Monitoring of the AUTOSAR Timing Extensions with TeSSLa

Überwachung der AUTOSAR Timing Extensions mittels TeSSLa

Hendrik Streichhahn

December 8, 2020

Table of Contents

Motivation - Timing

- Timing is fundamental for
  - Reliability and availability
  - Safety and security

Motivation - Timing

- Timing is fundamental for
  - Reliability and availability
  - Safety and security
- Timing problems
  - Difficult to identify, debug and solve
  - Especially in Cyber-Physical Systems

Motivation - Timing

- Timing is fundamental for
  - Reliability and availability
  - Safety and security
- Timing problems
  - Difficult to identify, debug and solve
  - Especially in Cyber-Physical Systems
- ISO 26262 ("Road vehicles – Functional safety")
  - Requires *Freedom of Interference*
  - Not possible to ensure, if timing problems exists

Motivation - Use Case[AUT18a]

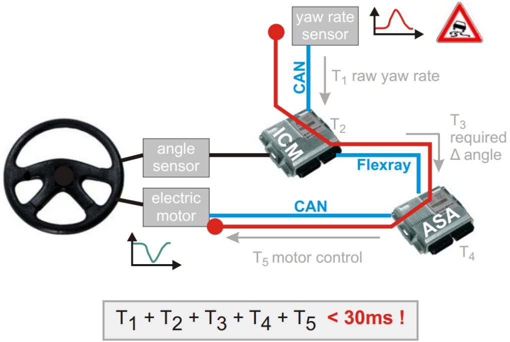


Figure: Figure 1.2 from [AUT18a]: Set-up and end-to-end timing requirement (red line) from an active steering project

AUTOSAR[AUT18b]

- Standardized Software Framework

AUTOSAR[AUT18b]

- Standardized Software Framework

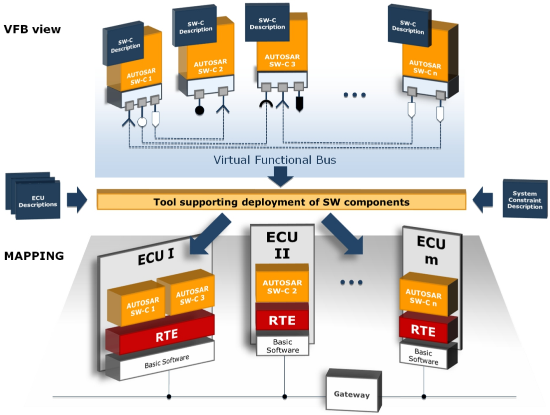


Figure: AUTOSAR Overview [Aut]

AUTOSAR[AUT18b]

- Standardized Software Framework

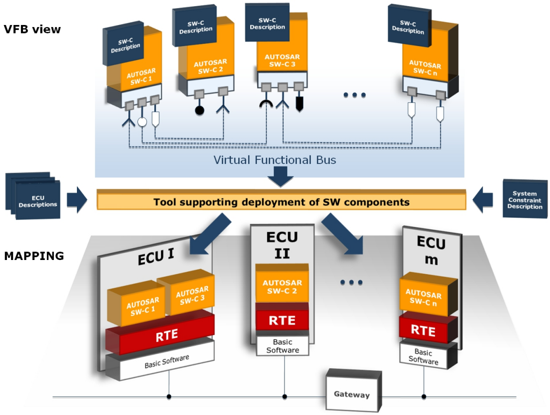


Figure: AUTOSAR Overview [Aut]

- AUTOSAR 4.0: Timing Extensions added
  - Allows precise timing constraint specification

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data
- *SynchronizationTimingConstraint*
  - Events from different subsystems occur synchronized

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data
- *SynchronizationTimingConstraint*
  - Events from different subsystems occur synchronized
- *SynchronizationPointConstraint*
  - *Source* events must occur before *target* events

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data
- *SynchronizationTimingConstraint*
  - Events from different subsystems occur synchronized
- *SynchronizationPointConstraint*
  - *Source* events must occur before *target* events
- *OffsetTimingConstraint*
  - Specifies the minimal and maximal time distance between arbitrary events

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data
- *SynchronizationTimingConstraint*
  - Events from different subsystems occur synchronized
- *SynchronizationPointConstraint*
  - *Source* events must occur before *target* events
- *OffsetTimingConstraint*
  - Specifies the minimal and maximal time distance between arbitrary events
- *ExecutionOrderConstraint*
  - Specifies the order, in which a list of events or executables must start and finish

AUTOSAR TIMEX Constraints

- *EventTriggeringConstraints*
  - E.g. *PeriodicEventTriggering*, *ConcretePatternEventTriggering*, *BurstPatternEventTriggering*
- *LatencyTimingConstraint*
  - Specifies time between *stimulus* events and their associated *response* events of an event chain
- *AgeConstraint*
  - Specifies minimum and maximum of received data
- *SynchronizationTimingConstraint*
  - Events from different subsystems occur synchronized
- *SynchronizationPointConstraint*
  - *Source* events must occur before *target* events
- *OffsetTimingConstraint*
  - Specifies the minimal and maximal time distance between arbitrary events
- *ExecutionOrderConstraint*
  - Specifies the order, in which a list of events or executables must start and finish
- *ExecutionTimeConstraint*
  - Specifies the minimal and maximal runtime of an executable

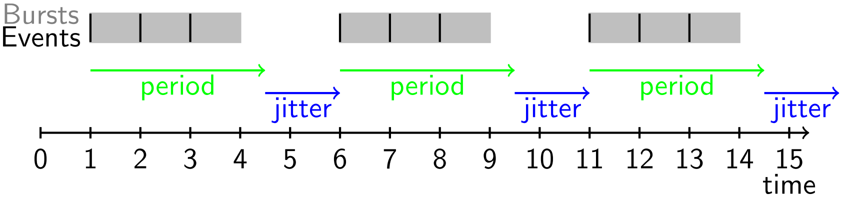
AUTOSAR TIMEX Constraints - Informal Definition

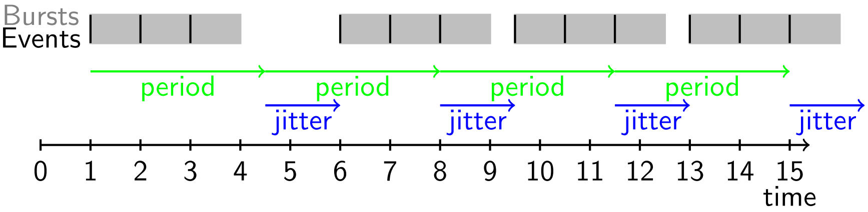
- Problem: Informal definitions, only textual often
  - Leaves room for different interpretations
  - Unsuitable for automated monitoring

AUTOSAR TIMEX Constraints - Informal Definition

- Problem: Informal definitions, only textual often
  - Leaves room for different interpretations
  - Unsuitable for automated monitoring
- Example *BurstPatternEventTriggering*
  - Parameter **patternPeriod** (time value)
    "The optional parameter "PatternPeriod" specifies the time distance between the beginnings of subsequent repetitions of the given burst pattern."[AUT18b]
  - Parameter **patternJitter** (time value)
    "The optional parameter "PatternJitter" specifies the deviation of the time interval's starting point from the beginning of the given period. This parameter is only applicable in conjunction with the parameter "Pattern Period"." [AUT18b]

BurstPatternEventTriggering





TADL2

- European ITEA2 Project TIMMO2USE (Timing Model, 2010-2012)[Blo+12]
  - worked on formally defining a timing language syntax, semantics and metamodel
- **T**iming **A**ugmented **D**escription **L**anguage v. 2(TADL2)
  - Timing Extension for **E**lectronics **A**rchitecture and **S**oftware **T**echnology-**A**rchitecture **D**escription **L**anguage (EAST-ADL)
- Constraints are strictly formally defined
  - TiCL (**Ti**ming **C**onstraint **L**ogic)

Groups of Constraints

- Response Constraints
  - *Delay-*, *StrongDelay-*, *Reaction-* and *AgeConstraint*

Groups of Constraints

- Response Constraints
  - *Delay-*, *StrongDelay-*, *Reaction-* and *AgeConstraint*
- Event Triggering
  - *Repeat-*, *Repetition-*, *Sporadic-*, *Periodic-*, *Pattern-*, *Burst-* and *ArbitraryConstraint*

Groups of Constraints

- Response Constraints
  - *Delay-*, *StrongDelay-*, *Reaction-* and *AgeConstraint*
- Event Triggering
  - *Repeat-*, *Repetition-*, *Sporadic-*, *Periodic-*, *Pattern-*, *Burst-* and *ArbitraryConstraint*
- Synchronization
  - *Synchronization-*, *StrongSynchronization-*, *InputSynchronization-* and *OutputSynchronizationConstraint*

Groups of Constraints

- Response Constraints
  - *Delay-*, *StrongDelay-*, *Reaction-* and *AgeConstraint*
- Event Triggering
  - *Repeat-*, *Repetition-*, *Sporadic-*, *Periodic-*, *Pattern-*, *Burst-* and *ArbitraryConstraint*
- Synchronization
  - *Synchronization-*, *StrongSynchronization-*, *InputSynchronization-* and *OutputSynchronizationConstraint*
- Others
  - *ExecutionTime-*, *Comparison-* and *OrderConstraint*

TADL2 - DelayConstraint

- $DelayConstraint(source, target) \Leftrightarrow$
  $\forall x \in source : \exists y \in target : lower \leq y - x \leq upper$

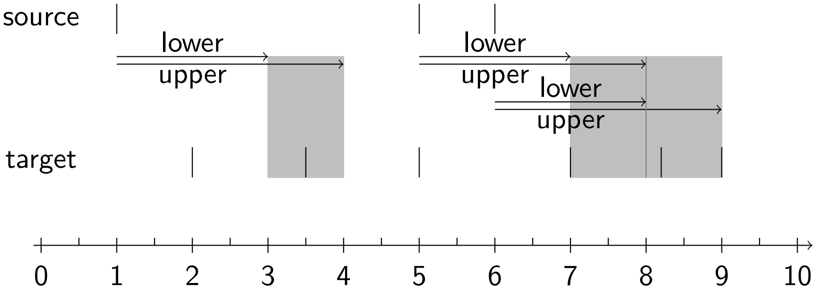


Figure: Example DelayConstraint - $lower = 2$, $upper = 3$

TADL2 - RepeatConstraint

- $RepeatConstraint(event, lower, upper, span) \Leftrightarrow$
  $\forall X \leq event : |X| = span + 1 \Rightarrow lower \leq \lambda([X]) \leq upper$

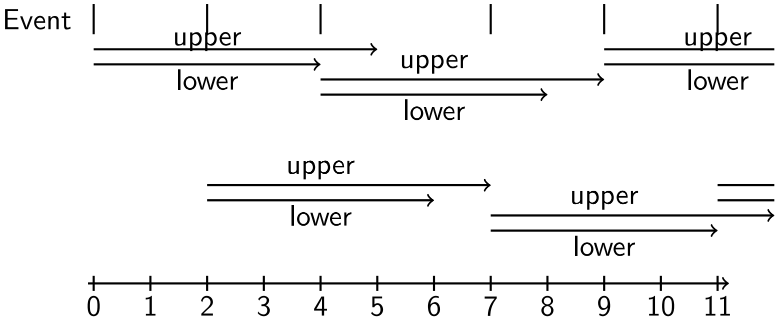


Figure: Example RepeatConstraint - $lower = 4$, $upper = 5$, $span = 2$

TADL2 - InputSynchronizationConstraint

- $InputSynchronizationConstraint(scope, tolerance) \Leftrightarrow$
  $\forall y \in scope_1.response : \exists t : \forall i : \exists x \in scope_i.stimulus :$
  $x.color = y.color$
  $\wedge (\forall x' \in scope_i.stimulus : x'.color = x.color \Rightarrow x \leq x')$
  $\wedge\, 0 \leq x - t \leq tolerance$

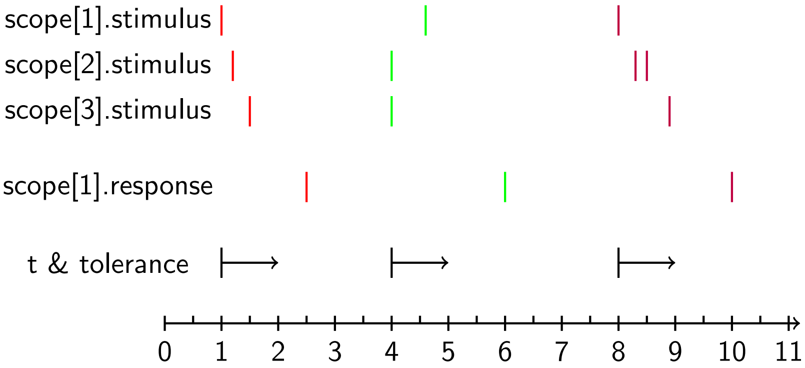


Figure: Example InputSynchronizationConstraint - $tolerance = 1$

AUTOSAR TIMEX 4.4.0 ⇔ TADL2

- Most AUTOSAR TIMEX Constraints can be expressed in TADL2 Constraints:

AUTOSAR TIMEX	**TADL2 Constraint**	Coverage
PeriodicEventTriggering	PeriodicConstraint	$\surd$
SporadicEventTriggering	SporadicConstraint	$\surd$
ConcretePatternEventTriggering	PatternConstraint	Minor differences
BurstPatternEventTriggering	BurstConstraint	Large differences
ArbitraryEventTriggering	ArbitraryConstraint	Minor differences, but irrelevant for monitoring
LatencyTimingConstraint	ReactionConstraint AgeConstraint	Minor differences
AgeConstraint	AgeConstraint	Minor differences
SynchronizationTimingConstraint	SynchronizationConstraint StrongSynchronizationConstraint OutputSynchronizationConstraint InputSynchronizationConstraint	$\surd$
SynchronizationPointConstraint	–	–
OffsetTimingConstraint	DelayConstraint	$\surd$
ExecutionOrderConstraint	multiple use of OrderConstraint	$\surd$
ExecutionTimeConstraint	ExecutionTimeConstraint	Minor differences (interruptions)

Monitorability

- Not every property can be infinitely monitored on infinite streams
  - Limited Resources (memory, time)

Monitorability

- Not every property can be infinitely monitored on infinite streams
  - Limited Resources (memory, time)
- Classification
  - Simple Monitorable
    - Worst case memory and runtime per event bounded independent of trace
    - No new timestamps required (*timestamp conservative*)

Monitorability

- Not every property can be infinitely monitored on infinite streams
  - Limited Resources (memory, time)
- Classification
  - Simple Monitorable
    - Worst case memory and runtime per event bounded independent of trace
    - No new timestamps required (*timestamp conservative*)
  - Simple Monitorable with Delay
    - Extension to Simple Monitorable
    - **Exactly one** new timestamp may be introduced

Monitorability

- Not every property can be infinitely monitored on infinite streams
  - Limited Resources (memory, time)
- Classification
  - Simple Monitorable
    - Worst case memory and runtime per event bounded independent of trace
    - No new timestamps required (*timestamp conservative*)
  - Simple Monitorable with Delay
    - Extension to Simple Monitorable
    - **Exactly one** new timestamp may be introduced
  - Not Simple Monitorable
    - Worst case memory and runtime per event is dependent of trace

Timestamps

- Infinite traces $\Rightarrow$ infinite large timestamps
  - Infinite large Timestamps cannot be stored in simple monitorable setting

[1] for example, a 64-bit unsigned integer variable is enough, to cover nanoseconds for 584.55 years

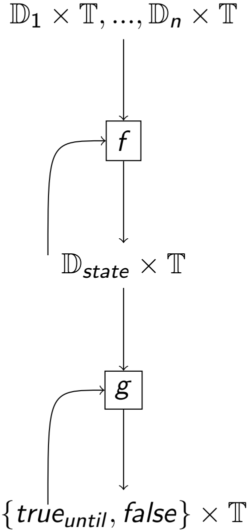
Timestamps

- Infinite traces $\Rightarrow$ infinite large timestamps
  - Infinite large Timestamps cannot be stored in simple monitorable setting
- Restriction of Timestamps
  - The first used timestamp has the value $t_0 = 0$
  - All used timestamps must be smaller than t_{max}.
    t_{max} must be big enough, so it is not reached in practical use [1].
  - The distance between two subsequent time values is predetermined, but arbitrary small.
  - The number of possible timestamps is significantly larger than the number of events.

---

[1] for example, a 64-bit unsigned integer variable is enough, to cover nanoseconds for 584.55 years

Simple Monitorability



- Input streams
- State transition function, worst case runtime independent from input streams
- State stream, worst case memory independent from input streams
- Evaluation Function, worst case runtime independent from input streams
- Output stream
- For given constraint parameters, a monitor can be build, which monitors the constraint infinitely with fixed resources

Theoretical Background - Deterministic Finite State Transducer[Ber79]

A *Deterministic Finite State Transducer*(DFST) is a 5-Tuple $(\Sigma, \Gamma, Q, q_0, \delta)$, where

- Σ is an input alphabet
- Γ is an output alphabet
- Q is a finite set of states, with initial state q_0
- $\delta : Q \times \Sigma \to Q \times \Gamma$ is a state transition function

The run of a DFST for an input word $w = w_0 w_1 w_2 ... \in \Sigma^\infty$ is a sequence $s_0 \xrightarrow{w_0/o_0} s_1 \xrightarrow{w_1/o_1} s_2 ...$, where $s_0 = q_0$, $\delta(s_i, w_i) = (s_{i+1}, o_i), i \geq 0$ and the output word $o = o_0 o_1 o_2 ... \in \Gamma^\infty$.

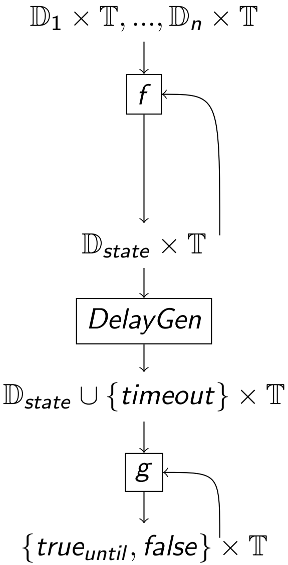
Simple Monitorability - Transducers

- State and state transition function are equivalent to a deterministic finite state transducer:
  - $Q = \mathbb{D}_{state}$ finite set of possible states with initial state q_0
  - $\Sigma = ((\mathbb{D}_1 \times \mathbb{T}), ..., (\mathbb{D}_n \times \mathbb{T}))$ input alphabet
  - $\Gamma = \mathbb{D}_{state}$ output alphabet and
  - $\delta : Q \times \Sigma \rightarrow Q \times \Gamma$ transition function.

Simple Monitorability - Transducers

- State and state transition function are equivalent to a deterministic finite state transducer:
  - $Q = \mathbb{D}_{state}$ finite set of possible states with initial state q_0
  - $\Sigma = ((\mathbb{D}_1 \times \mathbb{T}), ..., (\mathbb{D}_n \times \mathbb{T}))$ input alphabet
  - $\Gamma = \mathbb{D}_{state}$ output alphabet and
  - $\delta : Q \times \Sigma \rightarrow Q \times \Gamma$ transition function.
- Same goes for the output and the evaluation function:
  - $Q' = \{true_{until}, false\}$ states with initial state $true_{until}$
  - $\Sigma' = \mathbb{D}_{state} \times \mathbb{T}$ input alphabet
  - $\Gamma' = \{true_{until}, false\}$ output alphabet and
  - $\delta' : Q' \times \Sigma' \rightarrow Q' \times \Gamma'$ the transition function.

Simple Monitorability with Delay



- Input streams
- State transition function, worst case runtime independent from input streams
- State stream, worst case memory independent from input streams
- Delay Generator
- State stream including *timeouts*
- Evaluation function, worst case runtime independent from input streams
- Output stream

- For given constraint parameters, a monitor can be build, which monitors the constraint on infinitely with fixed resources

Theoretical Background - Timed Deterministic Finite State Transducer

Timed Deterministic Finite State Transducers(TDFST) are a 6-Tuple $(\Sigma, \Gamma, Q, q_0, C, \delta)$, where

- Σ is an input alphabet
- Γ is an output alphabet
- Q is a finite set of states, with initial state q_0
- C is a set of clocks
- $\delta : Q \times \Sigma \times \Theta(C) \to Q \times 2^C \times \Gamma$ is a state transition function, where for all $(q_a, \sigma_a, \vartheta_a, q'_a, R_a, \gamma_a), (q_b, \sigma_b, \vartheta_b, q'_b, R_b, \gamma_b) \in \delta$ the conjunction $\vartheta_a \wedge \vartheta_b$ is unsatisfiable.

Theoretical Background - Timed Deterministic Finite State Transducer

Let $v_i : C \to \mathbb{R}$ be functions that map each clock to its current value.
The run of a TDFST for an input word $w = (w_0, t_0)(w_1, t_1)(w_2, t_2)... \in (\Sigma, \mathbb{T})^\infty$ is a sequence $s_0, v_0 \xrightarrow[o_0]{(w_0,t_0),\vartheta_0,r_0} s_1, v_1 \xrightarrow[o_1]{(w_1,t_1),\vartheta_1,r_1} s_2, ...$ with output $o = o_0 o_1 o_2 ... \in \Gamma^\infty$, if, and only if,

- $s_0 = q_0$
- $\forall c \in C : v_0(c) = 0$
- $\forall i \geq 0 :$
  - $\delta(s_i, w_i, \vartheta_i) = (s_{i+1}, r_i, o_i)$
  - $\forall c \in r_i : v_{i+1} = v_i[c \leftarrow t_i]$
  - $t_i, v_i \models \vartheta_i$

Simple Monitorability with Delay - Transducers

- Like before
  - State and state transition function have equivalent transducer
  - Output and evaluation function have equivalent transducer

Simple Monitorability with Delay - Transducers

- Like before
  - State and state transition function have equivalent transducer
  - Output and evaluation function have equivalent transducer
- Delay generator as modified form of Timed Deterministic Finite State Transducers
  - ε-Transitions, that are guarded by clock constraints
  - Determinism must still be given

Delay Generator

Definition (Delay Generator)

A Delay Generator is a 6-Tuple $(\Sigma, \Gamma, Q, q_{start}, C, \delta)$, where

- $Q = \{q_{start}, q_{timeout}\} \cup \{q_{wait,i} | \forall i \in \mathbb{D}_{state}\}$ is a finite set of states with initial state q_{start}
- $\Sigma = \mathbb{D}_{state}$ is an input alphabet
- $\Gamma = \mathbb{D}_{state} \cup \{timeout\}$ is an output alphabet
- $C = \{c\}$ is a set of exactly one clock and
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Theta(C) \to Q \times 2^C \times \Gamma$ a state transition function. δ is defined as:

$$\begin{aligned}
\forall i \in \mathbb{D}_{state} &: \delta(q_{start}, i, \emptyset) = (q_{wait,i}, \{c\}, i) \\
\forall i, i' \in \mathbb{D}_{state} &: \delta(q_{wait,i'}, i, \{c \leq tmr(i')\}) = (q_{wait,i}, \{c\}, i) \\
\forall i \in \mathbb{D}_{state} &: \delta(q_{wait,i}, \varepsilon, \{c > tmr(i)\}) = (q_{timeout}, \emptyset, timeout) \\
\forall i \in \mathbb{D}_{state} &: \delta(q_{timeout}, i, \emptyset) = (q_{timeout}, \emptyset, timeout)
\end{aligned}$$

Delay Generator

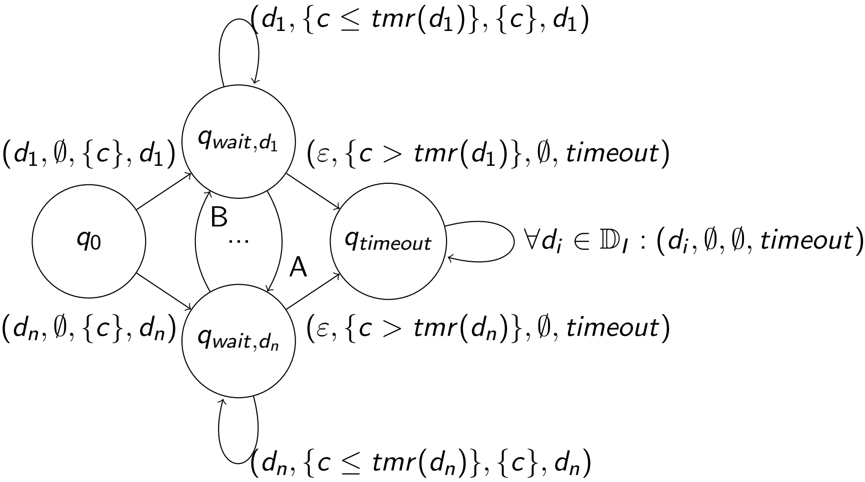


Figure: Description A means $(d_n, \{c < tmr(d_1)\}, \{c\}, d_n)$ and description B means $(d_1, \{c < tmr(d_n)\}, \{c\}, d_1)$.

Not Simple Monitorable

- Runtime per event or memory usage not bounded independently from trace
- Partition possible
  - Continuous growth of resource requirements
  - Unbounded growth only in worst cases

Simple Monitorable - Example *RepeatConstraint*

- $RepeatConstraint(event, lower, upper, span) \Leftrightarrow$
  $\forall X \leq event : |X| = span + 1 \Rightarrow lower \leq \lambda([X]) \leq upper$

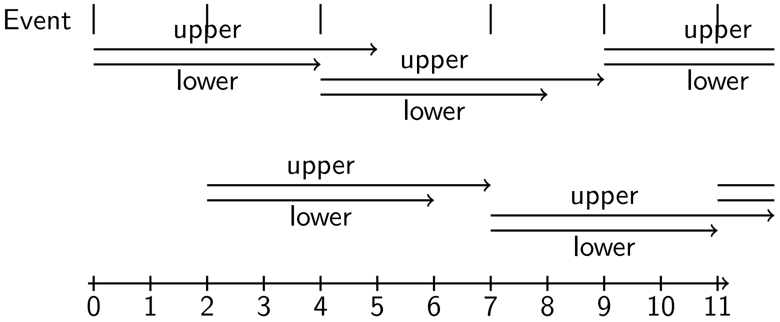


Figure: Example RepeatConstraint - $lower = 4$, $upper = 5$, $span = 2$

Example *RepeatConstraint*

- **State** Double linked list with the timestamps of the *span* latest events

Example *RepeatConstraint*

- **State** Double linked list with the timestamps of the *span* latest events
- **State Transition Function**
  $f(event, lower, upper, span, last\_state) :=$
  $List\_append(if(List\_size(last\_state) <= span)\ then$
  $last\_state$
  $else$
  $List\_tail(last\_state),$
  $time(event))$

Example *RepeatConstraint*

- **State** Double linked list with the timestamps of the *span* latest events
- **State Transition Function**
$$
\begin{aligned}
&f(event, lower, upper, span, last\_state) := \\
&\quad List\_append(if(List\_size(last\_state) <= span)\ then \\
&\qquad\qquad\qquad last\_state \\
&\qquad\qquad else \\
&\qquad\qquad\qquad List\_tail(last\_state), \\
&\qquad time(event))
\end{aligned}
$$
- **Delay**
$$delay(state) := (List\_head(state) + upper) - time(state)$$

Example *RepeatConstraint*

- **State** Double linked list with the timestamps of the *span* latest events
- **State Transition Function**
  $$
  \begin{aligned}
  &f(event, lower, upper, span, last\_state) := \\
  &\quad List\_append(if(List\_size(last\_state) <= span)\ then \\
  &\qquad\qquad\qquad last\_state \\
  &\qquad\qquad else \\
  &\qquad\qquad\qquad List\_tail(last\_state), \\
  &\qquad time(event))
  \end{aligned}
  $$
- **Delay**
  $$delay(state) := (List\_head(state) + upper) - time(state)$$
- **Evaluation Function**
  $$
  \begin{aligned}
  &g(lower, upper, span, state, last\_output) := \\
  &\quad last\_output\ \wedge \\
  &\qquad time(state) \geq list\_head(state) + lower \wedge \\
  &\qquad time(state) \leq list\_head(state) + upper
  \end{aligned}
  $$

DelayConstraint

- $DelayConstraint(source, target) \Leftrightarrow$
  $\forall x \in source : \exists y \in target : lower \leq y - x \leq upper$

DelayConstraint

- $\mathit{DelayConstraint}(\mathit{source}, \mathit{target}) \Leftrightarrow$
  $\forall x \in \mathit{source} : \exists y \in \mathit{target} : \mathit{lower} \leq y - x \leq \mathit{upper}$
- Time domain in TADL2 is $\mathbb{R}$
  - Arbitrary large number of events can be relevant for monitoring

DelayConstraint

- $DelayConstraint(source, target) \Leftrightarrow$
  $\forall x \in source : \exists y \in target : lower \leq y - x \leq upper$
- Time domain in TADL2 is $\mathbb{R}$
  - Arbitrary large number of events can be relevant for monitoring

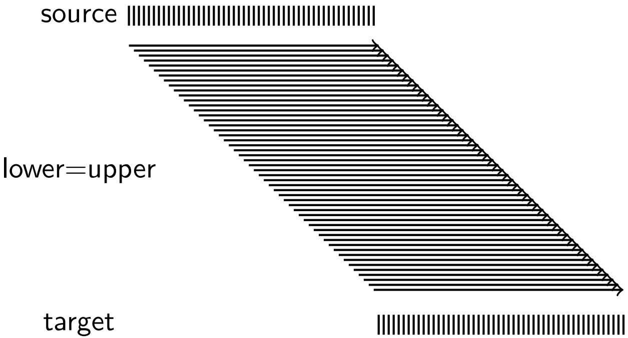


Figure: *DelayConstraint* with $lower = upper = 5$

- Worst case memory consumption is not bounded
  independently from input stream $\Rightarrow$ Not simple monitorable

InputSynchronizationConstraint

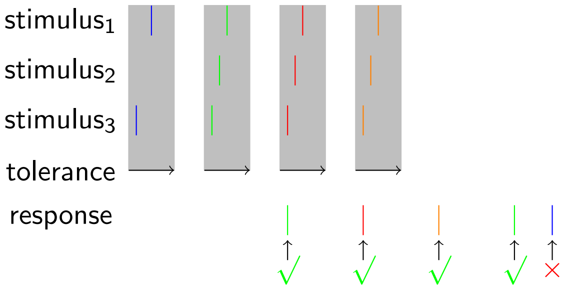
- $\mathit{InputSynchronizationConstraint}(\mathit{scope}, \mathit{tolerance}) \Leftrightarrow$
  $\forall y \in \mathit{scope}_1.\mathit{response} : \exists t : \forall i : \exists x \in \mathit{scope}_i.\mathit{stimulus} :$
  $\quad x.\mathit{color} = y.\mathit{color}$
  $\quad \wedge (\forall x' \in \mathit{scope}_i.\mathit{stimulus} : x'.\mathit{color} = x.\mathit{color} \Rightarrow x \leq x')$
  $\quad \wedge\, 0 \leq x - t \leq \mathit{tolerance}$

InputSynchronizationConstraint

- $InputSynchronizationConstraint(scope, tolerance) \Leftrightarrow$
  $\forall y \in scope_1.response : \exists t : \forall i : \exists x \in scope_i.stimulus :$
  $x.color = y.color$
  $\wedge (\forall x' \in scope_i.stimulus : x'.color = x.color \Rightarrow x \leq x')$
  $\wedge\, 0 \leq x - t \leq tolerance$
- All colors of fulfilled synchronization clusters must be stored
  - Not simple monitorable

InputSynchronizationConstraint

- $InputSynchronizationConstraint(scope, tolerance) \Leftrightarrow$
  $\forall y \in scope_1.response : \exists t : \forall i : \exists x \in scope_i.stimulus :$
  $x.color = y.color$
  $\wedge (\forall x' \in scope_i.stimulus : x'.color = x.color \Rightarrow x \leq x')$
  $\wedge\, 0 \leq x - t \leq tolerance$
- All colors of fulfilled synchronization clusters must be stored
  - Not simple monitorable



Monitorability Analysis Results for the 18 TADL2 Constraints

Figure: Simple Monitorable

Monitorability Analysis Results for the 18 TADL2 Constraints

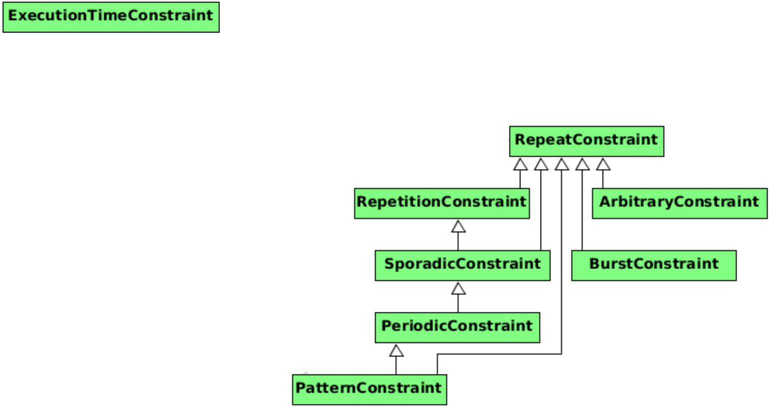


Figure: Simple Monitorable With Delay

Monitorability Analysis Results for the 18 TADL2 Constraints

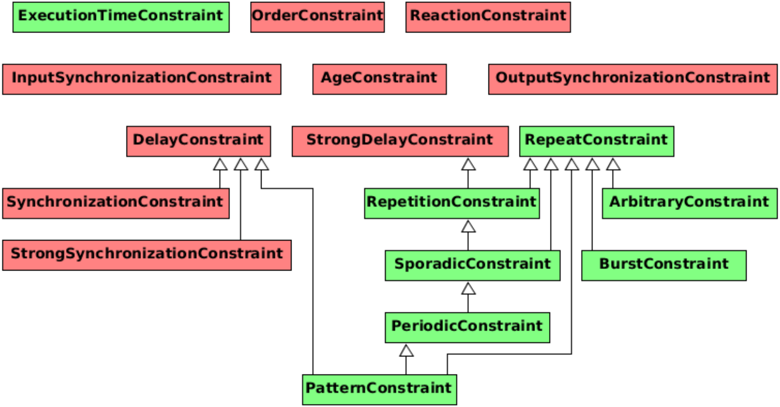


Figure: Simple Monitorable with Delay, Not simple Monitorable

Monitorability Analysis Results for the 18 TADL2 Constraints

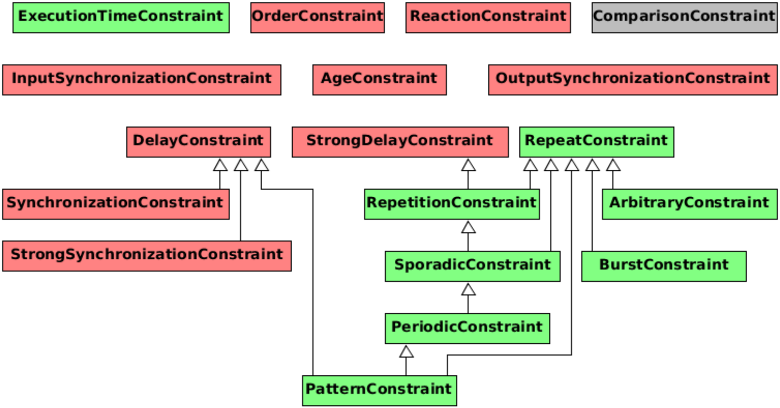


Figure: Simple Monitorable with Delay, Not simple Monitorable, Not applicable

Implementation

- According to the scheme presented in *Simple Monitorability (with Delay)*
  - State transition function
  - State stream
  - use of *delay*, if needed
  - Evaluation function

Implementation

- According to the scheme presented in *Simple Monitorability (with Delay)*
  - State transition function
  - State stream
  - use of *delay*, if needed
  - Evaluation function
- If possible, implementations were reused. E.g.:
  - **BurstConstraint**$(event, length, maxOccurences, minimum) \Leftrightarrow$
    $RepeatConstraint(event, length, \infty, maxOccurrences)$
    $\quad \wedge RepeatConstraint(event, minimum, \infty, 1)$
  - **ComparisonConstraint**
    Comparison of Timestamps already implemented in TeSSLa

Implementation *repeatConstraint* - State

```
def nLastTime[A](e: Events[A], n: Int): Events[Int]
    static if (n <= 0) then
        time(e)
    else
        last(nLastTime(e, n-1), e)

def repeatConstraint[A](e: Events[A], lower: Int,
    upper: Int, span: Int): Events[Bool] := {

  #stored state
  def latestSpanEventTimes := nLastTime(e, span)
```

Implementation *repeatConstraint* - Delay

```
# delay
def evaluateTimes = mergeUnit(e,
safeDelay((merge(prev(latestSpanEventTimes),
    firstEvent(time(e)))+ upper) − time(e),
  e))
```

Implementation *repeatConstraint* - Evaluation

```
allPreviousTrue(on(evaluateTimes,
 merge(
   default(
     (lower <=
       (time(evaluateTimes) - latestSpanEventTimes)
     upper >=
       time(evaluateTimes) - latestSpanEventTimes),
    true),
  firstEvent(time(e)) + upper > time(evaluateTimes)
}
```

Experimental Evaluation

- The implementations were tested on large Traces (10.000 Events), which were generated with different constraint parameters

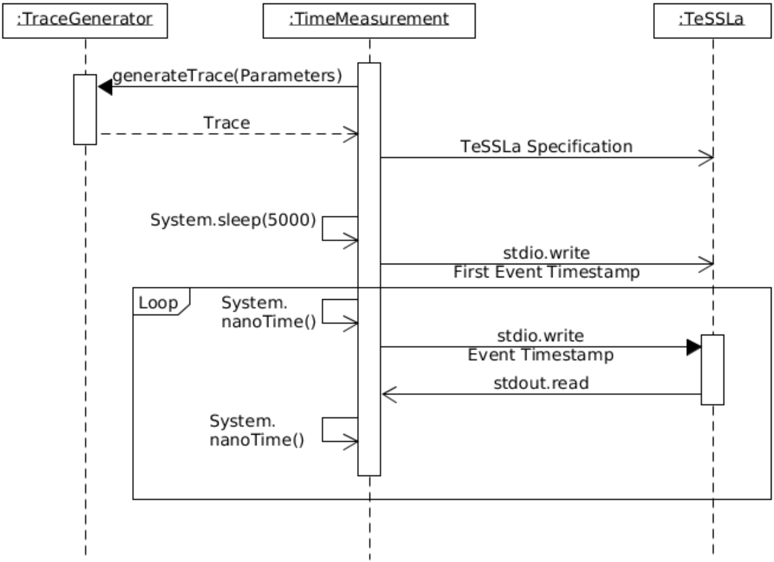
Experimental Evaluation

- The implementations were tested on large Traces (10.000 Events), which were generated with different constraint parameters
- Border cases were tested seperately

Experimental Evaluation

- The implementations were tested on large Traces (10.000 Events), which were generated with different constraint parameters
- Border cases were tested seperately
- Run time per input timestamp were measured
  - TeSSLa 1.0.12
  - Java 11.0.2
  - Windows 10.0.19041.0
  - Intel i5-6600k, 4.3 GHz

Time Measurement



Time Measurement - RepeatConstraint

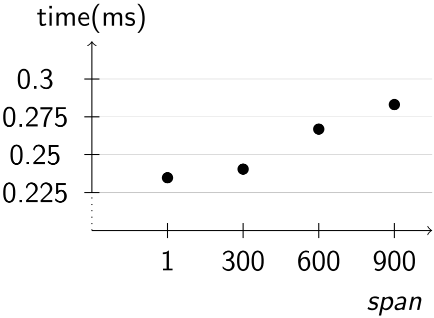


Figure: Average run times of the *RepeatConstraint* with the parameters $lower = 5000, upper = 7000$

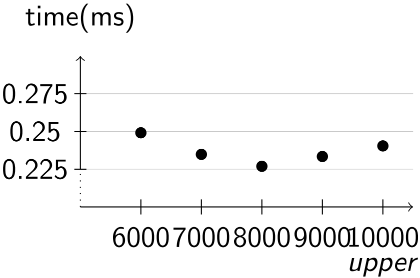


Figure: Average run times of the *RepeatConstraint* with the parameters $span = 1, lower = 5000$

Time Measurement - DelayConstraint

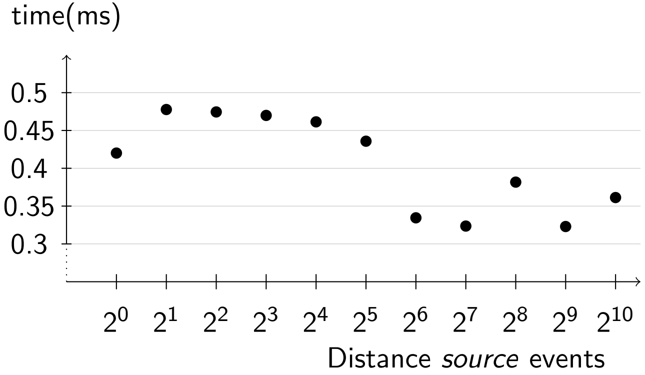


Figure: Average run times of the *DelayConstraint* with the parameters $lower = upper = 800$

Time Measurement - InputSynchronizationConstraint

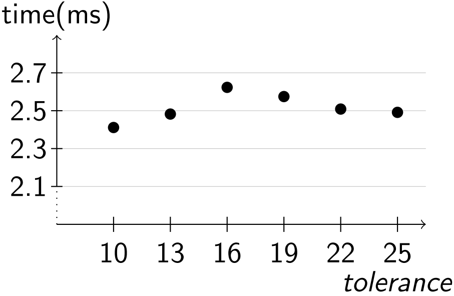


Figure: Average run times of the *InputSynchronizationConstraint* with 3 stimulus streams and a cluster distance of 2

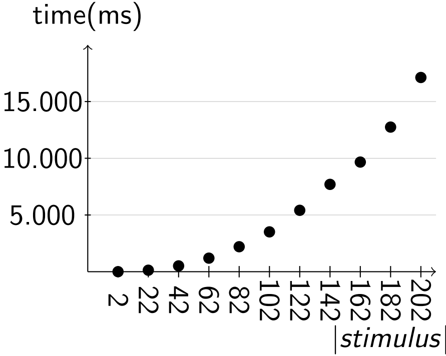


Figure: Average run times of the *InputSynchronizationConstraint* with a cluster distance of 2 and *tolerance* = 10

Summary

- TADL2 as formalization of the AUTOSAR TIMEX constraints
- Examation of the online Monitorability of the TADL2 constraints
- Implementation of a monitor for each TADL2 constraint
  - experimental evaluation on synthetic generated traces

References

AUTOSAR Basic Approach. https://web.archive.org/web/20170707082404/https://www.autosar.org/fileadmin/images/media_pictures/AUTOSAR_Basic_Approach.jpg. Accessed: 2020-12-06, archive from:2017-07-07.

AUTOSAR. *Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process*. Tech. rep. 4.4.0. AUTOSAR, 2018.

AUTOSAR. *Specification of Timing Extensions*. Tech. rep. 4.0. AUTOSAR, 2018.

Jean Berstel. *Transductions and Context-Free Languages -*. Wiesbaden: Vieweg+Teubner Verlag, 1979. ISBN: 978-3-519-02340-1.

Hans Blom et al. *TIMMO2USE Language syntax, semantics, metamodel V2*. Tech. rep. 1.2. ITEA2, 2012.