AMT 2.0: Qualitative and Quantitative Trace Analysis with Extended Signal Temporal Logic

Dejan Ničković[1], Olivier Lebeltel[2], Oded Maler[2], Thomas Ferrèrre[3], Dogan Ulus[2]

[1] Austrian Institute of Texhnology GmbH
[2] Verimag, CNRS / University of Grenoble-Alpes, France
[3] IST Austria

Abstract. We introduce in this paper AMT 2.0, a tool for qualitative and quantitative analysis of hybrid (mixed) signals that combines numerical values and discrete events. The evaluation of the signals is based on rich temporal specifications expressed in *extended Signal Temporal Logic* (xSTL), which combines Signal Temporal Logic (STL) with Timed Regular Expressions (TRE). The tool features qualitative monitoring (property satisfaction checking), trace diagnostics for explaining and justifying property violations and specification-driven measurement of quantitative features of the signal.

1 Introduction

Models of cyber-physical systems, such as automotive embedded controllers or analog and mixed-signal electronic circuits, are analyzed by extensive simulations that produce traces that admit real values and are often interpreted as continuous-time signals. To evaluate the system under design, these traces are inspected for satisfying some correctness requirements and are often subject to quantitative analysis based on recording some values in certain segments of the signal and performing some computation (summation, minimum) on them.

Over the past decade an extensive framework has been developed whose goal was to bring automated support for this tedious and error-prone task, centered around Signal Temporal Logic (STL) [15,16]. STL extends classical LTL in two directions: it uses predicates over real-values variables in addition to atomic propositions, and it is defined over dense continuous time accessed symbolically with timed modalities as in Metric Temoral Logic (MTL) [14]. This framework, which was initially accompanied by a rudimentary prototype tool [17], had a lot of reported applications in domains such as automotive, robotics, analog circuits, systems biology. It can be viewed as an extension of *runtime verification* toward cyber-physical hybrid systems. Interested readers may look at the recent survey in [6].

In this article we present AMT 2.0, a new version of the tool. The new version is much more mature in terms of software engineering aspects such as rigorous typing of signals and properties, introducing programming language features such as declaration and aliasing, improvement of the graphical editors, systematic software testing, etc. Furthermore, its functionality has been extended significantly by incorporating several new research results obtained over the last years:

1. We combine STL with a fragment of Timed Regular Expressions (TRE) [3,4], as a complementary formalism to express temporal patterns. The monitoring algorithm for our specification language xSTL thus obtained integrates into the tool the recent TRE pattern matching algorithm reported in [19].
2. We use the TRE formalism to define segments of the signal to which quantitative measurements should be applied. Thus we obtain a declarative measurement languages that does for the quantitative domain what formal specification languages do for correctness checking. The results, first reported in [11], are fully incorporated into the tool.
3. We implement the error diagnostics algorithm of [10] which accompanies the report on a property violation with a justification: a small sub-signal (temporal implicant) which is sufficient to imply the property violation and to convince the user in this fact.

With all these features we progress toward easing the task of designers coming to analyze a complex system based on simulations, providing them with an alternative to manual inspection or explicit programming of observers.

The rest of the paper is organized as follows. In Section 2 we present the xSTL specification language. Section 3 gives an overview of the tool and its main features. We illustrate the usage of AMT 2.0 in Section 4 with two examples. We present the related work in Section 5 and give concluding remarks in Section 6. The evaluation package for the tool that includes the executable JARs for the batch and the GUI versions of the tool, the tutorial and the examples can be found at `http://www-verimag.imag.fr/~maler/tacas18/amt-evaluation.zip`.

2 Extended Signal Temporal Logic

Extended Signal Temporal Logix (xSTL) essentially combines STL with a variant of TRE. In this section, we provide the mathematical definitions of the specification language.

We denote by P and X finite sets of *propositional* and *data* variables, such that $|P| = m$ and $|X| = n$. Data variables are defined over an arbitrary domain $\mathbb{D}$, typically the reals and the integers. We use the notation $w : \mathbb{T} \to \mathbb{D}^n \times \mathbb{B}^m$ to represent a multi-dimensional *signal* with $\mathbb{T} = [0, d) \subseteq \mathbb{R}$, We denote by w_p the *projection* of w on its component p. We denote by $\theta : \mathbb{D}^n \to \mathbb{B}$ a *predicate* that maps valuations of variables in X into $\{\mathsf{true}, \mathsf{false}\}$.

The syntax of an STL formula φ with both *future* and *past* temporal operators and interpreted over $X \cup P$ is defined by the grammar

$$\varphi := p \mid \theta(x_1, \ldots, x_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I \varphi_2 \mid \varphi_1 \,\mathcal{S}_I \varphi_2$$

where $p \in P$, $x_1, \ldots, x_n \in X$ and $I \subseteq \mathbb{R}^+$ is an interval. We use the *strict* semantics [1] for *until* and *since* temporal operators that allows us to specify instantaneous *rise*($\uparrow$) and *fall* ($\downarrow$) events using the rules $\uparrow \varphi \equiv (\neg\varphi \,\mathcal{S}\,\mathsf{true}) \vee (\varphi \,\mathcal{U}\,\mathsf{true})$ and $\downarrow \varphi \equiv (\varphi \,\mathcal{S}\,\mathsf{true}) \vee (\neg\varphi \,\mathcal{U}\,\mathsf{true})$. We derive other standard operators as follows: $\mathsf{true} \equiv p \vee \neg p$, $\mathsf{false} \equiv$

$\neg\mathsf{true}$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond_I \varphi \equiv \mathsf{true}\ \mathcal{U}_I\ \varphi$, $\diamondminus_I \varphi \equiv \mathsf{true}\ \mathcal{S}_I\ \varphi$, $\Box_I \varphi \equiv \neg\Diamond_I\neg\varphi$, and $\boxminus_I \varphi \equiv \neg\Diamond_I\neg\varphi$.

The semantics of an STL formula with respect to a signal w is described via the satisfiability relation $(w,t) \models \varphi$, indicating that the signal w satisfies φ at time point t, according to the following definition.

$$
\begin{array}{ll}
(w,t) \models p & \leftrightarrow w_p[t] = \mathsf{true} \\
(w,t) \models \theta(x_1,\dots,x_n) & \leftrightarrow \theta(w_{x_1}[t],\dots,w_{x_n}[t]) = \mathsf{true} \\
(w,t) \models \neg\varphi & \leftrightarrow (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \leftrightarrow (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \,\mathcal{U}_I \varphi_2 & \leftrightarrow \exists t' \in (t+I) \cap \mathbb{T} : (w,t') \models \varphi_2 \text{ and} \\
 & \quad \forall t < t'' < t', (w,t'') \models \varphi_1 \\
(w,t) \models \varphi_1 \,\mathcal{S}_I \varphi_2 & \leftrightarrow \exists t' \in (t-I) \cap \mathbb{T} : (w,t') \models \varphi_2 \text{ and} \\
 & \quad \forall t' < t'' < t, (w,t'') \models \varphi_1
\end{array}
$$

We now define a variant of TRE according to the following grammar:

$$r := \epsilon \mid p \mid \theta(x_1,\dots,x_n) \mid r_1 \cdot r_2 \mid r_1 \cup r_2 \mid r_1 \cap r_2 \mid r^* \mid \langle r \rangle_I \mid r_1 \,?\, r_2 \mid r_2 \,!\, r_2$$

where I is an interval of $\mathbb{R}_+$. The semantics of a timed regular expression r with respect to a signal w and times $t \le t'$ in $[0,d]$ is given in terms of a *match* relation $(w,t,t') \vDash r$, which indicates that the segment of w between t and t' matches the expression. This relation is defined inductively as follows:

$$
\begin{array}{lll}
(w,t,t') \vDash \epsilon & \leftrightarrow & t = t' \\
(w,t,t') \vDash p & \leftrightarrow & t < t' \text{ and } \forall t'' \in (t,t'),\ w_p[t] = \mathsf{true} \\
(w,t,t') \vDash \theta(x_1,\dots,x_n) & \leftrightarrow & t < t' \text{ and } t'' \in (t,t'),\ \theta(w_{x_1}[t],\dots,w_{x_n}[t]) = \mathsf{true} \\
(w,t,t') \vDash r_1 \cdot r_2 & \leftrightarrow & \exists t''\ t \le t'' \le t',\ (w,t,t'') \vDash r_1 \text{ and } (w,t'',t') \vDash r_2 \\
(w,t,t') \vDash r_1 \cup r_2 & \leftrightarrow & (w,t,t') \vDash r_1 \text{ or } (w,t,t') \vDash r_2 \\
(w,t,t') \vDash r_1 \cap r_2 & \leftrightarrow & (w,t,t') \vDash r_1 \text{ and } (w,t,t') \vDash r_2 \\
(w,t,t') \vDash r^* & \leftrightarrow & \exists k \ge 0,\ (w,t,t') \vDash r^k \\
(w,t,t') \vDash \langle r \rangle_I & \leftrightarrow & \text{ and } (w,t,t') \vDash r \text{ and } t' - t \in I \\
(w,t,t') \vDash r_1 \,?\, r_2 & \leftrightarrow & (w,t,t') \vDash r_2 \text{ and } \exists t'' \le t,\ (w,t'',t) \vDash r_1 \\
(w,t,t') \vDash r_1 \,!\, r_2 & \leftrightarrow & (w,t,t') \vDash r_1 \text{ and } \exists t'' \ge t',\ (w,t',t'') \vDash r_2
\end{array}
$$

The last two operations associate a pre-condition (resp. post-condition) to the expression. We note that with the pre- and post-condition, we can also syntactically define rise and fall operators by using the rules $\uparrow p \equiv \neg p \,?\, \epsilon \,!\, p$ and $\downarrow p \equiv p \,?\, \epsilon \,!\, \neg p$. Extended STL specifications require regular expressions to be embedded into STL formulas. We define two operators, *begin match* ($@(r)$) and *end match* ($(r)@$) that intuitively project the signal segments (t,t') that matches an expression r to its beginning t and its end t', respectively. Thus, xSTL simply extends STL with these two operators:

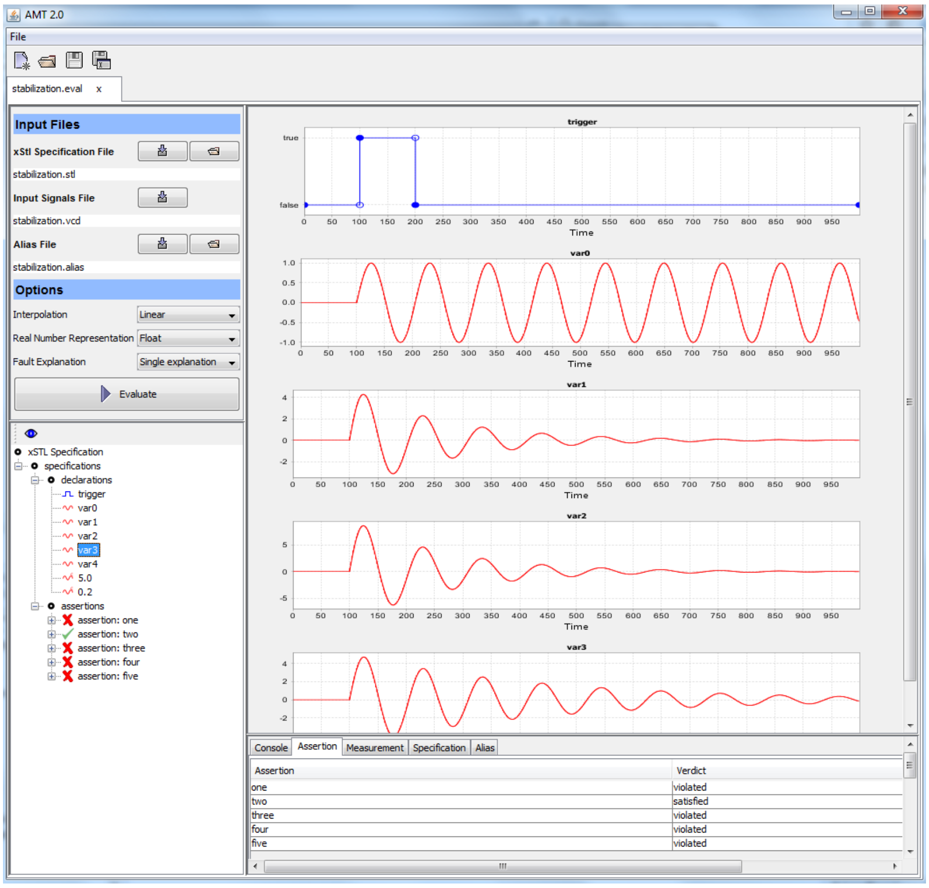
$$\varphi := p \mid \theta(x_1,\dots,x_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I \varphi_2 \mid \varphi_1 \,\mathcal{S}_I \varphi_2 \mid @(r) \mid (r)@$$

with the following semantics

$$
\begin{array}{l}
(w,t) \models @(r) \leftrightarrow \exists t' \ge t\ (w,t,t') \vDash r \\
(w,t) \models (r)@ \leftrightarrow \exists t' \le t\ (w,t',t) \vDash r
\end{array}
$$

3 Tool Presentation

The AMT 2.0 tool provides for qualitative and quantitative analysis of simulation/measurement traces. Its input consists of two major ingredients. The first is typically a formula or a collection of formulas in xSTL specifying the desired properties (and later measurements) of a continuous signal. The second is a finite representation of the continuous signal. Input signals obtained from simulators or measurement devices are given as finite sequences of time-stamped values of the form $(t_i, w[t_i])$. The tool supports two commonly-used formats: Value Change Dump (vcd) and Comma Separated Values (csv) files. To obtain continuous-time signals, values between sampling points are interpolated inside the tool to yield either piecewise-constant or piecewise-linear signals.



xStl Specification File: stabilization.stl

Input Signals File: stabilization.vcd

Alias File: stabilization.alias

Interpolation: Linear

Real Number Representation: Float

Fault Explanation: Single explanation

Assertion	Verdict
one	violated
two	satisfied
three	violated
four	violated
five	violated

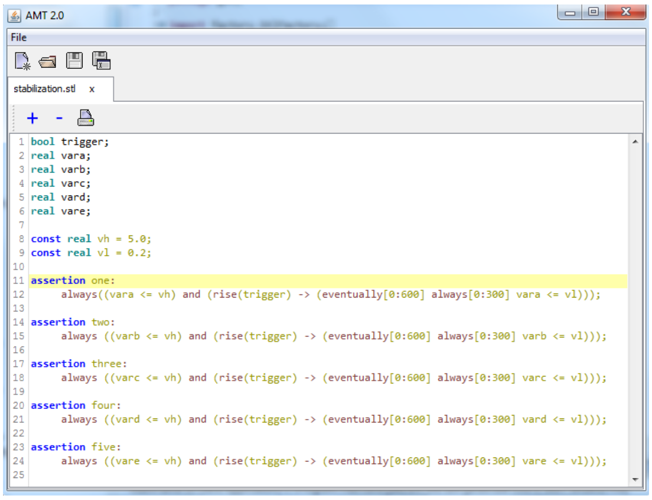
Fig. 1. AMT 2.0 - an overview of the graphical user interface.

The tool can work either interactively via its graphical user interface (GUI) or, alternatively, in batch mode when we want to monitor against many signals or incorporate monitoring in a more sophisticated analysis procedure that may iterate over behavior-generating models and/or properties in an outer loop. Figure 1 shows the main evaluation window of the GUI which provides two main functionalities: (1) editing xSTL specifications; and (2) launching the monitoring procedure by selecting properties and signals and presenting the outcome graphically. The AMT 2.0 tool is entirely implemented in Java to facilitate its usage across different platforms and operating systems.

The tool supports three major functionalities: (1) qualitative offline monitoring of extended STL specifications; (2) localization and explanation of property violations; and (3) measurements of quantitative features of signals driven by temporal pattern expressed using TRE. In the remainder of the section we present these functionalities in more detail.

3.1 Specifications in AMT 2.0

The tool facilitates specification of xSTL properties in several ways. The GUI provides an xSTL editor, depicted in Figure 2, with syntax highlighting and line numbering. In addition, the xSTL parser implements a number of features borrowed from programming languages. This includes (1) declaration of variables and constants, (2) paramterized property templates, (3) support for Boolean, real and bus variables and (4) type checking with extensive error reporting.



```
bool trigger;
real vara;
real varb;
real varc;
real vard;
real vare;

const real vh = 5.0;
const real vl = 0.2;

assertion one:
     always((vara <= vh) and (rise(trigger) -> (eventually[0:600] always[0:300] vara <= vl)));

assertion two:
     always ((varb <= vh) and (rise(trigger) -> (eventually[0:600] always[0:300] varb <= vl)));

assertion three:
     always ((varc <= vh) and (rise(trigger) -> (eventually[0:600] always[0:300] varc <= vl)));

assertion four:
     always ((vard <= vh) and (rise(trigger) -> (eventually[0:600] always[0:300] vard <= vl)));

assertion five:
     always ((vare <= vh) and (rise(trigger) -> (eventually[0:600] always[0:300] vare <= vl)));

```

Fig. 2. AMT 2.0 - xSTL editor.

3.2 Qualitative Monitoring of xSTL

In this section, we sketch the algorithm for the major functionality of the tool, qualitative monitoring of xSTL specifications. The procedure is based on two main methods that we describe in the sequel: the offline marking procedure for STL[16] and the pattern matching procedure for TRE [19].

The qualitative monitoring procedure for STL is an offline method that works directly on the input signals. The procedure is recursive on the structure of the specification – it propagates the truth values from input signals via super formulas up to the main formula. The algorithm uses the notion of a *satisfaction signal* – we assign to each sub-formula ψ of φ a Boolean signal w_ψ such that $w_\psi[t] = \texttt{true}$ iff $(w, t) \models \psi$. For each STL operator, we define a method that computes its satisfaction signal from the satisfaction signals of its arguments. For some operators, this computation is trivial. For example, satisfaction signal $w_{\neg\varphi}$ is obtained by flipping the truth values of the satisfaction signal w_φ. The computation of satisfaction signals for temporal operators is more involved. We give an intuition on the computation of w_ψ where $\psi = \diamondsuit_I \varphi$ and refer the reader to [16] for the technical description of the complete procedure. The computation is based on the following observation: whenever φ holds throughout an interval J, ψ holds throughout $(J \ominus I) \cap \mathbb{T}$, where $J \ominus I = \{t - t' \mid t \in J \text{ and } t' \in I\}$ is the Minkowski difference. Hence, the essence of the procedure is to back-shift (Minkowski difference saturated by $\mathbb{T}$) all the positive intervals in w_φ and thus obtain the set of time points where $\diamondsuit_I \varphi$ holds. This method is illustrated in Figure 3.

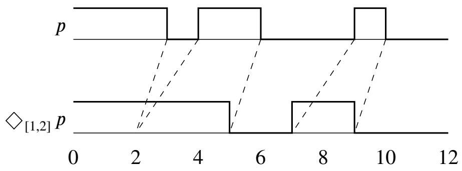


Fig. 3. Example of offline marking for $\diamondsuit_{[1,2]} p$ using back-shifting.

The integration of TRE into the monitoring procedure of xSTL is done in two steps. First, we define the *match-set* $\mathcal{M}(r, w)$ of a TRE over a signal w as the set of all segments of w that match r, i.e. $\mathcal{M}(r, w) = \{(t, t') \mid (w, t, t') \models r\}$, and use the algorithm of [19] to compute the match-set. We then use the match begin $(@(r))$ and match end $((r)@)$ operators to project the match-sets to satisfaction signals that are then directly integrated into the STL monitoring procedure described above.

The algorithm proposed in [19] computes the set of segments of a signal w that match a TRE φ. Since we are dealing with continuous-time signals, the number of segments is non-countable and so is potentially the number of matches. The algorithm is based on the observation that all those segments can be can be embedded in two-dimensional space, inside the triangle $0 \leq t \leq t' \leq |w|$, where a point (t, t') represents the segment starting at t and ending in t'. The matching algorithm uses a symbolic representation of the matches as a finite union of two-dimensional *zones*. Zones are special class of convex polytopes which are defined as the conjunction of inequalities of the form $x_i \prec b_i$ and $x_i - x_j \prec c_{i,j}$, where $\prec \in \{<, \leq\}$. For instance, the match set $\mathcal{M}(\epsilon, w)$ for the empty word ϵ is the diagonal zone $\{(t, t') \in \mathbb{T} \times \mathbb{T} \mid t = t'\}$, while the match for a literal p or $\neg p$ is a disjoint union of triangles touching the diagonal whose number depends on the number of switching points in w_p. The match set of the time restriction operator is obtained by intersecting the match set with the corresponding diagonal band, hence $\mathcal{M}(\langle\varphi\rangle_I, w) = \mathcal{M}(\varphi) \cap \{(t,t') \mid t'-t \in I\}$. The match sets for p and $\langle p\rangle_{[1,2]}$ are depicted in Figure 4. We point the reader to [19] for a complete description of the procedure. The satisfaction signals $w_{@(r)}$ and $w_{(r)@}$ for the match-begin and match-end operators are computed from the match set of r by projecting every $(t,t') \in \mathcal{M}(r)$ on t and t', respectively.

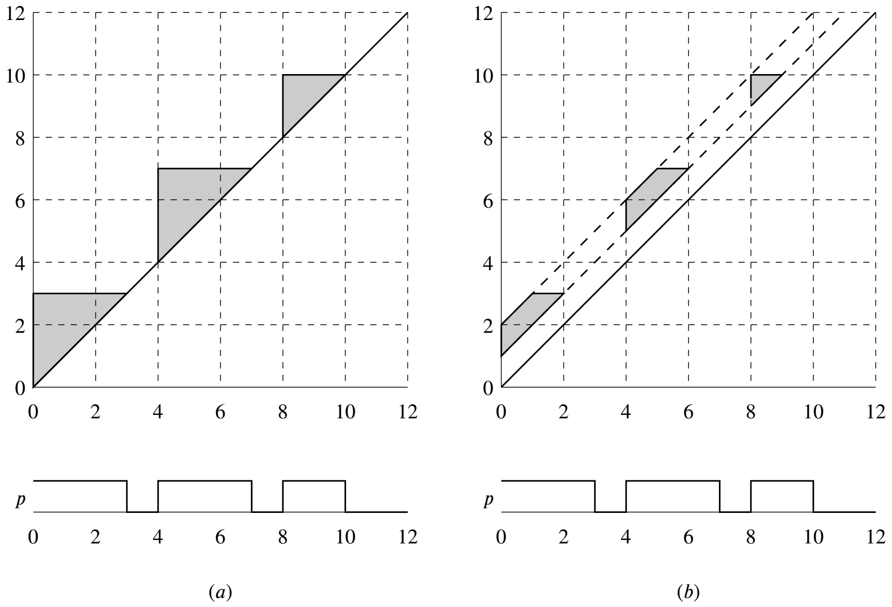


Fig. 4. Example of a match set - (a) p; and (b) $\langle p\rangle_{[1,2]}$.

3.3 Trace Diagnostics for STL

The trace diagnostics procedure implements the algorithm presented in [10]. Given an STL formula φ and a trace w that violates φ, the procedure gives an explanation of the fault in the form of a *temporal implicant*, which is a small sub-signal w' of w which is sufficient to imply violation. In other words, any possible completion of w' into a full signal will violate the property. The diagnostics procedure uses the satisfaction signals computed by the monitoring algorithm from Section 3.2 to explain the faults. The method uses the *satisfaction explanation* operator E (and its dual *violation explanation* operator F) that for a given formula φ returns an implicant of φ (respectively of $\neg\varphi$) which is satisfied by w. The explanation operators are defined inductively on the structure of the formula φ and on the times t at which explanation of its sub-formulas are required.

We illustrate the idea behind the procedure with the following example. Consider the STL specification $\varphi = \Diamond_{[0,1]} p$, a signal w in which p does not hold during $[0, 3)$ and then holds during $[3, 5)$. It is clear, for instance, that $(w, 0) \not\models \varphi$ and $(w, 3) \models \varphi$. The violation of φ by w at time 0 can be explained by the fact that w is continuously false throughout the interval $[0, 1]$. In other words, we have that $F(\varphi, w, 0) = \bigwedge_{t\in[0,1]}(w_p[t] = \mathsf{false})$. In contrast, the value of w at *any* time $t \in [3, 4]$ is sufficient to explain the satisfaction of φ by w at time 3. Thus $E(\varphi, w, 3)$ could be any $(w_p[t] = \mathsf{true})$ such that $t \in [3, 4]$. We define the notion of a *selection function* to choose one explanation when there are many possible ones. The full algorithm is described in [10].

3.4 Specification-driven Measurements

In this section, we present a simple declarative measurement specification language [11] built on top of TRE. The idea is to specify the signal segments over which measurement should be taken to be those that match some pattern specified by an expression. For example, to measure the time elapsed between the beginning and end of some activity, or the total fuel consumption in a segment where the acceleration pedal is continuously on until the velocity crosses some threshold.

We first recall that the match set of a TRE defines all the trace segments that match the expression, and the number of those can be uncountably infinite. However if we restrict ourselves to patterns that are delimited by instantaneous discrete events, we will have only finitely many matches. Formally, we use the following sub-class of expressions. An *event-bounded* TRE (E-TRE) is an expression of the form

$$\hat{r} := \uparrow p \mid \hat{r}_1 \cdot r \cdot \hat{r}_2 \mid \hat{r}_1 \cup \hat{r}_2 \mid \hat{r}_1 \cap r$$

with p a proposition, and $\hat{r}_1, \hat{r}_2$ event-bounded TREs.

The *measure patterns* that define the segments to be measured are of the form $\alpha \mathbin{?} \psi \mathbin{!} \beta$, where ψ is the *main* pattern, and α and β are, respectively pre- and post-conditions. The main pattern ψ specifies the portion of the signal over which the measure is taken. To guarantee a finite number of matching segments, ψ is restricted to be an E-TRE while α and β, which can be used to define additional constraints, are TREs.

Given a measure pattern φ and a signal w, we first compute all the segments of w that match φ. We then apply a measuring operator that collects specific signal values over the matched segments. A measure is written with the syntax $\mathrm{op}(\varphi)$ with $\mathrm{op} \in \{\mathsf{time}, \mathsf{value}_x, \mathsf{duration}, \mathsf{inf}_x, \mathsf{sup}_x, \mathsf{integral}_x, \mathsf{average}_x\}$. We finally aggregate the specific measures and provide to the user the minimum, maximum and average measured value, as well as a histogram that summarizes the measurements.

We illustrate specification-driven measurement with an example from the DSI3 automotive communication protocol [13]. The micro-controller and the sensors that use the protocol, communicate by sending *analog pulses* during the protocol initialization phase. The standard describes the acceptable shapes and duration of such pulses. Figure 5 depicts the specification of a *discovery response pulse* from the DSI3 standard. In particular, the standard defines the relevant thresholds ($2IResp$ and $IResp$) which are used to describe the shape, as well as the acceptable duration of the pulse's ramp (t_1) and its total duration (t_2).

To define the pulse pattern we first define the following predicates:

$$i_h \equiv i \geq 2IResp \quad i_b \equiv IResp \leq i < 2IResp \quad i_l \equiv i < IResp$$

and then let

$$\varphi = i_l \, ? \uparrow(i_b) \cdot i_b \cdot i_h \cdot i_b \cdot \downarrow(i_b) \, ! \, i_l.$$

We finally apply the measure operation $\mathsf{duration}(\varphi)$ to extract the duration of the segments that match the pulse pattern.

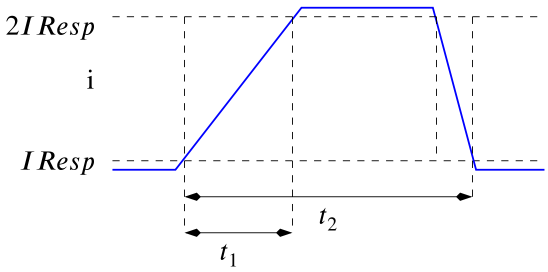


Fig. 5. Discovery response pulse from DSI3.

4 Examples

In this section, we introduce two running examples that we use to illustrate the features and the functionalities of AMT 2.0. The first example is concerned with a mixed-signal bounded stabilization property and is used to illustrate the qualitative monitoring and trace diagnostics functionalities. The second example demonstrates the measurement functionality as applied to jitter in a digital clock.

4.1 Mixed-Signal Bounded Stabilization

Informal Requirements This requirement states that after every rising edge of the Boolean *trigger*, the usually-stable analog signal *var* is allowed to oscillate under the following conditions:

1. *var* must always remain below $5V$; and
2. *var* must within $600s$ go below $0.2V$, and continuously remain under that threshold for at least $600s$.

Simulation Traces We evaluate this requirement on 5 different simulation traces. Figure 6 depicts the Boolean *trigger* signal, as well as the 5 traces named *var0* to *var4*. We can already reason informally about the satisfaction of the bounded stabilization property by these traces:

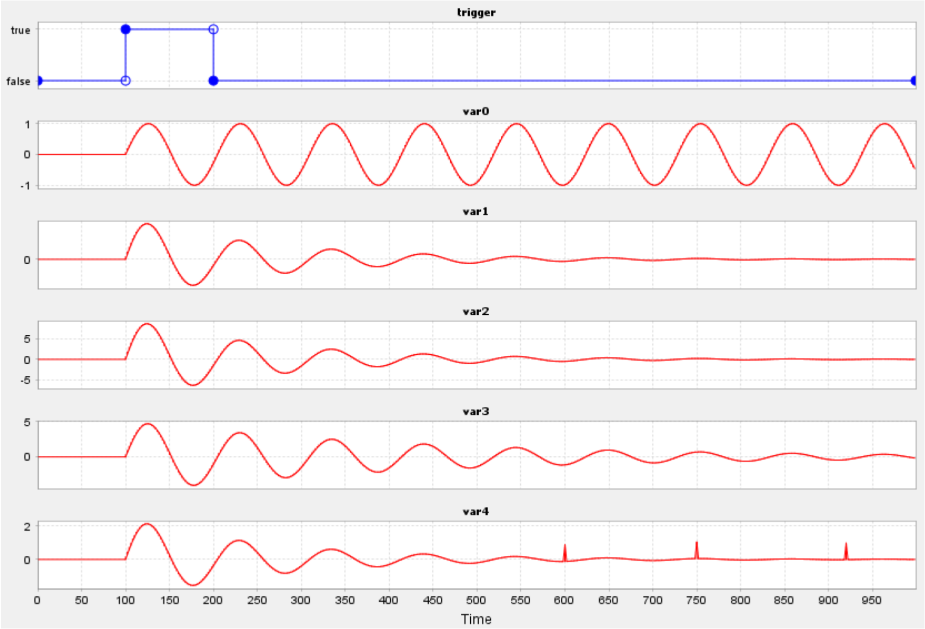


Fig. 6. Bounded stabilization - input signals.

1. Trace *var0* **violates** the specification because the signal never stabilizes, i.e. it continues oscillating until the end of simulation;
2. Trace *var1* **satisfies** the specification - the signal always remains smaller then $5V$, and it goes below $0.2V$ within $600s$, continuously remaining below that threshold until the end of the simulation;
3. Trace *var2* **violates** the specification because the signal exceeds $5V$;
4. Trace *var3* **violates** the specification because the signal does not stabilize below $0.2V$ within the specified period; and
5. Trace *var4* **violates** the specification because of the 3 glitches that occur towards the end of the simulation.

Formal Specification in xSTL To define the property we first declare the boolean variable *trigger*, as well as the real variables *var0* to *var4*. We also declare two constants *vh* and *vl*, representing the $5V$ and $0.2V$ thresholds, respectively. We note that we are evaluating the same formula over different signals. Hence, we define a generic property template *stab* for the bounded stabilization formula, which is the conjunction of conditions 1) and 2) of the informal requirements. The first conjunct says that the real-valued signal must be smaller than $5V$. The second conjunct is a conditional formula that uses logical implication. It says that whenever the *trigger* signal is on its rising edge, the x signal must go below $0.2V$ within $600s$ and continuously remain below that threshold for at least $300s$. Then each assertion is an instantiation of the template with one of the signals *var0* to *var4*.

```
bool trigger;
real vara;
...
real vare;
const real vh = 5;
const real vl = 0.2;

template bool stabilization (bool tg, real x, real vhigh,
    real vlow) {
  bool result = ((x <= vhigh) and (rise(tg) -> (eventually
    [0:600] always[0:300] x <= vlow)));
  return result;
}

assertion one:
  always(stabilization(trigger, vara, vh, vl));
...
assertion five:
  always(stabilization(trigger, vare, vh, vl));
```

Qualitative Monitoring of the Specification We illustrate the qualitative monitoring of the property applied to the traces as done using the GUI of the tool. In the evaluation configuration window, we first specify the xSTL specification, the simulation traces and an optional alias file. In addition to setting up the inputs, we also select the `Float` representation of the real numbers, the `Linear` interpolation and the `Single Explanation` feature of the diagnostics module.

After evaluating the specification on the traces, we can visually depict the results, as shown in Figure 1. The nodes in the xSTL parse tree view are expandable via a double click. By expanding the `assertions` node of the specification, we can see that assertion *two* is satisfied, while assertions *one*, *three*, *four* and *five* are violated. We note that we can visualize the satisfaction signals for any sub-property of the specification.

Fault Explanation The fault explanation is given in the form of temporal implicants which are (small) sub-segments of the input signals which are sufficient to imply the property violation. Figure 7 illustrates the visual output of the diagnostics procedure in AMT 2.0 for the bounded stabilization specification. The first two figures show the trace diagnostics report for the third assertion. We can see that the *trigger* signal does not contribute to the fault, but *var3* does at a single point in time within the interval [100, 150]. At that time, *var3* is greater than the invariant threshold $5V$ which explains the property violation. The last two figures show that same report, but for the fifth assertion. In this case, the fault is explained by the fact that signal *trigger* gets high at time 100 and by the values of signal *var4* at times 350, 600 and 750. We can see that the last two times coincide with the glitches, thus witnessing that *var4* never continuously holds below $0.2V$ for at least 300 time units.

We note that the tool computes the fault explanations in a hierarchical manner, following the parse tree of the formula. This additional and complementary information

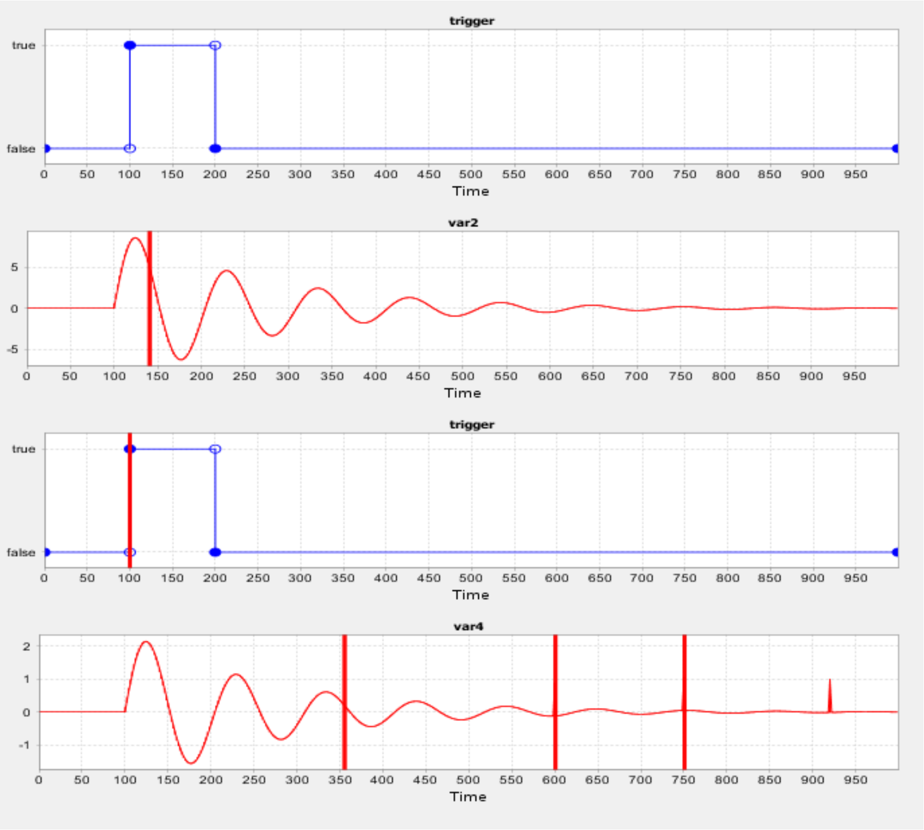


Fig. 7. Bounded stabilization - fault explanation.

can be quite useful in understanding the fault. Due to the space restrictions, we illustrate hierarchical trace diagnostics on the bounded stabilization example in Appendix A.

4.2 Digital Clock Jitter

Informal Requirements Given a continuous-time Boolean-valued signal *clock*, a clock period is defined as a segment that starts with the rising edge of the *clock* and ends with its consecutive rising edge. The measurement specification is to measure the duration of all the clock periods matched within the *clock* signal in order to assess the clock jitter..

Simulation Trace We apply the specifications to a Boolean *clock* signal. We depict one of its segment in Figure 8.

Formal Specification in xSTL We now formalize the measurement specification for the digital clock jitter analysis in xSTL. We first declare the Boolean variable *clock*,

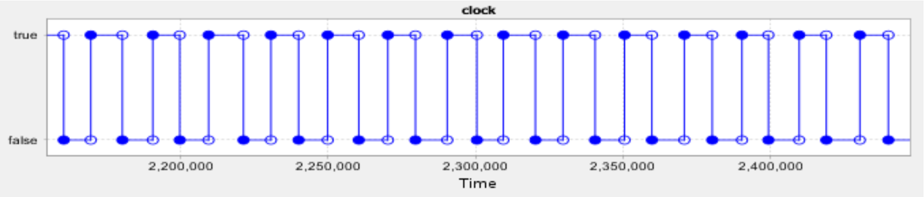


Fig. 8. Digital clock jitter - a segment of the input signal.

as well as its negation *nclock*. We then specify the pattern *clock_period* that consists of concatenations that starts with the rising edge of *clock* (*startclock*), followed by an interval of positive duration where *clock* holds, followed by another interval of positive duration where *nclock* holds, and ending with the next rising edge of *clock*. Finally, we declare the actual measurement to be taken as *duration(clock_period)* which extracts the durations of all signal segments that match the *clock_pattern* pattern.

```
bool clock;
bool nclock = not clock;


measurement jitter_clock_period {
  pattern clock_period = start(clock):clock:nclock:start(
    clock);
  measure duration(clock_period);
}

measurement jitter_clock_period_c {
  pattern clock_period_c = start(clock):{clock:nclock
    }[19000:21000]:start(clock);
  measure duration(clock_period_c);
}
```

Pattern-driven Measurements The visualization of the measurement specification consists of a histogram depicting the distribution of the measures taken over signal segments that match the pattern, the total number of matched segments, as well as the minimum, maximum and average value of the measures. The visual summary of the clock jitter measurement is shown in Figure 9.

5 Related Work

Breach [8] is a MATLAB/Simulink toolbox that enables various types of STL specification analysis. In particular, Breach supports falsification-based testing, parameter synthesis and requirement mining of STL properties.

S-TaLiRo [2] is another Simulink/MATLAB toolbox for different robustness analysis of MTL specifications. It provides support for falsification-based testing, parameter

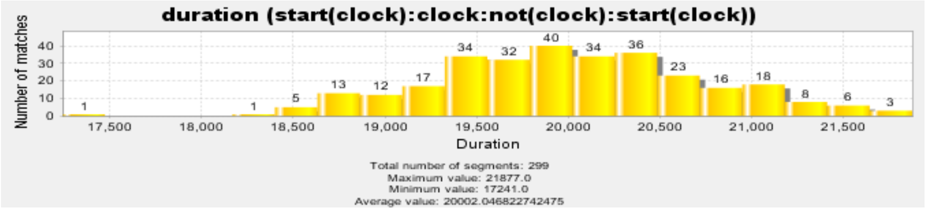


Fig. 9. Digital clock jitter - measurements.

mining, runtime verification, conformance testing, computing the worst expected robustness for stochastic systems and debugging of formal requirements. The ViSpec [12] tool, associated with S-TaLiRo, allows visual specification of MTL requirements.

Breach and S-TaLiRo are complementary to AMT 2.0 and focus on orthogonal features. Neither of them provides support for timed regular expressions, trace diagnostics nor property-driven measurements.

Montre [18] is a prototype tool for TRE pattern matching. It provides support for both offline and online matching. AMT 2.0 implements the offline matching algorithms used by Montre and adds a specification measurement language on top of it. Montre does not provide support for STL, monitoring and trace diagnostics.

The combination of STL and TRE was insipred by the Property Specification Language (PSL) [9] and SystemVerilog Assertions (SVA) [20] standards used in the digital hardware verification. Both PSL and SVA use the *suffix implication* operator to combine temporal logic with regular expressions. In contrast, we define *match begin* and *end* operators that give us more freedom to decide whether the begin or the end of an expression match is relevant for the property. The only other work that combines temporal logic and the regular expressions in the context of continuous-time applications is presented in [7], where the authors propose the *metric dynamic logic* as the specification language for reasoning about time-event sequences.

6 Conclusion

We introduced in this paper the AMT 2.0 tool for qualitative and quantitative analysis of traces coming from cyber-physical systems applications. The tool uses an expressive specification language based on a combination of STL and TRE and admits qualitative monitoring, trace diagnostics and property-driven measurements as its main functionalities. The development of the tool is a continuous work in progress and there is a number of features which are planned to be developed in the near future, in particular solving the inverse problem of finding parameters in a formula template the lead to satisfaction by a given signal or a set of signals [5].

References

1. Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

2. Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 254–257, 2011.
3. Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Logic in Computer Science (LICS)*, pages 160–171, 1997.
4. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.
5. Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers*, pages 147–160, 2011.
6. Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *The Handbook of Runtime Verification*. 2018.
7. David A. Basin, Srdan Krstic, and Dmitriy Traytel. Almost event-rate independent monitoring of metric dynamic logic. In *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*, pages 85–102, 2017.
8. Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 167–170, 2010.
9. Cindy Eisner and Dana Fisman. *A practical introduction to PSL*. Springer, 2006.
10. Thomas Ferrère, Oded Maler, and Dejan Nickovic. Trace diagnostics using temporal implicants. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, pages 241–258, 2015.
11. Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus. Measuring with timed patterns. In *Computer Aided Verification, 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2011, Proceedings*, 2015 (to appear).
12. Bardh Hoxha, Hoang Bach, Houssam Abbas, Adel Dokhanci, Yoshihiro Kobayashi, and Georgios Fainekos. Towards formal specification visualization for testing and monitoring of cyber-physical systems. In *International Workshop on Design and Implementation of Formal Tools and Systems, DIFTS'14*, 2014.
13. Distributed System Interface. *DSI3 Bus Standard*. DSI Consortium.
14. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
15. Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, pages 152–166, 2004.
16. Oded Maler and Dejan Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.
17. Dejan Nickovic and Oded Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, pages 304–319, 2007.
18. Dogan Ulus. Montre: A tool for monitoring timed regular expressions. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 329–335, 2017.

19. Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 222–236, 2014.
20. Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A practical guide for SystemVerilog assertions*. Springer, 2006.

A Hierarchical Trace Diagnostics Example

Here we illustrate step-by-step the construction and interpretation of diagnostics explanation. The fault explanation is hierarchical in the parse tree of the specification and is given in the form of *temporal implicants*, i.e. trace segments that are sufficient to explain the specification violation. We illustrate the fault explanation with assertion *five* of the mixed-signal bounded stabilization specification and show all the explanation steps in Figure 10.

Every specification which is violated is violated at time zero. This violation is due to the fact that the *always* operator is not satisfied throughout the simulation. By inspecting Figure 10, we can see that indeed, the *always* operation fails at time 100.

The violation of the always sub-formula at time 100 must be due to the fact that either the invariant or the bounded stabilization part of the formula is violated at time 100. In this case, it is indeed the bounded stabilization part of the formula is false at time 100.

This is the case because the boolean *trigger* is on its rising edge at that moment in time, but the analog signal x fails to drop below 0.2 within 600 time units and remain continuously below that threshold for at least 300 time units.

In Figure 10, we can see that the monitoring procedure fails to find a single point in the interval $[100, 700]$ from which the signal x remains continuously below the 0.2 threshold for at least 300 time units.

Finally, by inspecting the diagnostics results from Figure 10, we can see that the signal x does not continuously remain below 0.2 for at least 300 time units because the signal is still too high at time 350, and then there are two glitches at times 600 and 750 that are spaced in the intervals of duration smaller than 300 and that result in the property violation.

Remark: Due to the density of the continuous time, a specification violation may have infinitely many explanations. The diagnostics procedure implemented in AMT 2.0 computes a *single* fault explanation. The diagnostics algorithm is optimized to search for a small explanation in order to help the user understand the reason of the fault.

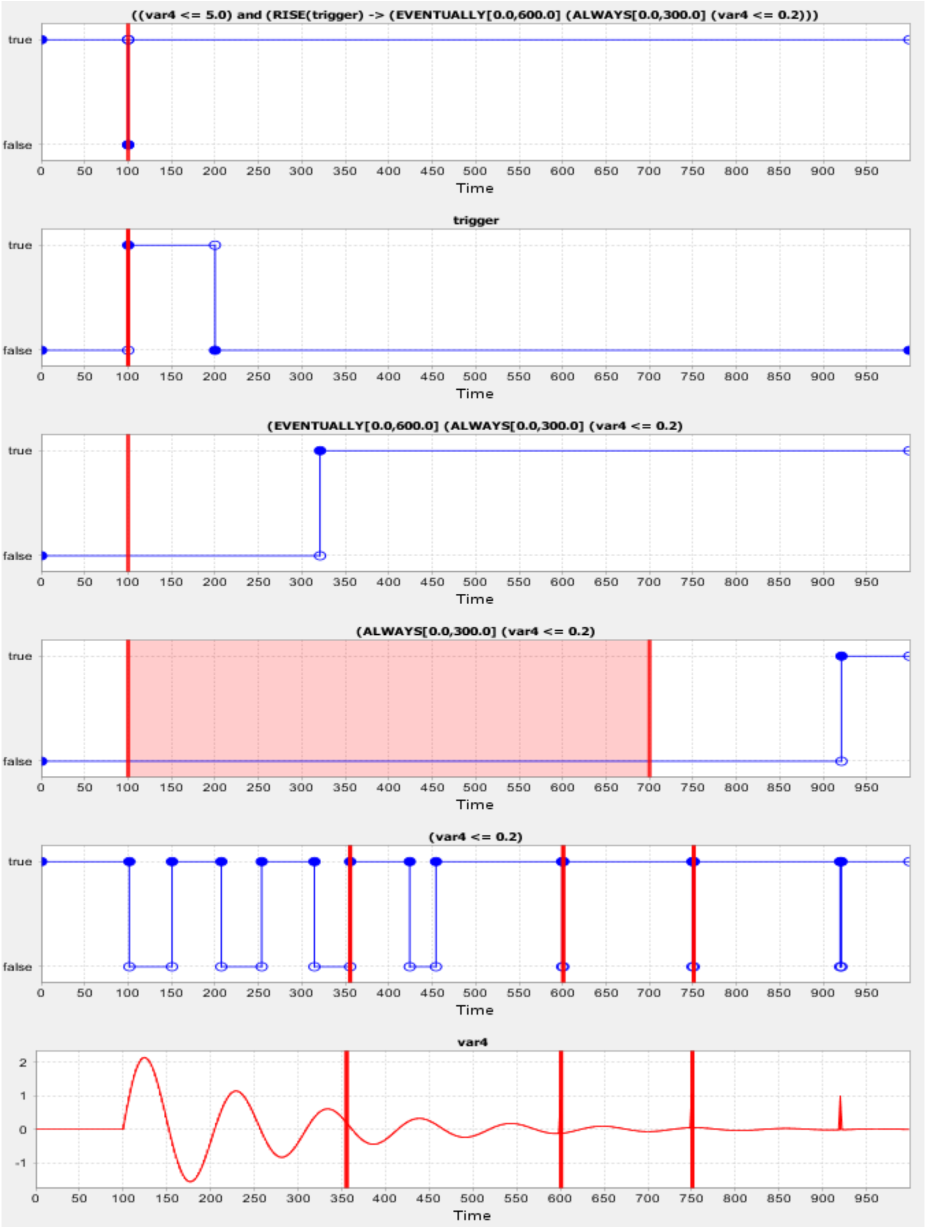


Fig. 10. Hierarchical fault explanation.