\!] := \bigvee_{j\leq i} \left(tc(i,j) \wedge [\![w, j \models \beta]\!] \wedge \bigwedge_{j<k\leq i} [\![w, k \models \alpha]\!]\right)$$

where

$$tc(i,j) := \begin{cases} \mathrm{t} & \text{if } \tau_i - \tau_j \in I \\ \mathrm{f} & \text{otherwise} \end{cases}$$

  - ∗ on i-words:

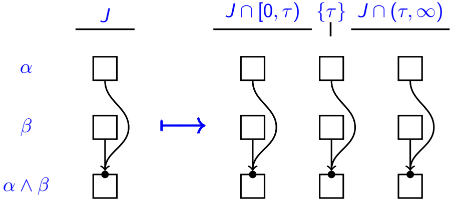
$$[\![u, i \models \alpha \,\mathsf{S}_I\, \beta]\!] := \bigvee_{j\leq i} \left(tci(i,j) \wedge o_j \wedge [\![u, j \models \beta]\!] \wedge \bigwedge_{j<k\leq i} [\![u, k \models \alpha]\!]\right)$$

where

$$tci(i,j) := \begin{cases} \mathrm{t} & \text{if } \tau - \tau' \in I, \text{ for all } \tau \in J_i \text{ and } \tau' \in J_j \\ \mathrm{f} & \text{if } \tau - \tau' \notin I, \text{ for all } \tau \in J_i \text{ and } \tau' \in J_j \\ \bot & \text{otherwise} \end{cases}$$

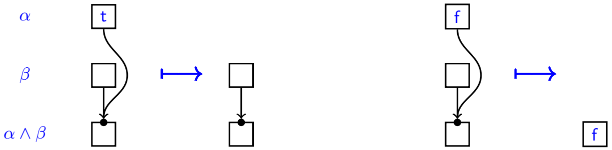
Algorithm Overview

- Initially all nodes have the form $(\alpha, [0, \infty))$
- For each received message $\mathsf{report}(p, b, \tau)$, the monitor
  - splits all nodes (α, J) with $\tau \in J$
    into 3 new nodes: $(\alpha, J \cap [0, \tau))$, $(\alpha, \{\tau\})$, $(\alpha, J \cap (\tau, \infty))$



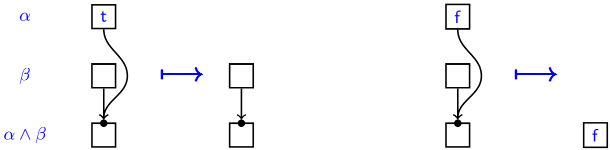
Algorithm Overview

- Initially all nodes have the form $(\alpha, [0, \infty))$
- For each received message $\mathsf{report}(p, b, \tau)$, the monitor
  - ∗ splits all nodes (α, J) with $\tau \in J$
    into 3 new nodes: $(\alpha, J \cap [0, \tau))$, $(\alpha, \{\tau\})$, $(\alpha, J \cap (\tau, \infty))$
  - ∗ propagates b from $(p, \{\tau\})$
    - "simplifying" $\bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} [\![node']\!]$



Algorithm Overview

- Initially all nodes have the form $(\alpha, [0, \infty))$
- For each received message $\mathsf{report}(p, b, \tau)$, the monitor
  - ∗ splits all nodes (α, J) with $\tau \in J$
    into 3 new nodes: $(\alpha, J \cap [0, \tau))$, $(\alpha, \{\tau\})$, $(\alpha, J \cap (\tau, \infty))$
  - ∗ propagates b from $(p, \{\tau\})$
    - "simplifying" $\bigvee_{g \in guards} \bigwedge_{node' \in precs(g)} [\![node']\!]$



  - ∗ removes nodes (α, J) with J a complete interval

Complexity (not analyzed)

Complexity of checking φ on the first n messages

- For MTL, no failures, in-order messages: $\mathcal{O}(n \cdot |\varphi|)$
- Our setting (conjecture):

$$\begin{cases} \mathcal{O}\big(n \cdot |\varphi| \cdot \max_I (r(I) - \ell(I))\big) & \text{if } \max_I r(I) < \infty, \\ \mathcal{O}(n^2 \cdot |\varphi|) & \text{otherwise.} \end{cases}$$

- Note: without optimization, we would get:

$$\begin{cases} \mathcal{O}\big(n \cdot |\varphi| \cdot \max_I r(I) \cdot (r(I) - \ell(I))\big) & \text{if } \max_I r(I) < \infty, \\ \mathcal{O}(n \cdot n^2 \cdot |\varphi|) & \text{otherwise.} \end{cases}$$

- Future (open) work: an algorithm that works in $\mathcal{O}(n \cdot |\varphi|)$.