Report No. 1

Abstract

In an attempt to model checking timing constraints of automotive systems defined by TADL2 documents, we used the UPPAAL & KRONOS model checker. To achieve this, in the first step each ECU in our systems has to be modeled as Timed Automaton, and in the subsequent, modeling the entire system by composing those Automata. UPPAAL supports a restricted subclass of TCTL logic to specify timed properties and use symbolic techniques to model checking them. Supported properties by UPPAAL can be classified into reachability, safety, and liveness. KRONOS is a true timed model checker, with noticeable performances. KRONOS can support a wider subclass of TCTL logic and is not limited like UPPAAL. However, under its current form, it is mostly intended for advanced users, with a good knowledge of formal methods.

Timing Constraints

Our goal is to show how to verify all the 18 timing constraints by providing different examples of distributed networks of ECUs modeled with the specification language of UPPAAL and KRONOS. Most of the timing constraints have been modeled as Timed Automaton in UPPAAL, whereas we used TCTL for addressing the problem of their specification in KRONOS. In the following, we will introduce each timing constraint, describe its semantics, and verify them with indicated tools.

Timing constraints domains are individual events and event chains. The restrictions defined by constraints can be classified as repetition of an event or a set of them, delays between events, and synchronization of an events group. An event denotes a distinct form of state change in a running system. It takes place at distinct points in time which are called

its occurrences.

Delay Constraint

A DelayConstraint forces time distance limits between the occurrences of the source and target event without considering whether the matching target occurrence is caused by the corresponding source occurrence or not. A system behavior satisfies a DelayConstraint c if and only if for each occurrence x of $c.source$, there is an occurrence y of $c.target$ such that

$$c.lower \leq y - x \leq c.upper$$

The semantics description of delay constraint based on logic is declared as follows

$$\forall x \in \text{ source} : \exists y \in \text{ target} : \text{lower } \leq y - x \leq \text{ upper}$$

Even more, we can formally specify this property in TCTL as follows

$$\forall\Box(Source \Rightarrow \forall\Diamond^{[lower,upper]}Target)$$

By means to understood how an Event sequence satisfies a Delay constraint, you can consider figure 1 as an example.

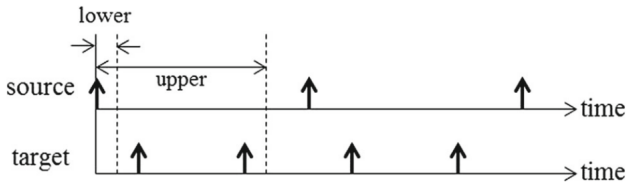

Figure 1: Event sequence satisfying a Delay constraint

To show how the delay constraint is verified on a simple ECU network by UPPAAL, figure 2 represents an example.

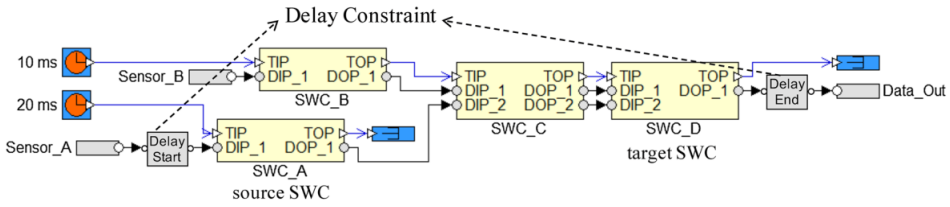

Figure 2: a simple ECU network

In this example, we have a network consisting of 4 ECUs, and each ECU is modeled as an independent software circuit. The software circuit runs a simple cycle scan whenever triggered by an event or a periodic clock. This cycle has three phases, reading data input port values, executing its predefined functions, and propagating the desired output on the data output ports. As shown in figure 2, SWC_A and SWC_B are both periodic components and are triggered periodically. In contrast, SWC_C and SWC_D are both sporadic components and are triggered by another component's event.

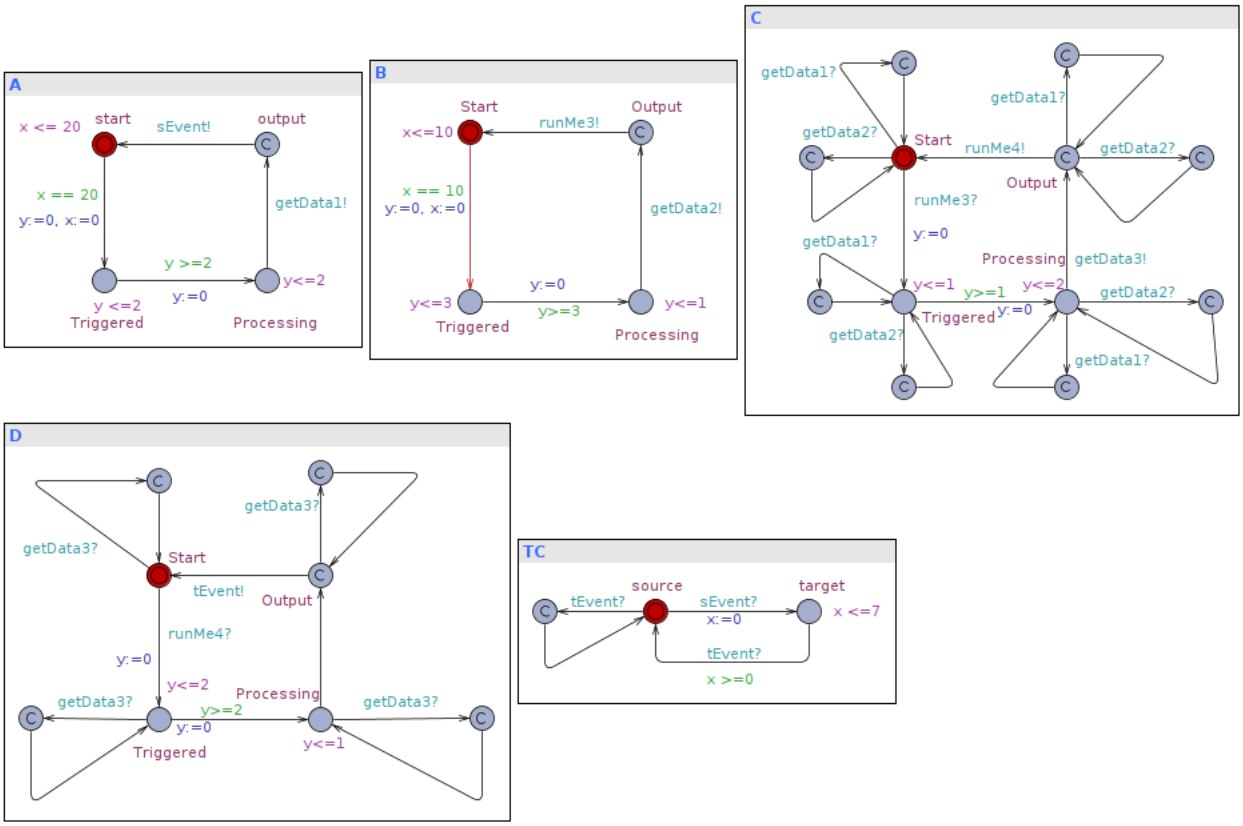

Figure 3: Timed Automaton models of ECUs

Figure 3 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Delay constraint as an observer timed automaton.

In order to model check our desired constraint on the system, the following two CTL formulas

$$\forall\Box(not\ Deadlock)$$

$$\forall\Box(Target \Rightarrow \forall\Diamond Source)$$

are needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[0, 7]$ as illustrated in figure 4.

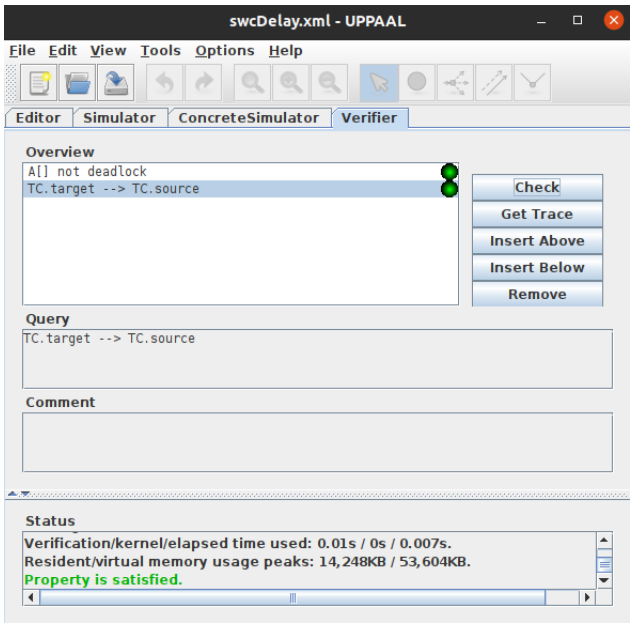

Figure 4: Results of Model Checking

Strong Delay Constraint

A StrongDelayConstraint forces time distance limits between each indexed occurrence of the source and identically indexed occurrence of the target event. Matching of the target occurrence caused by the corresponding source event occurrence is vital for this constraint.

A system behavior satisfies a StrongDelayConstraint c if and only if $c.source$ and $c.target$ have the same number of occurrences, and for each index i, if there is an i^{th} occurrence of $c.source$ at time x there is also an i^{th} occurrence of $c.target$ at time y such that

$$c.lower \leq y - x \leq c.upper$$

The semantics description of delay constraint based on logic is declared as follows

$$|source| = |target| \wedge \forall i : \forall x : x = source(i) \Rightarrow \exists y : y = target(i) \wedge lover \leq y - x \leq upper$$

Unfortunately, TCTL is unable to specify this kind of property, as it does not support indexed events. By means to understood how an Event sequence satisfies a Strong Delay constraint, you can consider figure 5 as an example. To show how the strong delay con-

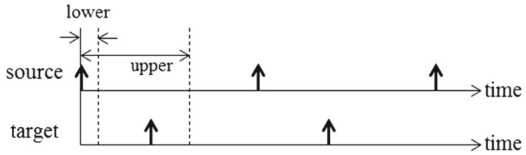

Figure 5: Event sequence satisfying a Strong Delay constraint

straint is verified on a simple ECU network by UPPAAL, figure 6 represents an example.

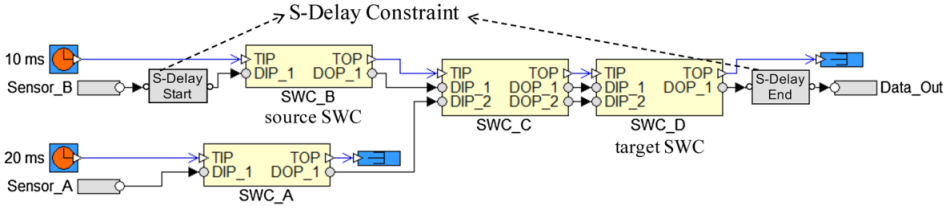

Figure 6: another simple ECU network

This example is very similar to figure 2. Figure 7 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Strong Delay constraint as an observer timed automaton.

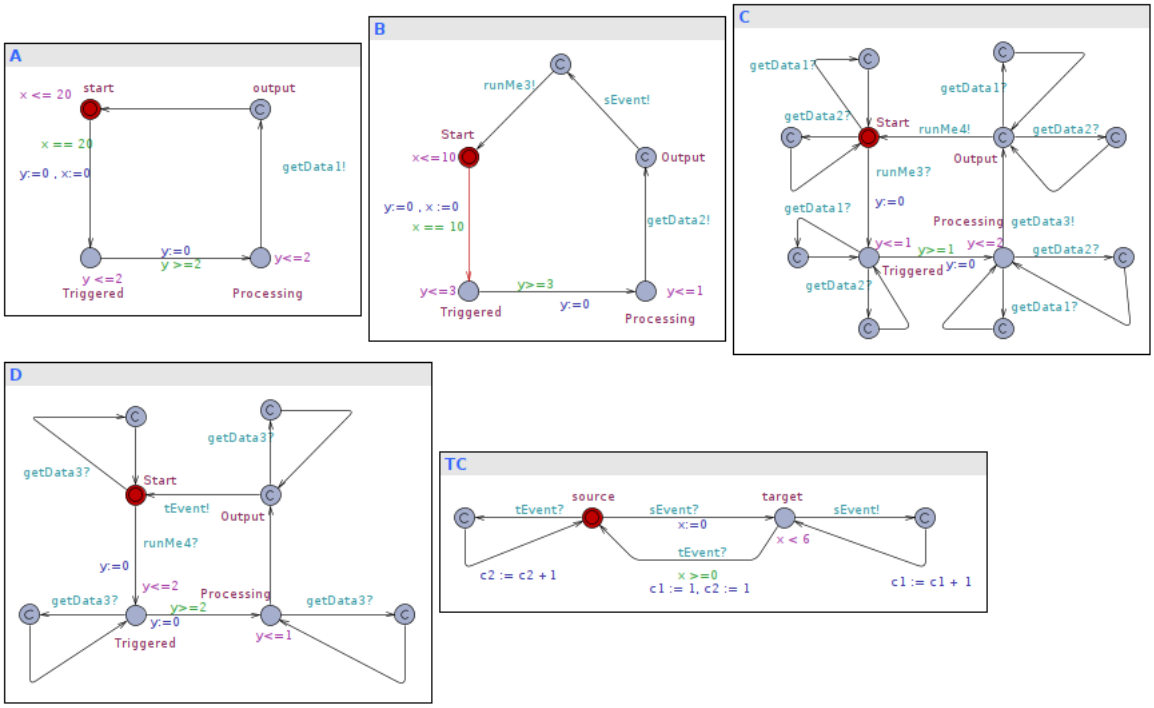

Figure 7: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following three CTL formulas

$$\forall\Box(not\ Deadlock)$$

$$\forall\Box(Target \Rightarrow \forall\Diamond Source)$$

$$\forall\Box(TC.target \Rightarrow TC.c1 == TC.c2)$$

are needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[0, 6]$ as illustrated in figure 8.

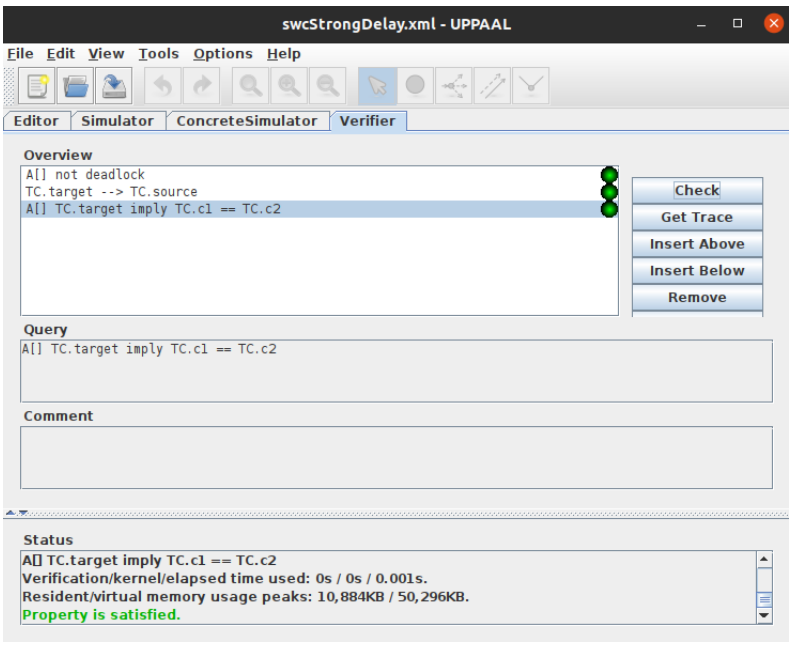

Figure 8: Results of Model Checking

Order Constraint

An OrderConstraint forces order limits between the occurrences of the source and target event. This constraint is a special case of the Strong Delay constraint and is equivalent to it if the following variations hold

- $c.lower$ is set to zero
- $c.upper$ is set to infinity
- s and t, respectively, cannot coincide.

A system behavior satisfies an OrderConstraint c if and only if $c.source$ and $c.target$ have the same number of occurrences, and for each index i, if there is an i^{th} occurrence of $c.source$ at time x, there is also an i^{th} occurrence of $c.target$ at time y such that

$$x < y$$

The semantics description of order constraint based on logic is declared as follows

$$|source| = |target| \wedge \forall i : \forall x : x = source(i) \Rightarrow \exists y : y = target(i) \wedge x < y$$

Unfortunately, TCTL is unable to specify this kind of property, as it does not support indexed events. To show how the Order constraint is verified on a simple ECU network by UPPAAL, we have used the example in figure 6.

Figure 9 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Order constraint as an observer timed automaton.

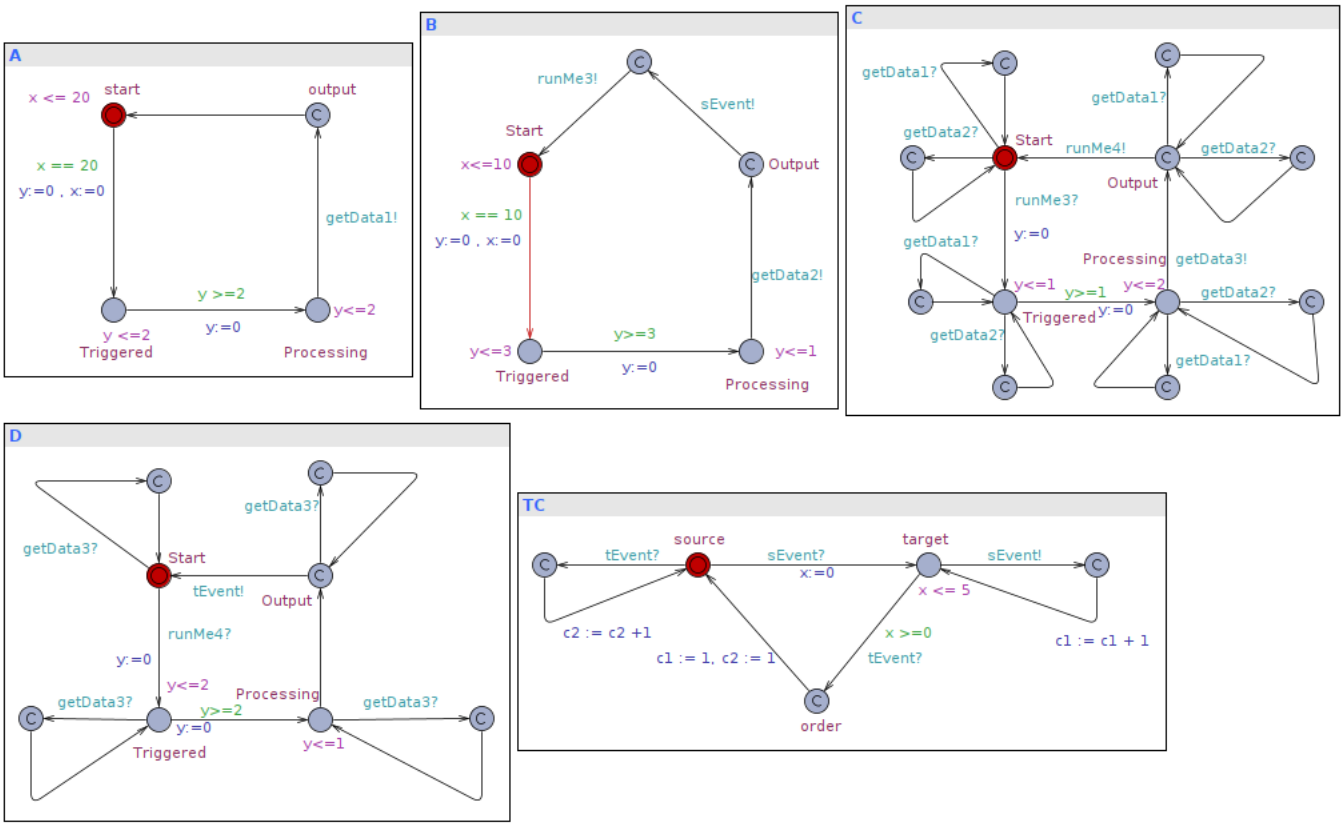

Figure 9: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following two CTL for-

mulas

$$\forall\Box(not\ Deadlock)$$

$$\forall\Box(Target \Rightarrow \forall\Diamond Source)$$

$$\forall\Box(TC.target \Rightarrow TC.c1 == TC.c2)$$

$$\forall\Box(TC.order \Rightarrow TC.x > 0)$$

are needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied as illustrated in figure 10.

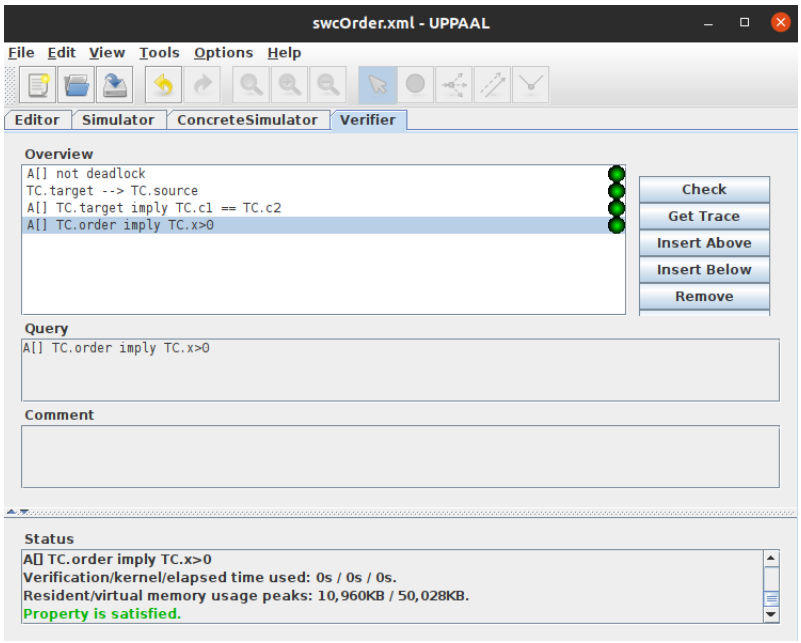

Figure 10: Results of Model Checking

Reaction Constraint

A ReactionConstraint implies how long after the occurrence of a stimulus a corresponding response must occur in an event chain. This constraint can only be applied to event chains and not to individual events. This constraint can be specified on an event chain, an event chain segment, or a distributed event chain. In the multi-rate event chains, multiple response occurrences due to each consecutive stimulus occurrence are differentiated using colors. To satisfy this constraint, the earliest occurrence of the response with the same color

as that of the stimulus must take place within the limits specified by this constraint.

A system behavior satisfies a ReactionConstraint c if and only if for each occurrence x in $c.scope.stimulus$, there is an occurrence y in $c.scope.response$ such that

$$y.color = x.color$$

and y is minimal in $c.scope.response$ with that color and

$$c.minimum \leq y–x \leq c.maximum$$

The semantics description of reaction constraint based on logic is declared as follows

$$\forall x \in scope.stimulus : \exists y \in scope.response :$$

$$x.color = y.color \wedge (\forall y' \in scope.response : y'.color = y.color \Rightarrow y \leq y')$$

$$\wedge\, minimum \leq y - x \leq maximum$$

By means to understood how an Event sequence satisfies a Reaction constraint, you can consider figure 11 as an example.

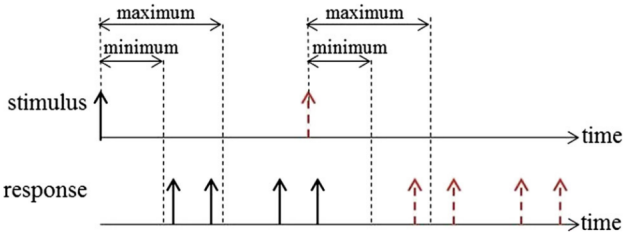

Figure 11: Event sequence satisfying a Reaction constraint

To show how the reaction constraint is verified on a simple ECU network by UPPAAL, figure 12 represents an example.

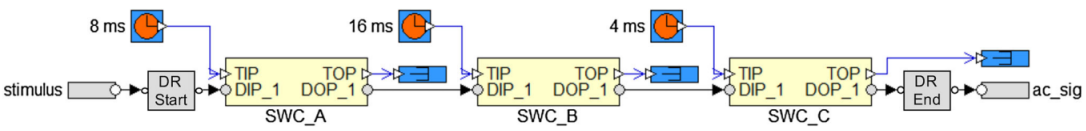

Figure 12: a simple ECU network

In this example, we have a network consisting of 3 ECUs, and each ECU is modeled as an independent software circuit. As shown in figure 12, SWC_A, SWC_B, SWC_C are periodic components and are triggered periodically.

Figure 13 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled reaction constraint as an observer timed automaton.

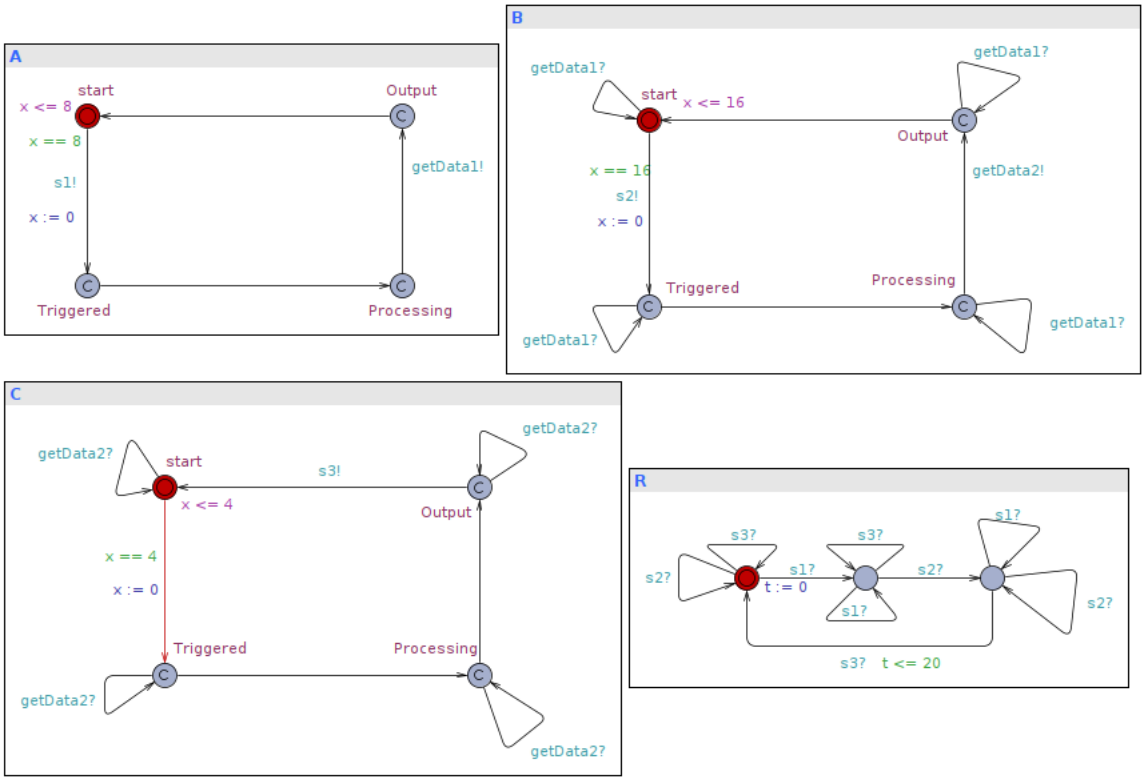

Figure 13: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 20]$ as illustrated in figure 14.

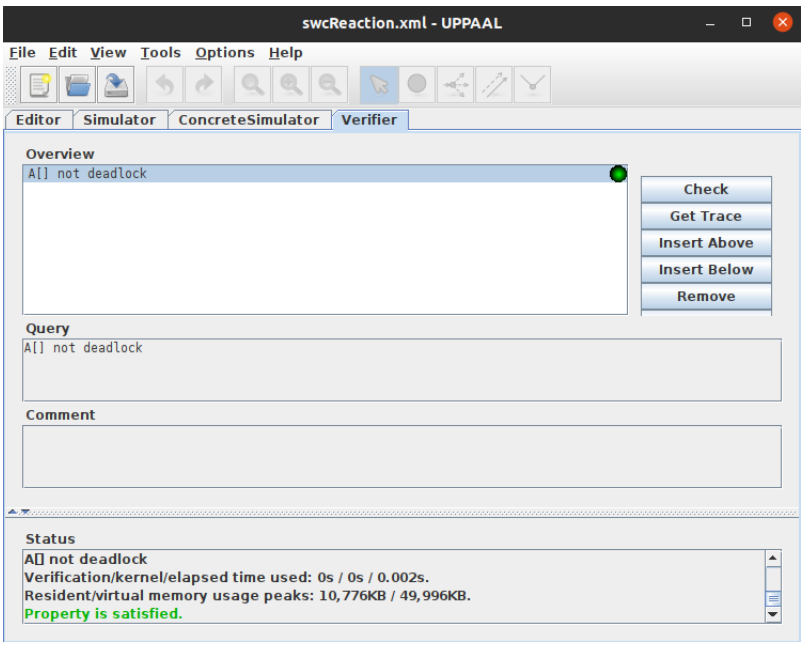

Figure 14: Results of Model Checking

Age Constraint

An AgeConstraint implies how long before each response a corresponding stimulus must have occurred in an event chain. This constraint can only be applied to event chains and not to individual events. This constraint can be specified on an event chain, an event chain segment, or a distributed event chain. In order to satisfy this constraint, the latest occurrence of the stimulus with the same color as that of the response must lie within the limits specified by this constraint. A system behavior satisfies an AgeConstraint c if and only if for each occurrence y in $c.scope.response$, there is an occurrence x in $c.scope.stimulus$ such that

$$x.color = y.color$$

and x is maximal in $c.scope.stimulus$ with that color and

$$c.minimum \leq y – x \leq c.maximum$$

The semantics description of age constraint based on logic is declared as follows

$$\forall y \in scope.response : \exists x \in scope.stimulus : x.color = y.color$$

$$\wedge (\forall x' \in scope.stimulus : x'.color = x.color \Rightarrow x' \leq x)$$

$$\wedge minimum \leq y - x \leq maximum$$

By means to understood how an Event sequence satisfies an Age constraint, you can consider figure 15 as an example.

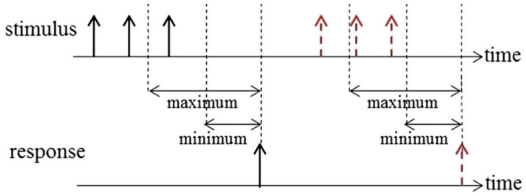

Figure 15: Event sequence satisfying an Age constraint

If the roles of stimulus and response are swapped, and the time bounds negated, a ReactionConstraint is obtained. The verification of this constraint is skipped because of its similarity to the Reaction Constraint.

Repetition Constraint

A RepetitionConstraint defines limits on the distribution of the occurrences of a single event, including the allowance for jitter. Jitter represents the maximum variation in time with which the event can be delayed. The span attribute associated with this constraint determines which repeated occurrence will be constrained. A system behavior satisfies a RepetitionConstraint c if and only if the following two are simultaneously satisfied for each subsequence X of $c.event$:

- if X contains $span + 1$ occurrences, then d is the distance between the outer- and inner-most occurrences in X and

$$c.lower \leq d \leq c.upper$$

- for each index i, if there is an i_{th} occurrence of X at time s, there also is an i_{th} occur-

rence of $c.event$ at time t such that

$$0 \leq (t–s) \leq c.jitter$$

If the span attribute is equal to one, jitter is equal to zero and the upper attribute is equal to the lower attribute, then the behavior becomes strictly periodic. By means to understood how an Event sequence satisfies a Repetition constraint, you can consider figure 16 as an example.

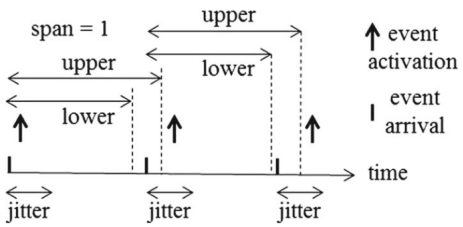

Figure 16: Event sequence satisfying a Repetition constraint

To show how the Repetition constraint is verified on a simple ECU network by UPPAAL, figure 17 represents an example. In this example, we have a single ECUs which is modeled as

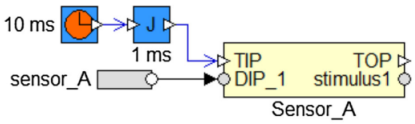

Figure 17: a simple ECU network

an independent software circuit. As shown in figure 17, SWC_A is periodic component and is triggered periodically.

Figure 18 illustrates the result of modeling the ECU as Timed Automaton with $span = 3$. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled repetition constraint as an observer timed automaton.

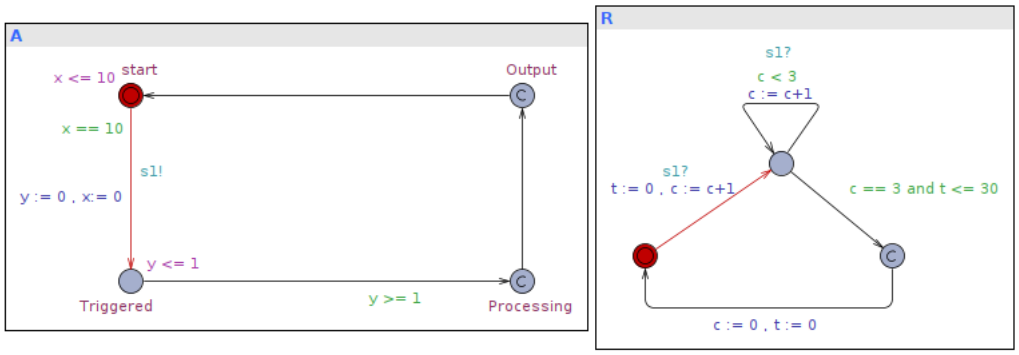

Figure 18: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 30]$ as illustrated in figure 19.

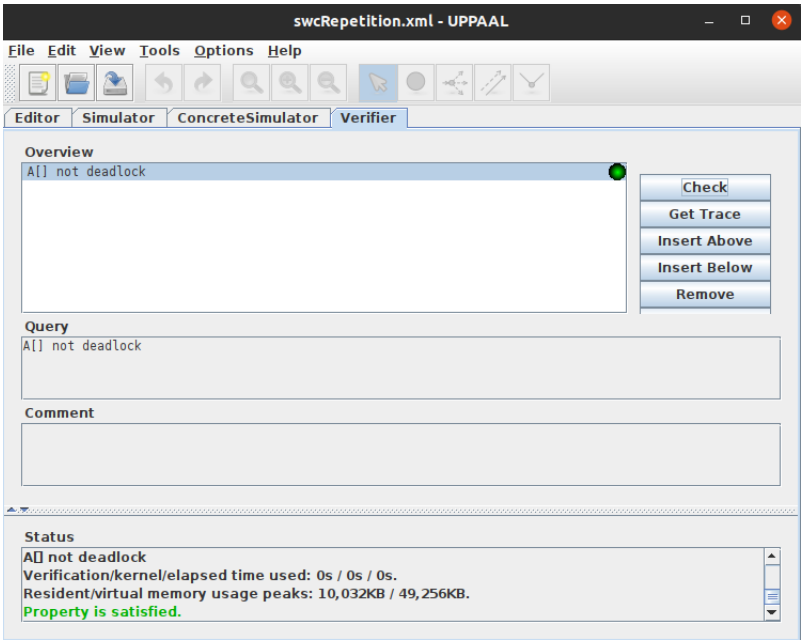

Figure 19: Results of Model Checking

Repeat Constraint

A RepeatConstraint defines the repeated distribution of the occurrences of a single event. This constraint is a special case of the Repetition constraint that does not experience any jitter. A system behavior satisfies a RepeatConstraint c if and only if for each subsequence X of $c.event$, if X contains $span+1$ occurrences then e is the distance between the outermost occurrences in X and

$$c.lower \leq e \leq c.upper$$

The semantics description of repeat constraint based on logic is declared as follows

$$\forall X \leq \text{event} : |X| = \text{span} + 1 \Rightarrow \text{lower} \leq \lambda([X]) \leq \text{upper}$$

where $\lambda(X)$ is the total length of all continuous intervals in X, such that if X contains all occurrences between times 3 and 5, as well as between times 6 and 8, $\lambda(X)$ is 4.
To show how the Repeat constraint is verified on a simple ECU network by UPPAAL, we have used the example in figure 17 but without jitter parameter.

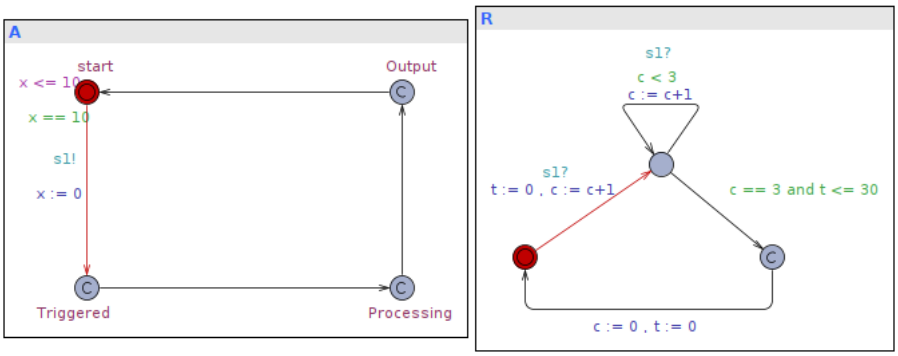

Figure 20: Timed Automaton models of ECUs

Figure 20 illustrates the result of modeling the ECU as Timed Automaton with $span = 3$. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled repeat constraint as an observer timed automaton.
In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 30]$ as illustrated in figure 21.

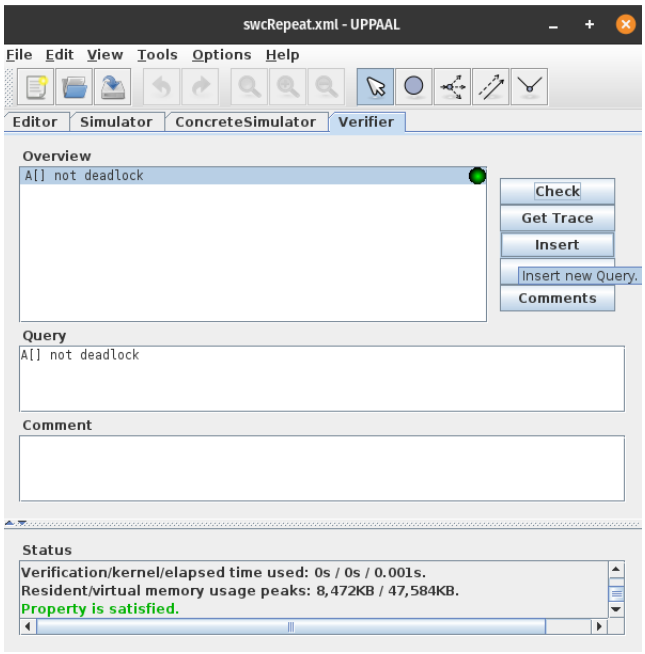

Figure 21: Results of Model Checking

Sporadic Constraint

A SporadicConstraint describes an event that occurs sporadically. This constraint is a special type of the Repetition constraint whose span is equal to 1. Moreover, any two subsequent activations of the event in this constraint must be separated by the minimum inter-arrival time (MIT). A system behavior satisfies a SporadicConstraint c if and only if the same system behavior concurrently satisfies

- RepetitionConstraint {event = c.event,lower = c.lower,upper = c.upper,jitter = c.jitter}
- RepeatConstraint {event = c.event,lower = c.minimum}

By means to understood how an Event sequence satisfies a Sporadic constraint, you can consider figure 22 as an example.

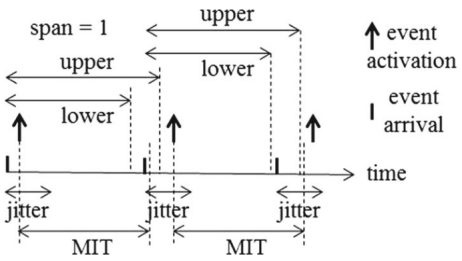

Figure 22: Event sequence satisfying an Sporadic constraint

To show how the Sporadic constraint is verified on a simple ECU network by UPPAAL, we have used the example in figure 17. Figure 23 illustrates the result of modeling the ECU as Timed Automaton with $span = 1$ and $MIT = 10$. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Sporadic constraint as an observer timed automaton. In order to model check our desired constraint on the system,

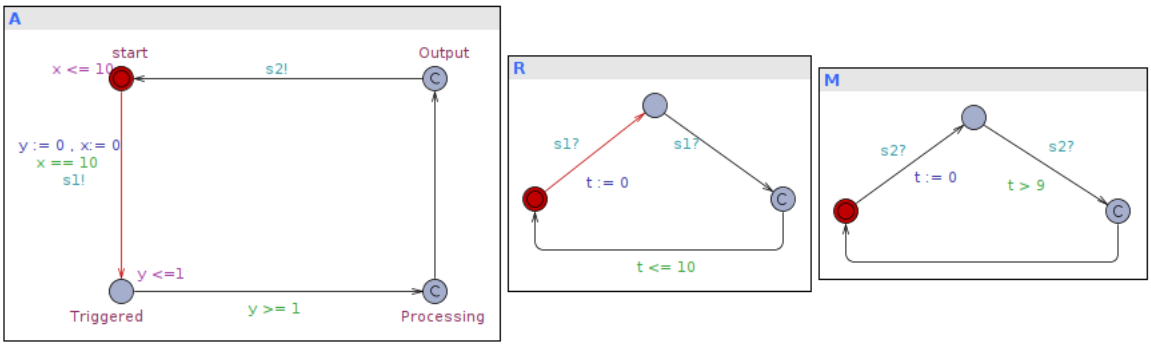

Figure 23: Timed Automaton models of ECUs

the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 10]$ and with MIT interval of $[0, 10]$ as illustrated in figure 24.

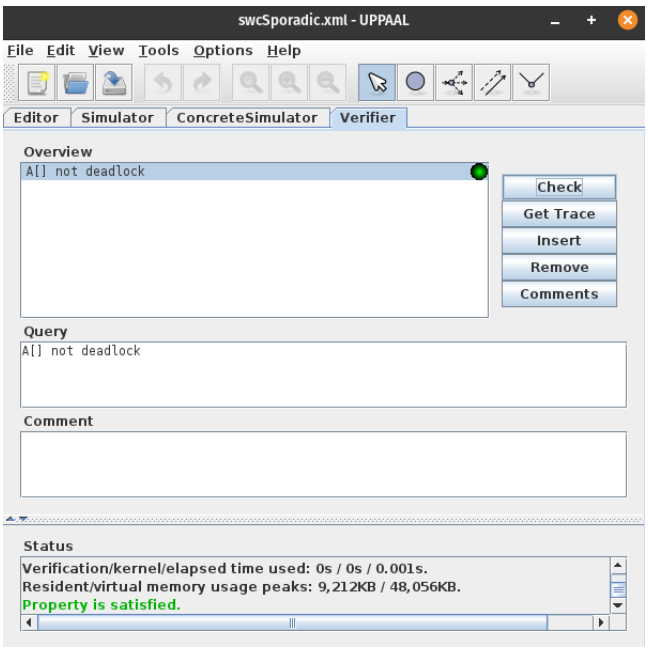

Figure 24: Results of Model Checking

Burst Constraint

A BurstConstraint describes an event that occurs in semi-regular bursts. This constraint is a special type of the Sporadic constraint with the following extensions.

- There is no allowance for jitter
- There is a maximum number of occurrences of the event, denoted by MaxOccurrences, in an interval. The size of the interval is denoted by length.
- Two subsequent activations in the interval must be separated by the minimum inter-arrival time (MIT).

By means to understood how an Event sequence satisfies a Burst constraint, you can consider figure 25 as an example.

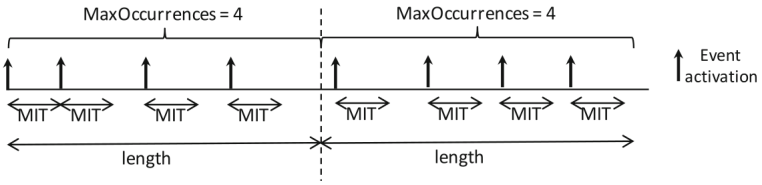

Figure 25: Event sequence satisfying an Burst constraint

To show how the Burst constraint is verified on a simple ECU network by UPPAAL, we have used the example in figure 17. Figure 26 illustrates the result of modeling the ECU as Timed Automaton with $MIT = 10$ and $MaxOccurrences = 4$ in each 30 unit interval. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Burst constraint as an observer timed automaton.

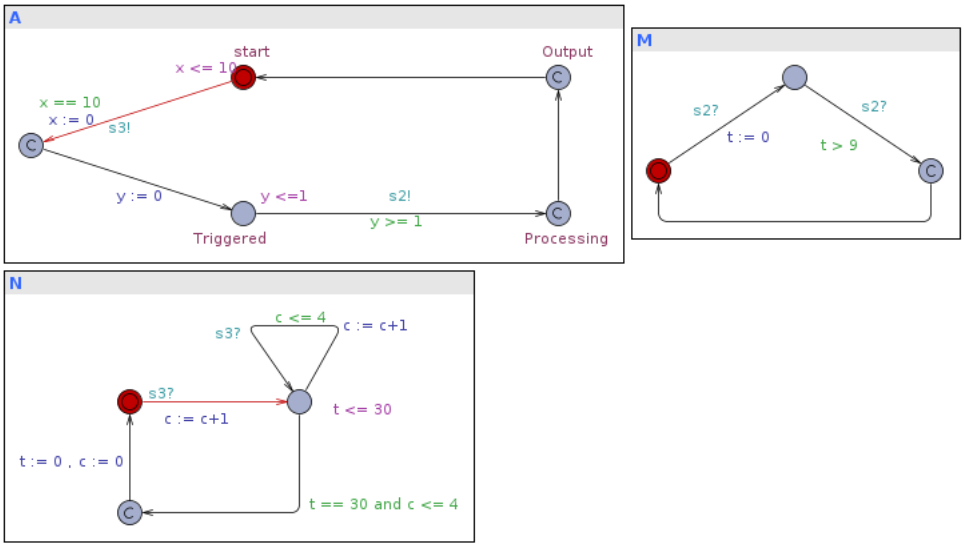

Figure 26: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 10]$ and with MIT interval of $[0, 10]$ as illustrated in figure 27

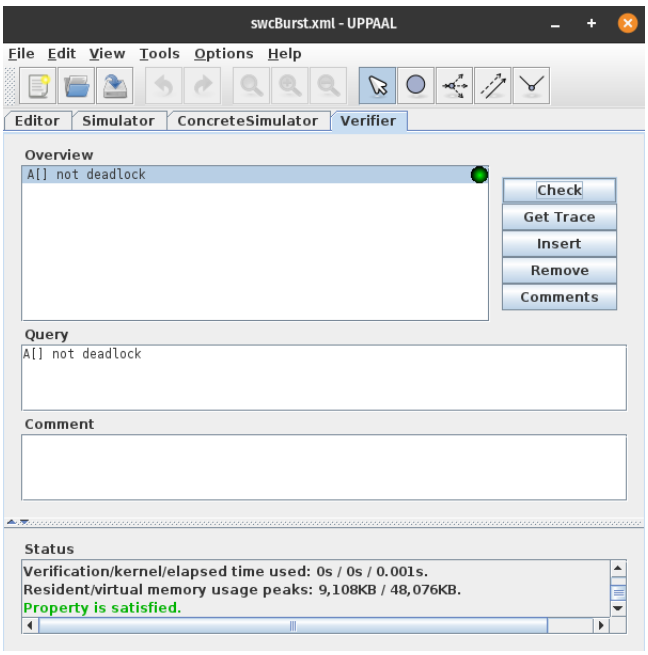

Figure 27: Results of Model Checking

Periodic Constraint

This constraint defines limits on the occurrence of a periodic event. This constraint is a special type of Sporadic constraint whose lower and upper attributes are equal. These attributes are assigned the value of the period. By means to understood how an Event sequence satisfies a Periodic constraint, you can consider figure 28 as an example. The veri-

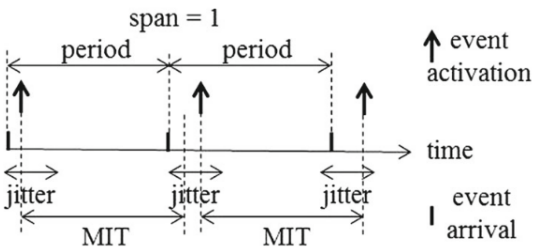

Figure 28: Event sequence satisfying an Periodic constraint

fication of this constraint is skipped because of its similarity to the Sporadic Constraint.

Pattern Constraint

This constraint defines limits on the occurrences of an event that follows a certain pattern relative to some periodic temporal points. A system behavior satisfies the specified PatternConstraint c if and only if there is a set of times X such that the same system behavior simultaneously satisfies the following conditions.

- PeriodicConstraint with its period equal to c.$period$.
- For each $c.offset$ index i, there is an occurrence x_i of X such that

$$c.offset \leq x_i \leq (c.offset_i + c.jitter)$$

- If X contains two occurrences, then d is the distance between the outer- and innermost occurrences in X and

$$c.minimum \leq d$$

Note that x_i represents all the occurrences of the event within each period. By means to understood how an Event sequence satisfies a Pattern constraint, you can consider figure 29 as an example. To show how the Pattern constraint is verified on a simple ECU network

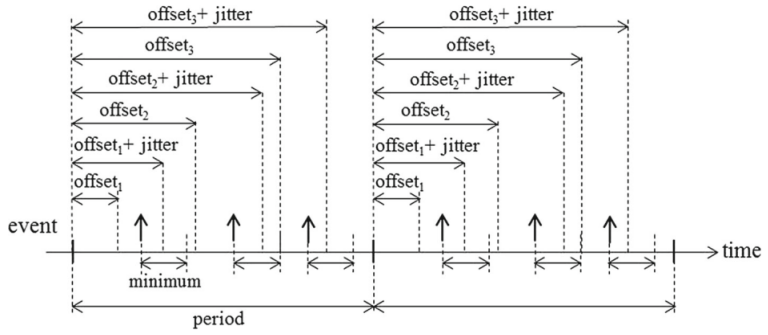

Figure 29: Event sequence satisfying an Pattern constraint

by UPPAAL, we have used the example in figure 17. Figure 30 illustrates the result of modeling the ECU as Timed Automaton with $MIT = 10$, $jitter = 5$, $period = 45$, $offset_1 = 10$, $offset_2 = 20$, and $offset_3 = 30$. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Pattern constraint as an observer timed automaton.

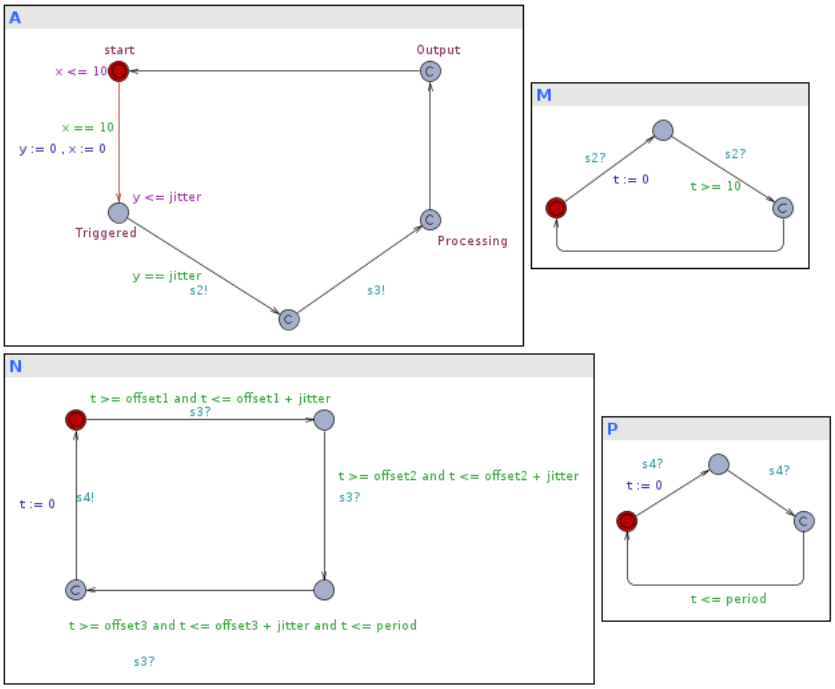

Figure 30: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied in the interval of $[0, 45]$ and with MIT interval of $[0, 10]$ as illustrated in figure 31

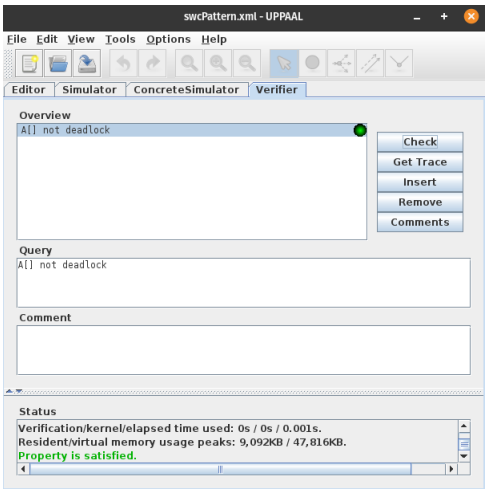

Figure 31: Results of Model Checking

Arbitrary Constraint

This constraint defines limits on an event that occurs irregularly. The constraint contains a set of pairs consisting of a minimum inter-arrival time (denoted by min) and a maximum inter-arrival time (denoted by max). A system behavior satisfies the specified ArbitraryConstraint C if and olny if for each $c.min$ index i, the same system behavior satisfies, for each subsequence X of $c.event$, if X contains $i+1$ occurrences then d is the distance between the outer- and inner-most occurrences in X and

$$c.min_i \leq d \leq c.max_i$$

By means to understood how an Event sequence satisfies a Arbitrary constraint, you can consider figure 32 as an example.

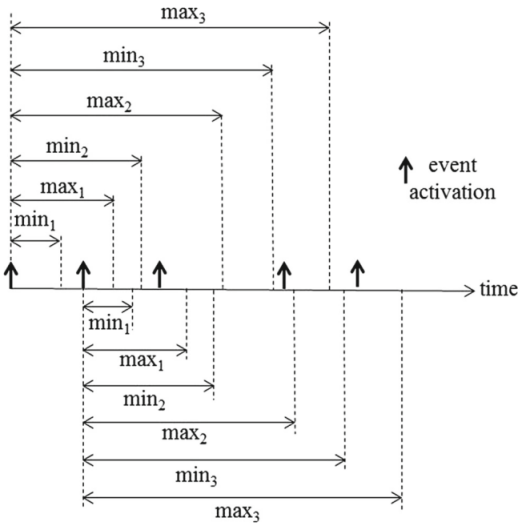

Figure 32: Event sequence satisfying an Arbitrary constraint

To show how the Arbitrary constraint is verified on a simple ECU network by UPPAAL, we have used the example in figure 17. Figure 33 illustrates the result of modeling the ECU as Timed Automaton with $min_1 = 0$, $min_2 = 10$, $min_3 = 20$, $max_1 = 13$, $max_2 = 26$, and $max_3 = 39$. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Arbitrary constraint as an observer timed automaton.

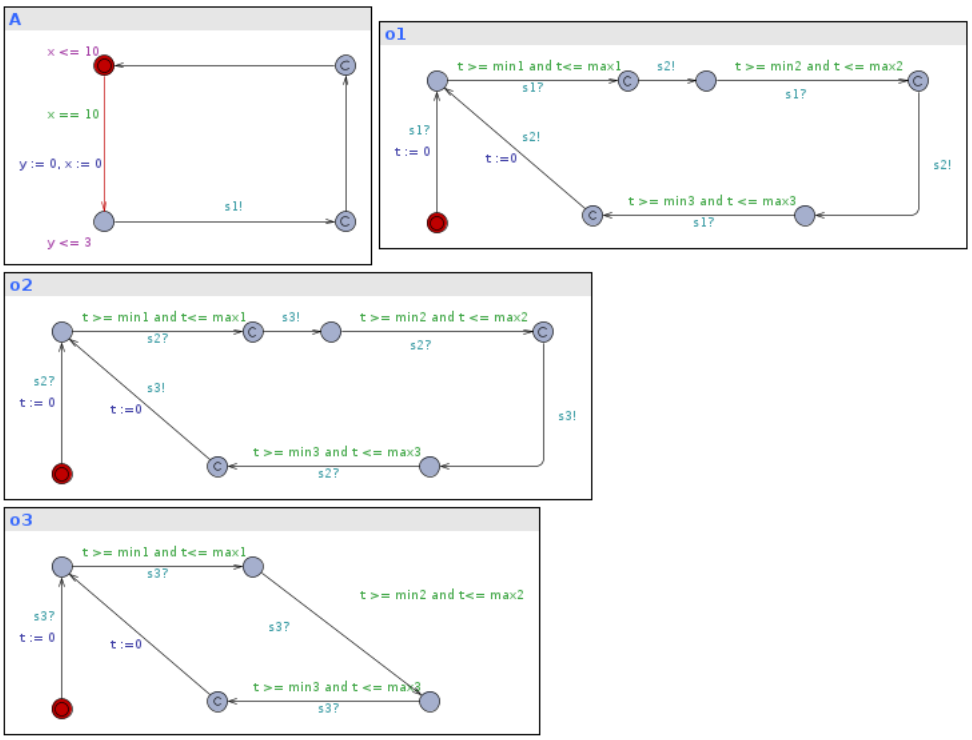

Figure 33: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former property, we find that the constraint is satisfied as illustrated in figure 34

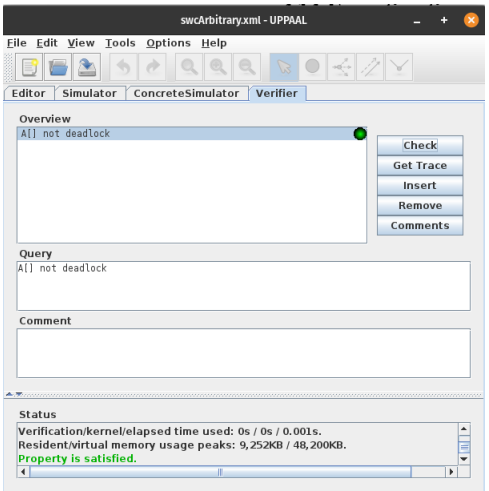

Figure 34: Results of Model Checking

Execution time Constraint

This constraint defines limits on the time between activation and completion of the execution of a function (executable entity). However, the intervals, when the execution of the function is interrupted due to preemptions and blocking, are not considered in this constraint. A system behavior satisfies the specified ExecutionTimeConstraint c if and only if for each occurrence x of the event $c.activate$, ET_i is the set of times between x and the next $c.completion$ while excluding the times due to $c.preemtion$ and $c.blocking$, and that

$$c.lower \leq \textit{sum of all continuous intervals in } ET_i \leq c.upper$$

By means to understood how an Event sequence satisfies a Execution time constraint, you can consider figure 35 as an example.

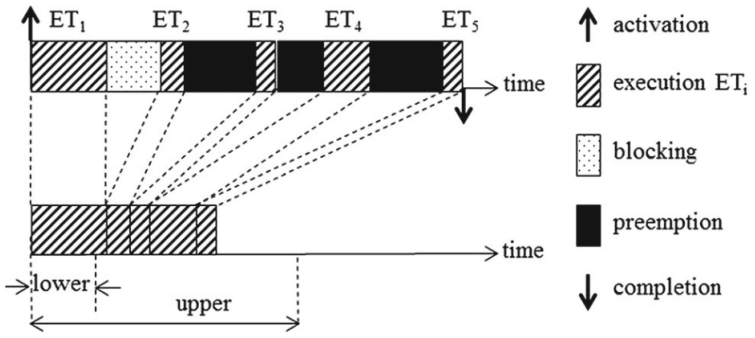

Figure 35: Event sequence satisfying an Arbitrary constraint

The verification of this constraint is skipped because of its simplicity.

Synchronization Constraint

This constraint defines limits on the closeness of the occurrences of a group of events. A system behavior satisfies the specified SynchronizationConstraint on a given set of events and given the occurrence of any event in this set, then the rest of the events in the set must occur at least once within a certain time window called tolerance. By means to understood how an Event sequence satisfies a Synchronization constraint, you can consider figure 36 as an example.

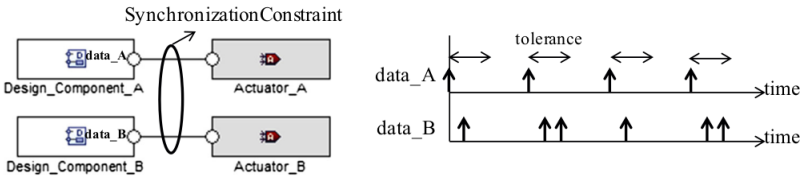

Figure 36: Event sequence satisfying an Synchronization constraint

In this constraint, more than one instance of the events may exist in a time window, provided the above conditions are met. Moreover, the windows may overlap and they may share occurrences of the events. To show how the Synchronization constraint is verified on a simple ECU network by UPPAAL, figure 37 represents an example.

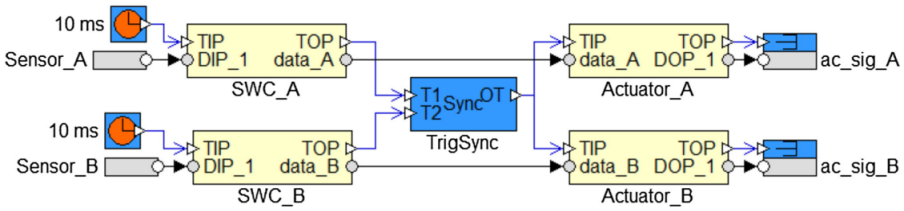

Figure 37: a simple ECU network

In this example, we have a network consisting of 4 ECUs, and each ECU is modeled as an independent software circuit. As shown in figure 37, SWC_A and SWC_B are both periodic components and are triggered periodically. In contrast, SWC_C and SWC_D are both sporadic components and are triggered by another component's event. Components C and D are triggered by components A and B. now we have two ways to describe this situation. First is that the TOP of components A and B both follows the AND semantics. Second is the case that they both follow the OR semantics. We have covered both senarios in the following. Figure 38 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them by considering the AND semantics with $tolerance = [0, 3]$. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Synchronization constraint as an observer timed automaton.

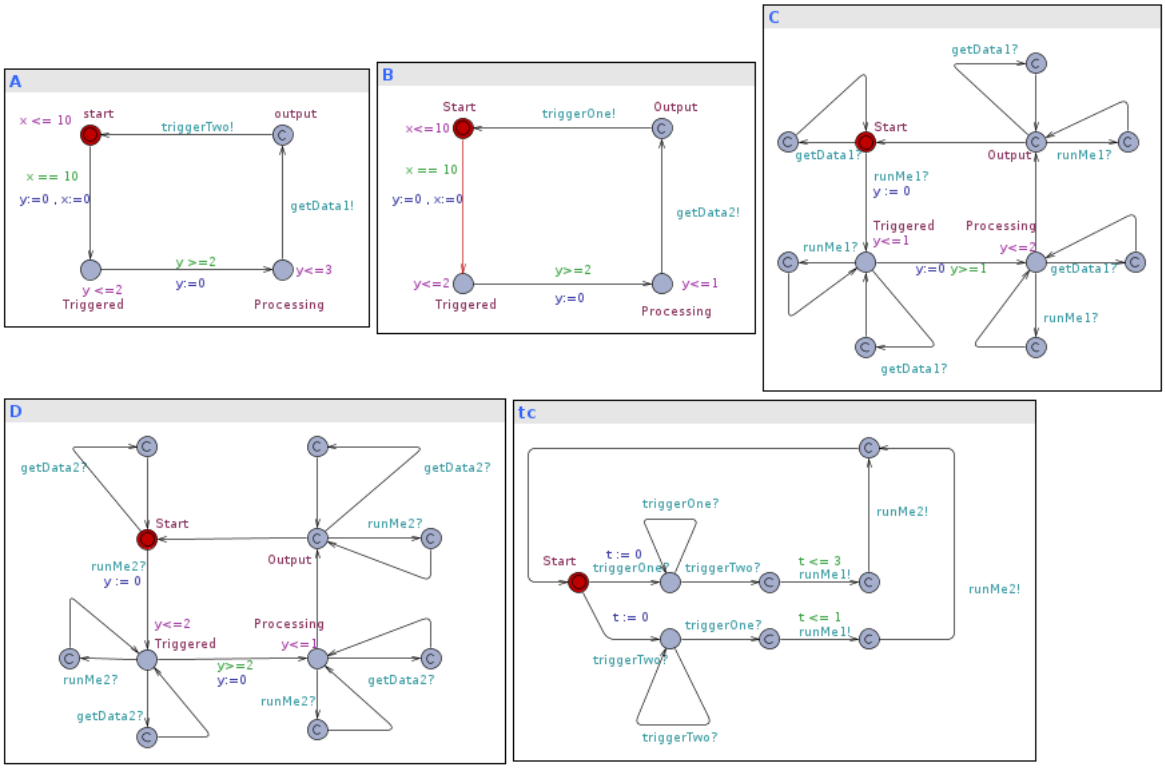

Figure 38: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[1, 3]$ as illustrated in figure 39.

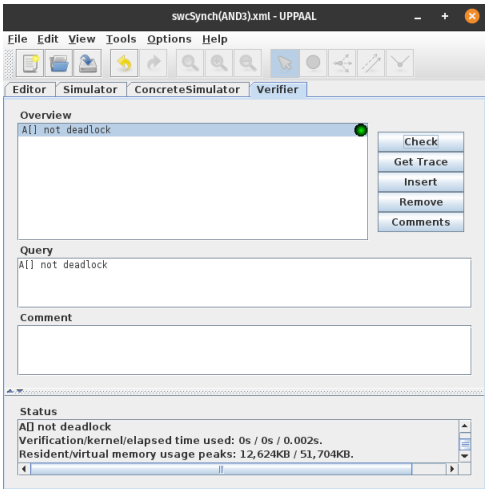

Figure 39: Results of Model Checking

Figure 40 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them by considering the OR semantics with $tolerance = [0, 15]$. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Synchronization constraint as an observer timed automaton.

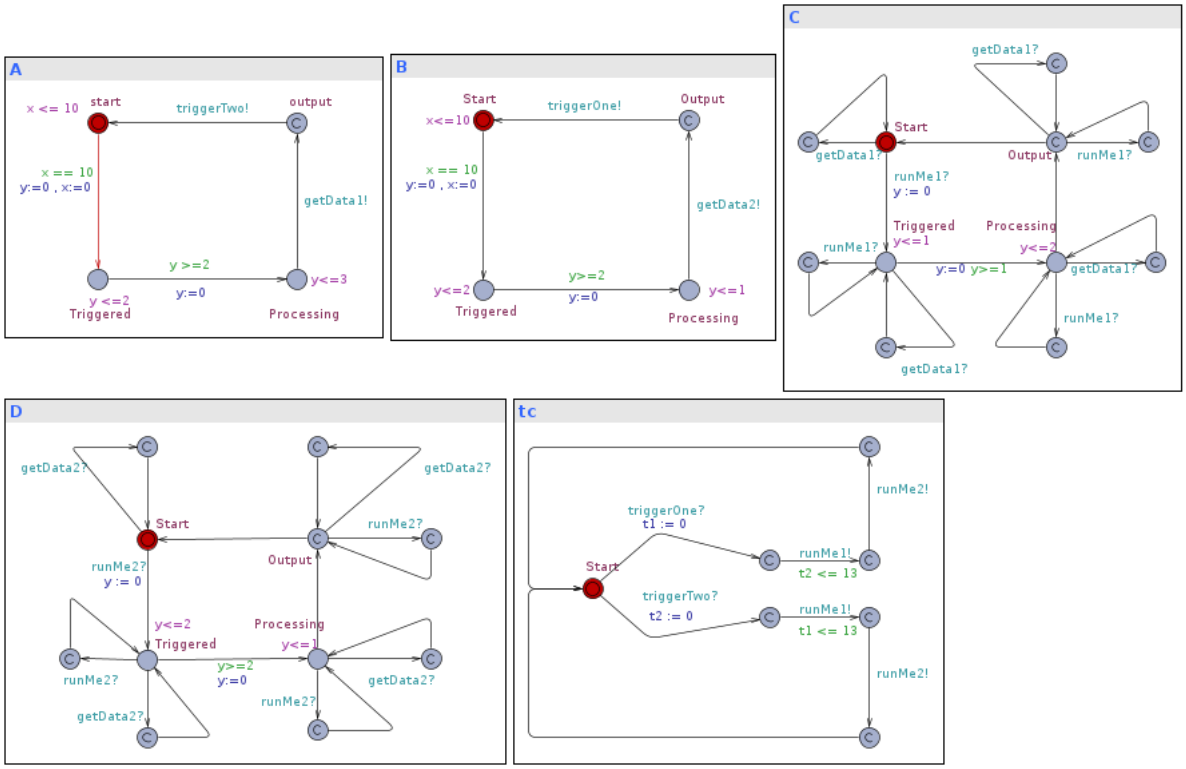

Figure 40: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[0, 15]$ as illustrated in figure 41.

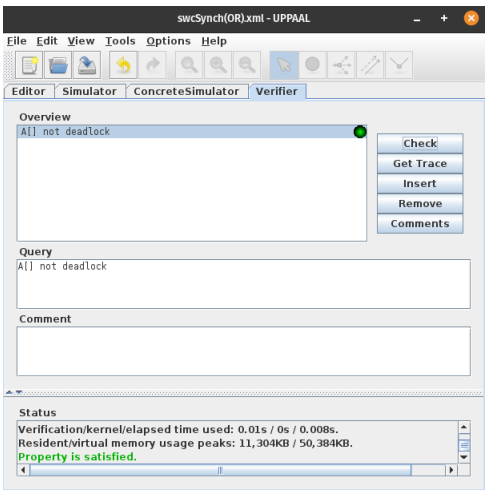

Figure 41: Results of Model Checking

Strong Synchronization Constraint

This constraint defines limits on the closeness of the occurrences of a group of events. The semantics of the StrongSynchronizationConstraint differs from the semantics of the SynchronizationConstraint in a way that the occurrences of the events in a window must have same indices. Therefore, at most one instance of the events can exist in the time window. Moreover, the windows cannot overlap and they may share occurrences of the events. By means to understood how an Event sequence satisfies a Strong Synchronization constraint, you can consider figure 42 as an example. To show how the Strong Synchronization con-

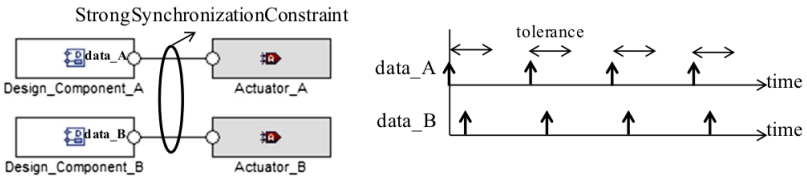

Figure 42: Event sequence satisfying an Strong Synchronization constraint

straint is verified on a simple ECU network by UPPAAL, we have used the example in figure 37. In this case we only consider the AND semantics.

Figure 42 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them by considering the AND semantics with $tolerance = [0, 3]$. More information is augmented to each ECU, e.g., jitter and computation delay, to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Strong Synchronization constraint as an observer timed automaton.

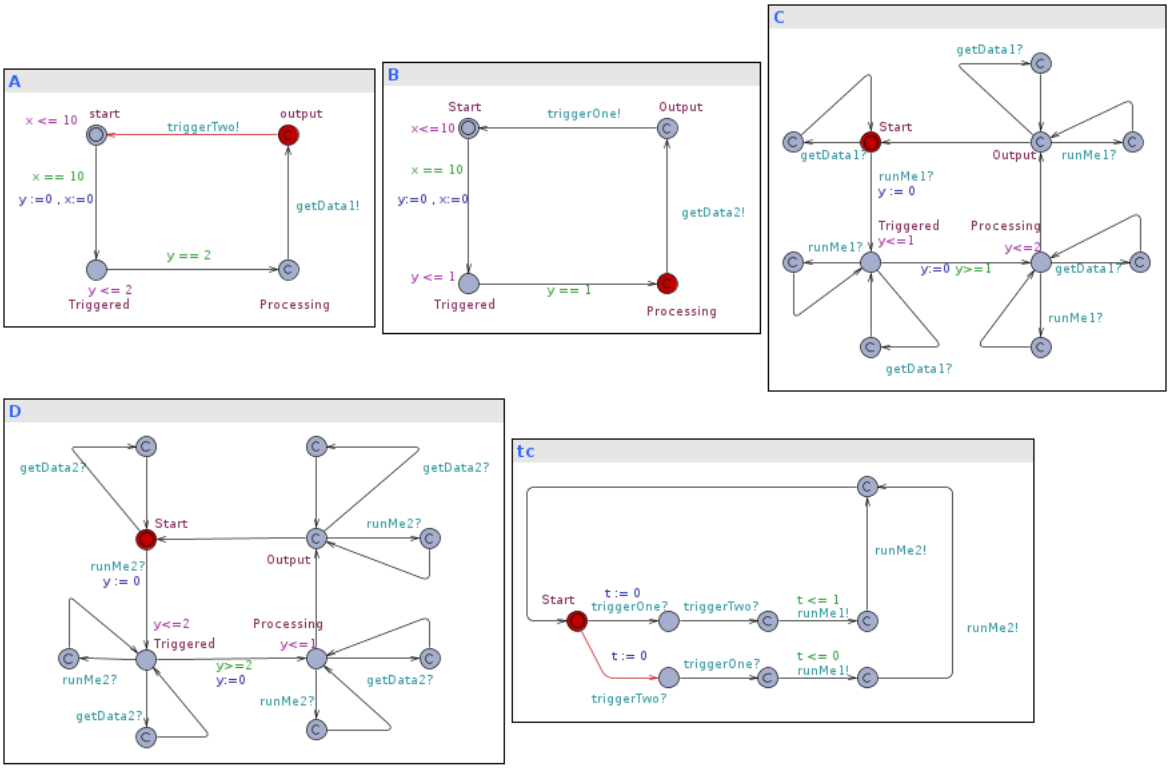

Figure 43: Timed Automaton models of ECUs

n order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[0, 1]$ as illustrated in figure 44.

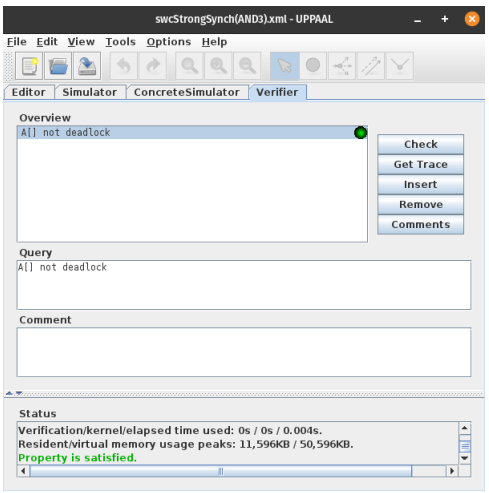

Figure 44: Results of Model Checking

Output Synchronization Constraint

This constraint defines limits on the closeness of the occurrences of responses to a certain stimulus. Basically, the constraint defines how far apart the responses to a certain stimulus can occur. This constraint differs from the SynchronizationConstraint in a way that it can only be applied to a set of event chains such that there are multiple responses to a single stimulus.

A system behavior satisfies an OutputSynchronizationConstraint c if and only if for each occurrence x in $c.scope1.stimulus$, there is a time t such that for each $c.scope$ index i, there is an occurrence y in $c.scopei.response$ such that

$$y.color = x.color$$

and y is minimal in $c.scopei.response$ with that color and

$$0 \leq y - t \leq c.tolerance$$

The semantics description of Output Synchronization constraint based on logic is declared as follows

$$\forall x \in scope_1.stimulus : \exists t : \forall i : \exists y \in scope_i.response :$$

$$x.color = y.color \wedge (\forall y' \in scope_{\cdot}.response : y'.color = y.color \Rightarrow y \leq y')$$

By means to understood how an Event sequence satisfies a Output Synchronization constraint, you can consider figure 45 as an example. To show how the Output Synchronization

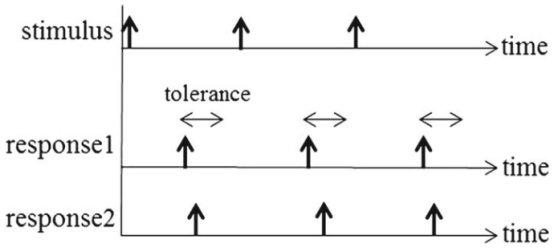

Figure 45: Event sequence satisfying an Output Synchronization constraint

constraint is verified on a simple ECU network by UPPAAL, figure 46 represents an example. In this example, we have a network consisting of 3 ECUs, and each ECU is modeled as an

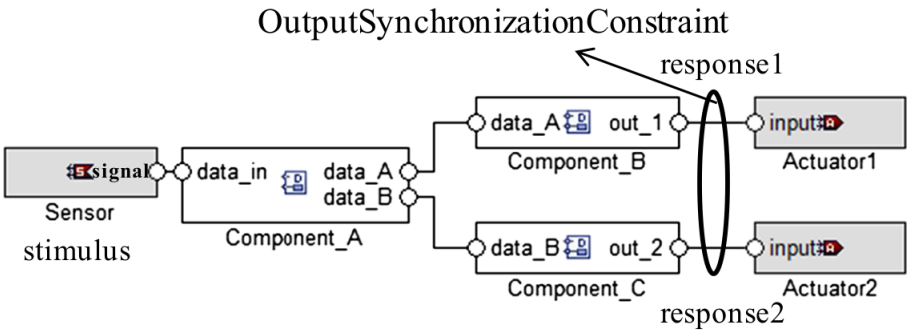

Figure 46: a simple ECU network

independent software circuit. As shown in figure 46, SWC_A is periodic components and is triggered periodically. In contrast, SWC_B and SWC_C are both sporadic components and are triggered by another component's event.

Figure 47 illustrates the result of modeling each ECU as Timed Automaton and modeling the system by composing them with $tolerance = [0, 3]$. More information is augmented to each ECU, e.g., jitter to make this example closer to a real-world use case. Due to the limitations of UPPAAL in specifying timing constraints with TCTL logic, we modeled Output Synchronization constraint as an observer timed automaton.

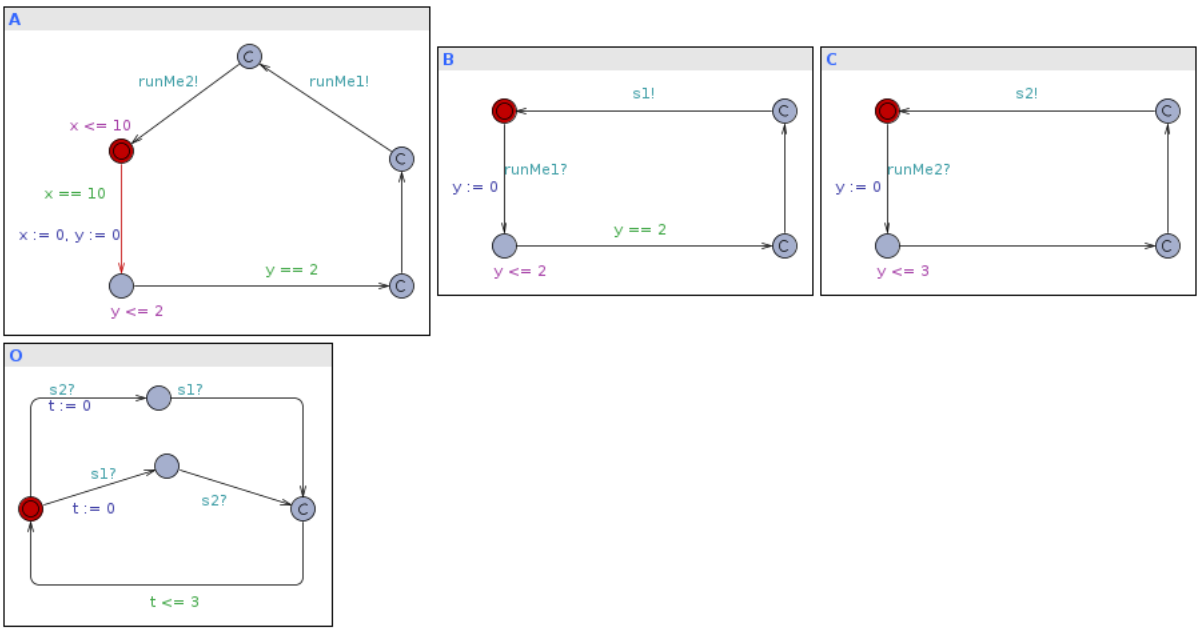

Figure 47: Timed Automaton models of ECUs

In order to model check our desired constraint on the system, the following CTL formula

$$\forall\Box(not\ Deadlock)$$

is needed to be verified by UPPAAL. By checking the former properties, we find that the constraint is satisfied in the interval of $[0, 3]$ as illustrated in figure 39.

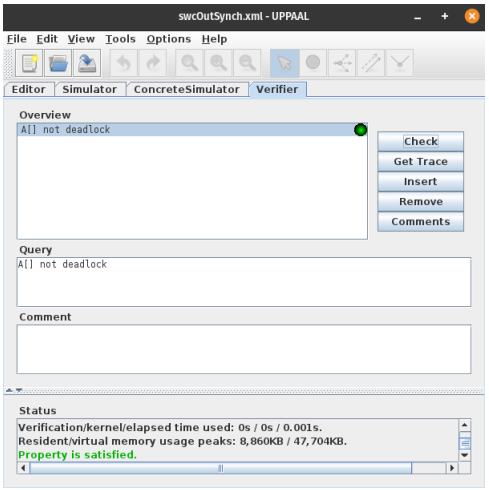

Figure 48: Results of Model Checking

Input Synchronization Constraint

This constraint constrains the closeness of the occurrences of stimuli corresponding to a certain response. Basically, the constraint defines how far apart the stimuli corresponding to a certain response can occur. This constraint differs from the Synchronization constraint in a way that it can only be applied to a set of event chains such that there are multiple stimuli and a single corresponding response.

A system behavior satisfies an InputSynchronizationConstraint c if and only if for each occurrence y in $c.scope1.response$, there is a time t such that for each $c.scope$ index i, there is an occurrence x in $c.scopei.stimulus$ such that

$$y.color = x.color$$

and x is maximal in $c.scopei.stimulus$ with that color and

$$0 \leq x - t \leq c.tolerance$$

The semantics description of Input Synchronization constraint based on logic is declared as follows

$$\forall y \in scope_1.response : \exists t : \forall i : \exists x \in scope_i.stimulus :$$

$$x.color = y.color \wedge (\forall x' \in scope_{\cdot}.stimulus : x'.color = x.color \Rightarrow x \leq x')$$

By means to understood how an Event sequence satisfies a Input Synchronization constraint, you can consider figure 49 as an example.

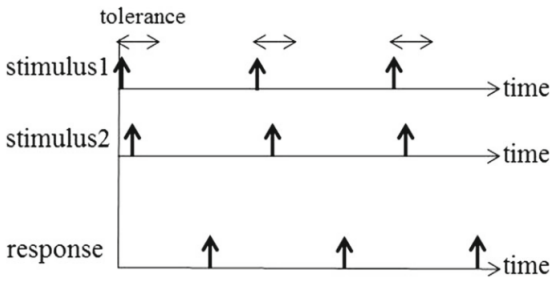

Figure 49: Event sequence satisfying an Input Synchronization constraint

The verification of this constraint is skipped because of its similarity to the Output Synchronization Constraint.

Comparison constraint

A ComparisonConstraint indicates that a certain ordering relation must exist between two timing expressions. This constraint is not a timing constraint. In fact, it is used to represent the comparison between the value of a specified constraint and the values of the variables that have arithmetic relations between/among them. For example, consider a distributed chain that consists of three sub-chains. Also assume that the delay of each sub-chain is calculated separately. The distributed chain is considered schedulable if the sum of the three delays is less than or equal to the Delay constraint specified on the distributed chain. The verification of this constraint is skipped because of its simplicity.