Failure-aware Runtime Verification of Distributed Systems

David Basin¹  Felix Klaedtke²
Eugen Zălinescu³

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
In a distributed system
- components might crash
- components communicate asynchronously
- network failures may occur
- 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: \( p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi S_I \varphi \mid \varphi U_I \varphi \)

where \( p \in P \) and \( I \) is a non-empty interval over \( \mathbb{Q}_+ \)
Metric Temporal Logic

**Syntax:**
\[ p | \neg \varphi | \varphi \lor \varphi | \varphi \mathcal{S}_I \varphi | \varphi \mathcal{U}_I \varphi \]
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 \( \sigma_i \in \{t, f\}^P \) and \( \tau_i \in \mathbb{Q}_+ \)
* \( [w, i \models \varphi] \in \{t, f\} \)

\[
\begin{align*}
\tau_i - \tau_j &\in I \\
0 &\quad \tau_j \quad \tau_{j+1} \quad \tau_{j+2} \quad \cdots \quad \tau_{i-2} \quad \tau_{i-1} \quad \tau_i \\
\vdots &\quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
p &\quad p \quad \cdots \quad p \quad p \quad p \\
q &
\end{align*}
\]

\( \varphi = p \mathcal{S}_I q \)
Metric Temporal Logic

- **Syntax:** \( p | \neg \varphi | \varphi \lor \varphi | \varphi S_I \varphi | \varphi U_I \varphi \)
  
  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 \( \sigma_i \in \{t, f\}^P \) and \( \tau_i \in \mathbb{Q}_+ \)
  
  * \( [w, i \models \varphi] \in \{t, f\} \)

\[ \tau_i - \tau_j \in I \]

\begin{align*}
0 & \quad \tau_j & \quad \tau_{j+1} & \quad \tau_{j+2} & \cdots & \tau_{i-2} & \tau_{i-1} & \tau_i \\
\text{p} & \quad \text{t} & \quad \text{t} & \quad \cdots & \quad \text{t} & \quad \text{t} & \quad \text{t} & \quad \text{t} \\
\text{q} & \quad \text{t} \\
\text{pS}_I \text{q} & \quad \text{t} \end{align*}
Three-valued MTL

- Additional truth value $\bot$ (‘unknown’) represents a knowledge gap.

- Truth tables given by strong Kleene logic

<table>
<thead>
<tr>
<th></th>
<th>$\neg$</th>
<th>$\lor$</th>
<th>$\land$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
</tr>
<tr>
<td>$f$</td>
<td>$t$</td>
<td>$f$</td>
<td>$f$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$t$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>
Three-valued MTL

- Additional truth value \( \bot \) (’unknown’) represents a knowledge gap.
  - truth tables given by strong Kleene logic

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\neg & t & f & t & f & t & f & t & f & t \\
\hline
\top & f & t & f & t & f & t & f & t & f \\
\hline
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\end{array}
\quad
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\lor & t & f & t & f & t & f & t & f & t \\
\hline
\top & t & t & t & t & t & t & t & t & t \\
\hline
\top & f & f & f & f & f & f & f & f & f \\
\hline
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\end{array}
\quad
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\land & t & f & t & f & t & f & t & f & t \\
\hline
\top & f & t & f & t & f & t & f & t & f \\
\hline
\top & \top & \top & \top & \top & \top & \top & \top & \top & \top \\
\hline
\top & \bot & \bot & \bot & \bot & \bot & \bot & \bot & \bot & \bot \\
\end{array}
\]

- Satisfaction relation lifted to \( \{ \bot, f, t \} \)
  - letters \((\sigma, \tau)\) with \(\sigma \in \{ \bot, f, t \}^P\)
  - \([w, i \models \varphi] \in \{ \bot, f, t \}\), e.g. \([w, i \models \neg \varphi] = \neg[w, i \models \varphi]\)

\[
\begin{array}{cccccccc}
0 & \tau_j & \tau_{j+1} & \tau_{j+2} & \cdots & \tau_{i-2} & \tau_{i-1} & \tau_i \\
\hline
p & f & t & t & \cdots & t & f & \bot \\
q & f & f & f & \cdots & f & f & \bot \\
P S_I q & t & t & \cdots & t & f & \bot \\
\end{array}
\]
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. \( \text{report}(q, f, 2.3) \)

**Example**

\[ \varphi = \alpha \land \beta \]
\[ \alpha = \neg p \ S \ q \]
\[ \beta = \Box_{[1,2]} q \]
Requirements on Monitors

- System behavior captured by
  - the set \( O \) of messages sent by system components, or
  - a timed word \( w_0 \)

\[
\text{report}(p, t, 1.0) \quad \text{report}(q, f, 2.2) \quad \text{report}(q, t, 3.0)
\]

\[
\begin{array}{ccc}
0 & 1.0 & 2.2 & 3.0 \\
\hline
p & t & \perp & \perp \\
q & \perp & f & t \\
\end{array}
\]
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 $M(m_i) \subseteq \mathbb{Q}_+ \times \{t, f\}$
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 $M(m_i) \subseteq \mathbb{Q}_+ \times \{t, f\}$

Goal. Given formula $\varphi$ and set $O$ of sent messages, design $M_\varphi$:

- 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 $M(m_i) \subseteq Q_+ \times \{t, f\}$

Goal. Given formula $\varphi$ and set $O$ of sent messages, design $M_\varphi$:

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

- Completeness:
  - If $[w_O, \tau \models \varphi] = b \in \{f, t\}$, then there 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 $\varphi$ 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: $[w, i \models \Diamond[1, 2]t]$ 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: \([w, i \models \Box_{[1,2]} t]\) may be \(f\)

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

![Interval Example](image-url)
(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: $[w, i \models \Diamond_{[1,2]} t]$ may be $f$

$J$ is complete if $M_\varphi$ knows of all letters $(\sigma, \tau)$ of $w_O$ with $\tau \in J$

* New message type: notify$(C, \tau, s)$ — $C$ has made an observation at $\tau$
  * $s$ is the sequence number of this observation
  * $J$ is complete iff $M_\varphi$ 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: $[w, i \models \Box_{[1,2]} t]$ may be $f$

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

```
0                     2.5          3.0                     \infty
\hline
\text{notify}(C, 2.5, 1)
\hline
\text{notify}(D, 3.0, 2)
```
(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: \([w, i \models \Diamond_{[1,2]} t]\) may be false

- \(J\) is complete if \(M_\varphi\) knows of all letters \((\sigma, \tau)\) of \(w_O\) with \(\tau \in J\)
  * New message type: notify\((C, \tau, s)\) — \(C\) has made an observation at \(\tau\)
    * \(s\) is the sequence number of this observation
    * \(J\) is complete iff \(M_\varphi\) has received all notify\((\_, \tau, \_)\) with \(\tau \in J\)
  * Example (2 components: \(C\) and \(D\))
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 \((\sigma, \tau)\) in \(w\) there is a letter \((\sigma', J)\) in \(u\) such that \(\tau \in J\) and \(\sigma' \leq \sigma\).

If \([u, J \models \varphi] = b \in \{f, t\}\), then \([w, \tau \models \varphi] = b\) for any \(w \in TW(u)\) and any \(\tau\) 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

notify(C, 2.0, 1)
report(p, f, 2.0)

0 ≤ p ≤ ∞
q
Example

notify($C$, 2.0, 1)
Example

\[ \text{notify}(C, 2.0, 1) \]

\[ \text{report}(p, f, 2.0) \]
Overview of Monitoring Algorithm

- \{w_O\} \subseteq \cdots \subseteq TW(u_{\ell+1}) \subseteq TW(u_\ell) \subseteq \cdots \subseteq TW(u_0)

- If \( [u_\ell, J \models \varphi] = b \in \{f, t\} \),
  then \( [w_O, \tau \models \varphi] = b \) for any \( \tau \) 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 \([u_\ell, J \models \varphi] = b \in \{f, t\}\),
  then \([w_O, \tau \models \varphi] = b\) for any \(\tau\) with \(\tau \in J\).

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

Nodes \((\psi, J)\) are linked according to the formula defining the semantics of \(\psi\)

\[
[u_\ell, J \models \alpha \lor \beta] := [u_\ell, J \models \alpha] \lor [u_\ell, J \models \beta]
\]

\[
[(\alpha \lor \beta, J)] = [(\alpha, J)] \lor [(\beta, J)]
\]
Nodes \((\psi, J)\) are linked according to the formula defining the semantics of \(\psi\)

\[
[u_\ell, J \models \alpha \lor \beta] := [u_\ell, J \models \alpha] \lor [u_\ell, J \models \beta]
\]

\[
[(\alpha \lor \beta, J)] = [(\alpha, J)] \lor [(\beta, J)]
\]

**Terminology**

* Nodes have guards.
* Guards have preconditions.

**Intuition** (such formulas are in DNF)

\[
[\text{node}] = \bigvee_{g \in \text{guards}} \bigwedge_{\text{node}' \in \text{precs}(g)} [\text{node'}]
\]
Monitor State

Nodes $(\psi, J)$ are linked according to the formula defining the semantics of $\psi$

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

$$
[(\alpha \lor \beta, J)] = [(\alpha, J)] \lor [(\beta, J)]
$$

Terminology

* Nodes have guards.
* Guards have preconditions.

Intuition (such formulas are in DNF)

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

Nodes \((\psi, J)\) are linked according to the formula defining the semantics of \(\psi\)

\[
[u_\ell, J \models \alpha \land \beta] := [u_\ell, J \models \alpha] \land [u_\ell, J \models \beta]
\]

\[
[(\alpha \land \beta, J)] = [(\alpha, J)] \land [(\beta, J)]
\]

Terminology

* Nodes have guards.
* Guards have preconditions.

Intuition (such formulas are in DNF)

\[
[node] = \bigvee_{g \in \text{guards}} \land_{node' \in \text{precs}(g)} [node']
\]
Monitor State

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

\[
\begin{align*}
\llbracket u_\ell, J \models \Diamond I \beta \rrbracket &:= \bigvee_{K: (J-K) \cap I \neq \emptyset} \llbracket u_\ell, K \models \beta \rrbracket \\
\llbracket (\Diamond I \beta, J) \rrbracket &:= \bigvee_{K: (J-K) \cap I \neq \emptyset} \llbracket (\beta, K) \rrbracket
\end{align*}
\]

Terminology
- Nodes have guards.
- Guards have preconditions.

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

\((\beta, K) \sim (\Diamond I \beta, J)\) iff there are \(\tau \in J\) and \(\kappa \in K\) such that \(\tau - \kappa \in I\)
Example

\[ \text{notify}(C, 0, 1) \]

\[ \text{report}(p, f, 0) \]

\[ \text{[0,1] } p \]
Example

```
notify(\(C,2.0,2\))
```

Diagram:
- Two horizontal bars representing intervals [0, 1] and [2.0, \(\infty\)]
- An arrow labeled \(\infty\) pointing from 1 to \(\infty\)
- An arrow labeled 2 pointing from 2.0 to 1
- An arrow labeled \(\infty\) pointing from 0 to \(\infty\)
- An arrow labeled \(\infty\) pointing from 0 to 2.0
- An arrow labeled \(\infty\) pointing from 2.0 to 0
- An arrow labeled \(\infty\) pointing from 0 to 1
Example

notify($C, 2.0, 2$)

report($p, f, 2.0$)
Example (cont.)

\[ \text{notify}(C, 0.5, 1) \]

\[ \text{report}(p, t, 0.5) \]
Example (cont.)

```
notify(C, 0.5, 1)
```

```
report(p, t, 0.5)
```
The Since and Until Cases

From $\bigvee_{j \in \{\ell \in \mathbb{N} | \tau_i - \tau_\ell \in I\}} \left( \llbracket w, j \models \beta \rrbracket \land \bigwedge_{j < k \leq i} \llbracket w, k \models \alpha \rrbracket \right)$:

$\begin{align*}
(\alpha, L_1) & \quad (\alpha, L_2) & \quad (\alpha, L_3) & \quad \ldots & \quad (\alpha, L_{n-1}) & \quad (\alpha, L_n) \\
(\beta, K_1) & \quad (\beta, K_2) & \quad \ldots & \quad & \quad (\beta, K_n) \\
(\gamma, J) & & & & 
\end{align*}$

$(\beta, K) \leadsto (\alpha S_I \beta, J)$ iff there are $\tau \in J$, $\kappa \in K$ such that $\tau - \kappa \in I$
The Since and Until Cases

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

$$(\alpha, L_1) (\alpha, L_2) (\alpha, L_3) \ldots (\alpha, L_{n-1}) (\alpha, L_n)$$

$$(\beta, K_1) (\beta, K_2) \ldots (\beta, K_n)$$

$$(\gamma, J)$$

$$(\beta, K) \leadsto (\alpha S_I \beta, J)$$ iff there are $\tau \in J$, $\kappa \in K$ such that $\tau - \kappa \in I$

Key optimization: only one link $(\alpha, \_)$ $\leadsto (\alpha S_I \beta, J)$ 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)
  \[ \downarrow_r \cdot \text{req}(r) \rightarrow \Diamond \text{ack}_{[0,5]}(r) \]
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 \([0, \infty)\)
  - \(o_i \in \{\bot, f, t\}\) represents whether there are observations in \(J_i\)
    - if \(|J_i| = 1\) then \(o_i = t\)
    - 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:
    \[
    \llbracket w, i \models \alpha \leq S_I \beta \rrbracket := \bigvee_{j \in \{\ell \in \mathbb{N} | \ell \leq i, \tau_i - \tau_\ell \in I\}} \left( \llbracket w, j \models \beta \rrbracket \land \bigwedge_{j < k \leq i} \llbracket w, k \models \alpha \rrbracket \right)
    
    \text{or}
    
    \[
    \llbracket w, i \models \alpha \leq S_I \beta \rrbracket := \bigvee_{j \leq i} \left( tc(i, j) \land \llbracket w, j \models \beta \rrbracket \land \bigwedge_{j < k \leq i} \llbracket w, k \models \alpha \rrbracket \right)
    
    \text{where}
    
    \[
    tc(i, j) := \begin{cases} 
    t & \text{if } \tau_i - \tau_j \in I \\
    f & \text{otherwise}
    \end{cases}
    \]
  - on i-words:
    \[
    \llbracket u, i \models \alpha \leq S_I \beta \rrbracket := \bigvee_{j \leq i} \left( tci(i, j) \land o_j \land \llbracket u, j \models \beta \rrbracket \land \bigwedge_{j < k \leq i} \llbracket u, k \models \alpha \rrbracket \right)
    
    \text{where}
    
    \[
    tci(i, j) := \begin{cases} 
    t & \text{if } \tau - \tau' \in I, \text{ for all } \tau \in J_i \text{ and } \tau' \in J_j \\
    f & \text{if } \tau - \tau' \notin I, \text{ for all } \tau \in J_i \text{ and } \tau' \in J_j \\
    \bot & \text{otherwise}
    \end{cases}
    \]
Algorithm Overview

- Initially all nodes have the form \((\alpha, [0, \infty))\)
- For each received message \(\text{report}(p, b, \tau)\), the monitor
  * splits all nodes \((\alpha, J)\) 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 \(\text{report}(p, b, \tau)\), the monitor
  * splits all nodes \((\alpha, J)\) 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” \(\bigvee_{g \in \text{guards}} \bigwedge_{\text{node} \in \text{precs}(g)} \text{node}'\)

```
\begin{align*}
\alpha & \quad t \\
\beta & \quad \rightarrow \\
\alpha \land \beta & \quad \rightarrow \\
\end{align*}
```

```
\begin{align*}
\alpha & \equiv f \\
\beta & \quad \rightarrow \\
\alpha \land \beta & \quad \rightarrow \\
\end{align*}
```
Algorithm Overview

- Initially all nodes have the form \((\alpha, [0, \infty))\)
- For each received message \(\text{report}(p, b, \tau)\), the monitor
  
  * splits all nodes \((\alpha, J)\) 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” \(\bigvee_{g \in \text{guards}} \bigwedge_{\text{node}' \in \text{precs}(g)} \text{node}'\)
  
  * removes nodes \((\alpha, J)\) with \(J\) a complete interval
Complexity (not analyzed)

**Complexity** of checking $\varphi$ on the first $n$ messages

- For MTL, no failures, in-order messages: $O(n \cdot |\varphi|)$
- Our setting (conjecture):
  
  $$
  \begin{cases}
  O(n \cdot |\varphi| \cdot \max_I (r(I) - \ell(I))) & \text{if } \max_I r(I) < \infty, \\
  O(n^2 \cdot |\varphi|) & \text{otherwise}.
  \end{cases}
  $$

- Note: without optimization, we would get:
  
  $$
  \begin{cases}
  O(n \cdot |\varphi| \cdot \max_I r(I) \cdot (r(I) - \ell(I))) & \text{if } \max_I r(I) < \infty, \\
  O(n \cdot n^2 \cdot |\varphi|) & \text{otherwise}.
  \end{cases}
  $$

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