Parallel verification of temporal properties using dynamic analysis

Antoine Ferlin, Philippe Bon, Simon Collart-Dutilleul, Virginie Wiels

Parallel verification of temporal properties using dynamic analysis

(presented at the 6^{th} IESM Conference, October 2015, Seville, Spain)

Antoine Ferlin
Philippe Bon
Simon Collart-Dutilleul
Ifsttar,
20 Rue Élisée Reclus,
Villeneuve d'Ascq, France
antoine.ferlin@ifsttar.fr
philippe.bon@ifsttar.fr
simon.collart-dutilleul@ifsttar.fr

Virginie Wiels

Onera,
2 Avenue Édouard Belin,
31000 Toulouse, France
virginie.wiels@onera.fr

***Abstract*—Verification methods can be classified according to two kinds of criteria: static or not - i.e. dynamic - and formal or not. This paper follows a work about verification of temporal properties using dynamic analysis. The approach proposes to transform an LTL property into a Büchi automaton and to run the automaton on an execution trace to be verified. Because traces are finite, the end of trace problem can be bypassed with computation of statistical information about the verified trace if and only if the property follows a predefined given pattern. For very big traces, this approach is well-adapted, but traces have to be sequentially verified. This paper proposes to parallelize the verification approach by splitting the execution trace and executing the Büchi automaton on each sub-trace separately analysable, which allows a significant time saving.**

I. Introduction

The development of critical software is constrained by certification standards. The standard depends on the application domain. For instance, DO-178 is the certification standard for avionics software. In railway transportation, IEC 50128 is the certification standard. Even if standards define goals for each step of the software development, companies have responsibility for choosing the methods to achieve these goals. Known methods are proposed by standards but are not mandatory.

The verification phase is one of the steps of the software life-cycle. Several kinds of verification methods exist and can be classified using two criteria: formal or not, static or dynamic. Formal verification means the use of computable mathematical rules based on a language with unambiguous grammar and a defined semantics. These rules allow a mathematical verification of a given property. Static verification means that the program to be verified is not executed. On the contrary, dynamic verification requires the execution of the program.

For instance, classic verification means are typically not formal: the test is dynamic whereas reviews are static. We can notice that a lot of formal techniques are static: B methods [2], abstract interpretation [11], model checking [10]... RuntimeVerification[1], which is the purpose of this paper, can be classified as formal and dynamic.

This work follows a PhD thesis done at AIRBUS and at ONÉRA, the French aerospace lab, about verification of temporal properties [15], [16]. Several proprietary embedded programs have been studied in order to determine the properties which are difficult to be verified. But for reasons of industrial confidentiality, the detailed results cannot be published. According to this study, it appears that temporal properties are complex to be verified with the current industrial practices. No automatic verification method is currently used. Indeed, only code reviews are processed. The goal of the work was to propose a new equipped method to verify these temporal properties, in order to reduce time and cost of verification.

The resulting method is based on runtime verification. A specific language has been defined to formalize encountered temporal properties. This one is grounded on an aggregation of Linear Temporal Logic (LTL) and regular expressions, which are well adapted for sequence properties. The defined approach consists of two major steps: transforming the temporal property into a non-deterministic Büchi automaton, using *Ltl2ba* [17], and then executing the Büchi automaton on an execution trace to verify the property. Because LTL has a semantics on infinite traces, the finite execution trace is transformed into an infinite one by looping over the last state of the trace. In order to counter the side effects, statistical information is computed to help to the interpretation of the results, for unclear cases. The topic of this paper is not the language definition, the verification phase or the finite trace problem. Readers interested in this topic will find this information in [16].

This approach has been reused in the railway context. In order to save time during the verification phase, this paper presents a parallelized version of this method. This new method is based on the divide-and-conquer strategy using parallel execution and the result should be predictable. But, this strategy has to be validated, because we cut a partial execution trace, and because the analysis of a sub-trace naturally depends

[1]Runtime Verification, 2001-2014, www.runtime-verification.org

on the previous sub-traces. In addition, the fusion operation of each sub-trace analysis requires the computation of additional features in comparison with the classic sequential method. We aim to test this approach on traces coming form an European Rail Traffic Management (ERTMS) proprietary simulator[1]. However, because this platform is currently evolving with the PERFECT project[2], we first propose to do experimentation on a generated trace.

After this introduction, section II discusses this approach with regard to the related works and the industrial context. Section III summarizes the approach defined in [16]. Section IV defines some notations in this paper, then section V formalizes the classic approach which consists in executing a Büchi automaton on an execution trace and section VI formalizes the parallelized approach. Then, section VII presents experiments and comparison between the two methods. Finally, section VIII concludes this article discussing efficiency of the parallelized approach versus the sequential approach, and identifying the current limitations of the parallel method.

II. CONTEXT

A. Related Work

The existing works on runtime verification can be classified in two categories: online methods which perform the verification during execution of the program and a-posteriori methods, which perform verification on execution traces. There is a lot of work on online verification, especially for Java programs [13], [23], [21], [20] and in the aspect-oriented programming community [27]. Existing works are based on rewriting techniques [20] or specific techniques, such as translation of LTL formula into state machines [12], [19].

Reduction of verification time is a crucial issue in allowing us to deal with industrial software. In online verification, many efforts are made in this way. [6] proposes to reduce the number of monitors, using static analysis. [5] proposes a parallel verification of several three-valued-LTL properties encoded in a monitoring system using the GPU [3]. [24], [28], [29] separates monitors from the program to be verified.

The approach of this paper is an offline approach; execution traces come from a railway simulation framework. Verifications are done on execution traces rather than online, for several reasons. One of the most important is that the verification phase must not disturb the execution of the program to be verified. Actually, for real-time programs, a reduction of the execution speed can change the veracity of a property. In this context, there are three factors depending on the trace and affecting the verification time: the size of the trace, the number of variables inside the trace, and the format of the trace. The last factor is not optimizable because the simulator is proprietary.

In several existing works, execution traces are obtained by listening to all variables of the program to be verified, even if variables are not necessary to verify the specified properties [4], [25]. These approaches lead to generation of big traces, because some variables and states are unnecessary. Actually, verification time increases linearly with the size of traces, and with the average number of variables which are modified at each state.

Static analysis may be used to minimize the size of execution traces [15]. Static analysis detects all program points where variables relevent for the verified property are modified. An observation point is defined by collecting instruction at each needed location. But reduction of trace is not sufficient if the simulation goes on for several hours because some variables perpetually evolve and hence the size of the trace dramatically grows.

Consequently, the verification must be improved. [18] proposes an approach to verify several traces coming from a parallel application. Properties are verified using parallel processes (one process for each trace) instead of merging all traces into a unique trace to be analysed. This approach allows the verification of traces coming from parallel programs. If splitting the generation trace is not possible, another solution consists in splitting the trace after its generation and verifying the property on each piece of trace. This is our proposition.

B. Railway Transport context

Nowadays, each new line built inside the European Union must comply with both national rules and ERTMS/ETCS (European Railway Transport Management System/European Train Control System). ERTMS/ETCS specification is a European proposition about embedded systems and communication between trains and infrastructures.

This new technological and legal context must be implemented in different European states. Indeed, line operation rules must comply with the ERTMS/ETCS norm.

In this context, the considered software are critical and their verification is essential for avoiding collisions or near-collisions [3]. To do this, a proprietary simulator allows the simulation of the train behaviours. This proprietary simulator virtualizes control centres, railway with stations, markers, a train with control command and the communication with the control center. A train can be driven by an operator as a real train or by a defined scenario. We propose to adapt and to parallelize the approach of [16], in order to analyse temporal properties on execution trace of the simulator.

III. VERIFICATION APPROACH

In this section, the approach defined in [16] is summarized. The goal of this work has been to verify temporal properties on avionics embedded software.

A. Language

Instead of using a complex language, we chose to define a language which is well adapted to our needs. Hence, the first step of our work was to study the documentation of a lot of avionics software to extract temporal properties. For reasons of confidentiality, this study has never been published.

After extracting temporal properties, they were classified according to an extension of Dwyer's classification [14]. This expansion was necessary because frequency properties, properties with time interval, were not present in the original

[2]Performing Enhanced Rail Formal Engineering Constraints Traceability, http://perfect.ifsttar.fr/Site/

[3]Graphic Processing Unit

classification. The goal of this classification was to determine what kind of logic and operators were necessary.

Finally, the defined language is a combination of LTL (Linear Temporal Logic), regular expressions, and some additional operators for properties on numerical variables (integer, float): comparison between two numerical numbers, addition, substraction, multiplication, division. Operators on variables have been defined to change the variable scope: if x is a variable

- $x?T$ refers to the last time when x was modified,
- $x?C$ refers to the number of changes the variable suffered,
- $x?n$ refers to the value of the variable n changes ago,
- $x?n?T$ refers to the time when x was modified n changes ago,
- $x?n?C$ refers to the number of modifications n changes ago.

B. Trace Generation

In order to generate the trace, a dynamic analysis platform was used. This virtualizes the hardware of the software to be verified. For efficiency reasons, it was not possible to listen to a variable of the program. Observation points are used instead. As a consequence, in order to correctly verify a property using a set V of variables, it was necessary to set an observation point each time a variable of V changed.

Static analysis was used to find all necessary observation points. A plugin for Frama-C [9], called Breakpointer, was implemented to do this task. It was based on the Frama-C plugin Value Analysis which performs a semantic analysis. Hence, pointers on variables are taken into account.

Instead of using this avionics simulator, the ERTMS simulator will be used to generate execution traces. Each trace is stored inside a database. Each trace maps to one table with a timestamp, a type of message, and the associated values. The type of message describes which variable is modified.

C. Verification

The verification phase consists in:

- Transforming the temporal property into a Büchi automaton using *Ltl2ba*
- Verifying the property by executing the generated Büchi automaton on the trace execution.

If the temporal property follows a pattern, a statistical automaton is generated and executed concurrently to the Büchi automaton. This automaton is a state-transition machine, each transition contains a propositional logic formula to activate the transition and a list of counter assignments to change the value of defined global counters representing interesting information defined for each property pattern.

IV. Recall

A. Notations

In the following sections, some notations are used to help formalization of both approaches. The meaning of these notations is described hereafter:

- $\mathcal{P}(Q)$ is the set of subsets of a set Q
- the set E is a Cartesian product such as $E = E_1 \times E_2 \times ... \times E_n$, then $E_{|i}$ is projection of E on E_i. In other words, $E_{|i} = E_i$. If E_i is a Cartesian product itself such as $E_i = E_{i,1} \times ... \times E_{i,m}$, hence $E_{|i,j}$ refers to $E_{i,j}$. This notation is applicable on function ; $f : X_1 \times ... \times X_n \to Y_1 \times ... \times Y_m$ is a function. $f_{|i}$ refers to projection of f on Y_i.
- if E is a set, E^ω is the infinite Cartesian product of E.
- $[\![a; b]\!]$ such as $(a, b) \in \mathbb{N}^2$ and $a < b$ is the set of integers $\{a, a + 1, ..., b\}$.
- a trace σ is a sequence of states. A state $\sigma_i \in \Sigma$, where Σ is the alphabet, at index $i \in [\![0; |\sigma| - 1]\!]$ of the trace σ is a total function which returns the value of a given variable. $|\sigma|$ is the length of the trace σ. In this paper, **a trace is considered as a word** based on Σ alphabet in the meaning of formal language theory [26].
- if $\mathbb{K}$ is a set of values, then $\mathbb{K}^{¿} = \mathbb{K} \cup \{¿\}$, where ¿ is the *unknown* value;

In addition, item hereafter, to avoid confusions, we will speak about element to define a trace state and about state to define a automaton state.

B. Büchi automaton definition [8]

In this section, formal definitions of an automaton [22], of a Büchi automaton and of a statistical Büchi automaton are recalled. After, we schematized an execution of a Büchi automaton.

1) Büchi automaton:

Definition 1: An automaton is classically defined as a five-uplet $A = (Q, \Sigma, \to, q_0, F)$ where:

- Q is a set of states
- Σ an alphabet
- $\to \subseteq Q \times \Sigma \times Q$ is a transition relation
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is a set of accepting states.

The accepting condition of a finite word by an automaton is:

Definition 2: A finite word $w \in \Sigma^*$ is recognized by an automaton $A = (Q, \Sigma, \to, q_0, F)$, if and only if there is a sequence $(q)_{i \in [\![0;|w|]\!]}$, which starts with the initial state q_0, such that for all $i \in [\![0; n-1]\!]$, $(q_i, w_i, q_{i+1}) \in \to$, such that $q_n \in F$.

A word w which is recognized by A is written $w \in \mathcal{L}(A)$, where $\mathcal{L}(A)$ is the set of words recognized by A,

A Büchi automaton is a classic automaton with a special accepting condition which allows the handling of infinite traces. The accepting condition of a trace is:

Definition 3: An infinite word $w \in \Sigma^{\omega}$ is a word inside $\mathcal{L}(B)$, where $B = (Q, \Sigma, \rightarrow, q_0, F)$ is a Büchi automaton, if and only if:

- there is a sequence $(q)_{i \in \mathbb{N}}$ such that $\forall i \in \mathbb{N}$, $(q_i, w_i, q_{i+1}) \in \rightarrow$
- $\forall j \in \mathbb{N}\ \exists k \in \mathbb{N}$ such that $k > j$ and $q_k \in F$.

In this paper, a property has to be verified on an execution trace. Hence, the corresponding Büchi automaton is executed on this trace. Consequently, the alphabet used is built from the trace states.

In order to compute statistical information on a given infinite word, a statistical Büchi automaton is defined as an extension of a Büchi automaton. When a formula of a given transition is true, then primary operations, defined below, are done on some given counters. At the end of the execution of the statistical automaton, counters quantify properties which are orthogonal to the temporal property. The statistical Büchi automaton can be deterministic or not. Actually, a statistical automaton is not an automaton used to analyse a program whose the behaviour is not determined as in [7]. The word statistical only refers to the statistical information which are computed.

Example 1: For instance, the property to be verified is $\Box$ ($\Diamond$ e), which means e infinitely often occurs. The number of occurrences of e inside a given trace is statistical information.

Contrary to the work in [16], in this article, the statistical Büchi automaton replaces the classic Büchi automaton when a temporal property maps to the pattern of the statistical automaton. Before defining a statistical Büchi automaton, we define statistical operations.

Definition 4: $\mathcal{C}$ is a set of integer variables which will be called counters. $\Lambda_{\mathcal{C}} : (\mathcal{C} \rightarrow \mathbb{Z}^{\iota}) \rightarrow (\mathcal{C} \rightarrow \mathbb{Z}^{\iota})$ is a set of action on $\mathcal{C}$, depending on current value of all variables. Operations can be:

- doing nothing,
- assigning a constant or another counter value, or an expression which can be:
  - addition, subtraction, multiplication, division of counter/constant
  - minimum, maximum of counter/constant

In the case of parallel execution of a statistical deterministic Büchi automaton, the *unknown* value (¿) is used and allows the symbolic computation of the value of counters.

As a definition, the statistical operation λ of a given transition is a list of actions.

Finally, the statistical Büchi automaton is defined as follows:

Definition 5: A statistical automaton is a five-uplet $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$ where:

- Q is a set of states
- Σ an alphabet
- $\rightarrow \subseteq Q \times \Sigma \times \Lambda_{\mathcal{C}} \times Q$ a transition relation
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ a set of accepting states;
- $\mathcal{C}_0 : \mathcal{C} \rightarrow \mathbb{Z}^{\iota}$, is a function which returns initial value of each element of $\mathcal{C}$

Hereafter, the term *word* is replaced by *execution trace*.

2) Execution of a Büchi automaton: We use three cases of Büchi automaton: the deterministic one, the statistical deterministic one and the non-deterministic one. Before giving a formal definition of an execution of a Büchi automaton for each case, we illustrate these three cases with examples.

We speak about a step of computation when the Büchi automaton consumes an element in order to enable accessible transitions. When the Büchi automaton is deterministic (figure 1), at each step of computation, there is only one state before the consumption of the mapping element (current element) and at most one state after the consumption of the element. Hence, the execution of the Büchi automaton can be seen as a path with several nodes. Each node is a computation step and there is only one link between two nodes.

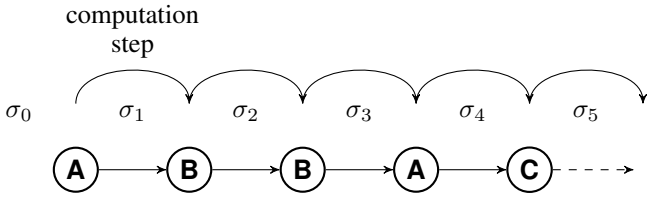


Fig. 1: Execution of a deterministic Büchi automaton

The execution of a statistical deterministic Büchi automaton corresponds to the execution of a deterministic Büchi automaton with a set of counters with their value for each current state. In figure 2, counters with their value are modelled by $\mathcal{C}_i$.

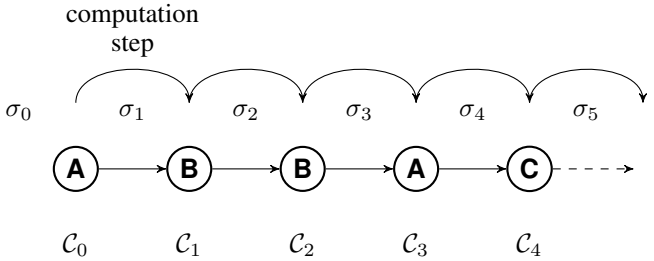


Fig. 2: Execution of a statistical deterministic Büchi automaton

If the Büchi automaton is not deterministic, then there will be a set of current states instead of one state. The path becomes a lattice instead of a sequence of automaton states. In figure 3,

each rectangle is a set of current states. Rectangle of level 1 is the initial state, the set $\{A\}$. One of level 2 is the set $\{A, B\}$ after consumption of the first element. And so on...

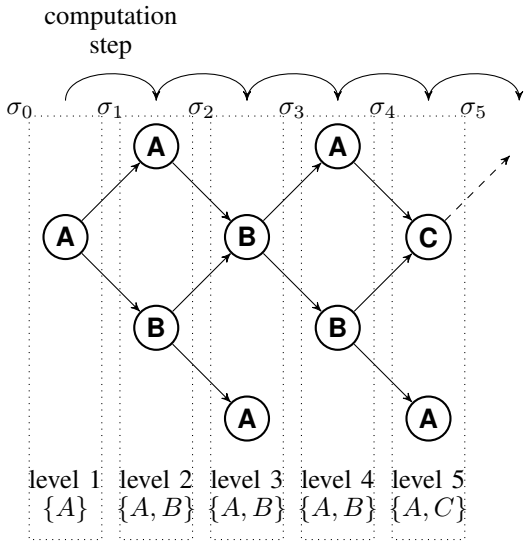


Fig. 3: Execution of a non-deterministic Büchi automaton

Schematically, a computation step consists in going from a rectangle of level i to the nearby one $(i + 1)$. Hence, the formal definition of an execution of a program has to contain two sets of state(s). The following sections take this fact into account.

The execution of a statistical non-deterministic Büchi automaton is not described. Consequently, in the rest of the article, we will speak about **statistical automaton** for a **Büchi statistical deterministic automaton**.

V. Classical execution of Büchi automaton on traces

Now, we formalize a sequential execution of a non-deterministic Büchi automaton and a statistical Büchi automaton.

A. Execution of a non-deterministic automaton on trace

The formalization of the execution of a non-deterministic Büchi automaton is a core material for the other cases.

In the approach in [16], regular expression are translated into deterministic Büchi automaton by an AIRBUS-proprietary tool, whereas LTL formulas are translated into non-deterministic Büchi automaton by *Ltl2ba*.

As a reminder, the formal definition 3 requires a sequence of states, hence the definition is sufficient for describing the first case. On the contrary, generated Büchi automata are non-deterministic in the second case. To deal with this case, definition 3 is sufficient. However, the complete formalization requires the definition of a sequence of set of states instead of a sequence of states.

Formalization of an execution of a non-deterministic Büchi automaton on a trace is based on figure 3. An automaton execution is a sequence of computation steps. Each computation step is a pair of state sets: one set before the trace state consumption and one set after. Formally, the definition of an execution of a Büchi automaton on a trace is:

Definition 6: An infinite word $w \in \Sigma^\omega$ is a word inside $\mathcal{L}(B_{nd})$, where $B_{nd} = (Q, \Sigma, \rightarrow, q_0, F)$ is a non-deterministic Büchi automaton, if and only if there is a sequence $R_{i\in\mathbb{N}} \in \mathcal{P}(Q)^2$ such that:

- $R_0 = (\{q_0\}, \{q_0\})$ (1)
- $\forall i \in \mathbb{N}^*$, $R_i \in \mathcal{P}(Q)^2$, such that:
  - $R_{i|1} = R_{i-1|2}$ (2)
  - $\forall r_i \in R_{i|2}, \exists r_{i-1} \in R_{i|1}$, such that $(r_{i-1}, w_i, r_i) \in \rightarrow$ (3)
- $\forall j \in \mathbb{N}$, $\exists k \in \mathbb{N}$, $k > j$ and $\exists R_{k|1} \cap F \neq \emptyset$. (4)

Property 2 ensures the consistency of the automaton execution. Property 3 is the part of the definition of an accepted word, which handles the existing path, for a given Büchi automaton. Property 4 is the other part of the definition of an accepted word, which requires an accepting state which is infinitely often reached.

If an infinite word w is not recognized by a non-deterministic Büchi automaton, then $\exists k$ such that $R_{k|2} = \emptyset$ and for all $k' > k, R_{k'} = (\emptyset, \emptyset)$, or else the sequence of the set of the current automaton states contains a finite number of accepting states.

In figure 3, we obtain the following sequence:

$R_0 = (\{A\}, \{A\})$
$R_1 = (\{A\}, \{A, B\})$
$R_2 = (\{A, B\}, \{A, B\})$
$R_3 = (\{A, B\}, \{A, B\})$
$R_4 = (\{A, B\}, \{A, C\})$

In our industrial railway context, handled finite traces are transformed into infinite ones by looping over the last state of the trace. This choice allows the use of existing tools such as *Ltl2ba* without modification. The finite trace problem is not the purpose of this article and was broached in [16].

The limit of the classic verification of a trace with a Büchi automaton is that the sequence $R_{i\in[\![0;|\sigma|-1]\!]}$ is sequentially computed. Hence, it slows the verification process.

B. Execution of a deterministic statistical Büchi automaton on trace

In this section, an execution of a statistical deterministic Büchi automaton on a trace is formalized.

Definition 7: An infinite word $w \in \Sigma^\omega$ is a word of $\mathcal{L}(B_S)$, where $B_S = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$, if and only if there is a sequence $R^S_{i\in\mathbb{N}} \in (Q)^2 \times (\mathcal{C} \longrightarrow \mathbb{Z}^\iota)^2$ such that :

- $R^S_0 = (q_0, q_0, \mathcal{C}_0, \mathcal{C}_0)$ (5)
- $\forall n \in \mathbb{N}^*$, $R^S_n \in (Q)^2 \times (\mathcal{C} \longrightarrow \mathbb{Z}^\iota)^2$ such that:
  - $R^S_{n|1} = R^S_{n-1|2}$ et $R^S_{n|3} = R^S_{n-1|4}$ (6)

- $(R^S_{n|1}, w_i, \lambda_n, R^S_{n|2}) \in \rightarrow$ (7)
- $R^S_{n|4} = \lambda_n(R^S_{n|3})$ (8)

- $\forall j \in \mathbb{N}, \exists k \in \mathbb{N}, k > j$ and $R^S_{k|1} \in F$. (9)

Property 6 ensures consistency of automaton execution. Property 7 is a part of the definition of an accepted word. Property 9 is the other pat of the definition of an accepted word, which requires an infinitely-often-reached accepting states. In property 7, w_n is the current element and λ_n is the current operation applied to the counters (property 8).

If we deal with figure 2, then the sequence will be:
$R^S_0 = (A, A, \mathcal{C}_0, \mathcal{C}_0)$
$R^S_1 = (A, B, \mathcal{C}_0, \mathcal{C}_1)$
$R^S_2 = (B, B, \mathcal{C}_1, \mathcal{C}_2)$
$R^S_3 = (B, A, \mathcal{C}_2, \mathcal{C}_3)$
$R^S_4 = (A, C, \mathcal{C}_3, \mathcal{C}_4)$

Despite the fact that trace analysis naturally looks like a single sequential process, we will show that parallel execution of the Büchi automaton improves the computing efficiency.

VI. Parallel execution of Büchi automaton on finite traces

A. Principle

Until now, when a trace has been verified by execution of a Büchi automaton, it has been necessary to compute $R_0, ..., R_{i-1}$, before computing R_i, because of the equality $R_{i-1|2} = R_{i|1}$. We propose the following new approach:

1) splitting the execution trace in several pieces,
2) executing a Büchi automaton B on each piece of the trace
3) merging the results of each execution of B to determine if the trace belongs to $\mathcal{L}(B)$.

Figure 4 synthesises the approach on a trace which is split into two pieces, at trace state 3. Two identical Büchi automata are executed on traces $\sigma_0..\sigma_2$ and $\sigma_3...\sigma_n$. The execution of the Büchi automaton on the first trace follows the rules defined in the previous section. The execution of the Büchi automaton on the second trace begins with Q ($\{A, B, C\}$) as the set of initial states.

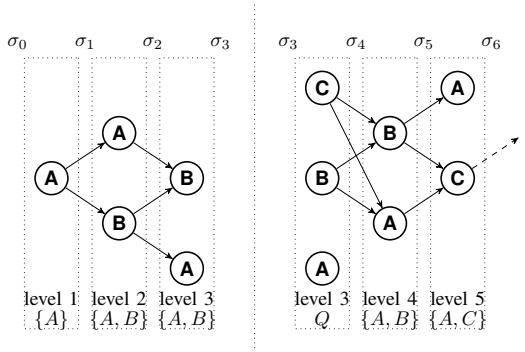
The verification of the property on the entire trace is performed using a function which acts as a short-cut between the beginning and the end of each part of the trace.

For instance, if the execution trace stops after trace state 5, we can see in Figure 4 that:

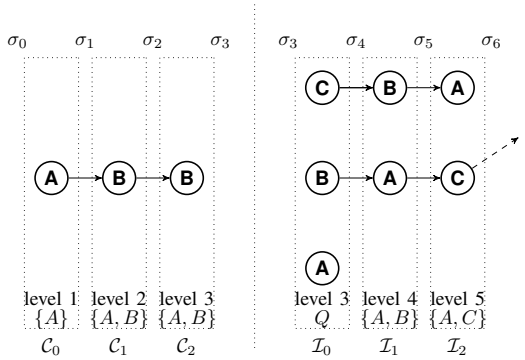
- state A before element 3 leads to nothing after element 5.
- states B and C before element 3 lead to A and C after element 5.

Because at level 3 the set of current states is $\{A, B\}$, the set of states after trace state 5 is $\{A, C\}$.

After presenting the principle, we will formalize it on non-deterministic and statistical deterministic Büchi automata. For reasons of clarity, the formalization will be done on a trace which is split into two sub-traces, but this one is generalizable to several divisions.



(a) non-deterministic



(b) statistical

Fig. 4: Principle of parallel approach

B. Execution of a non deterministic Büchi automaton

The trace σ is split into two sub-traces at element index c. The Büchi automaton will be executed on the both sub-traces. To do so, two sequences of computation steps will be defined: $\mathcal{R}_{n \in [\![0;c]\!]}$ and $\overline{\mathcal{R}}_{n \in [\![0;|\sigma|-c+1]\!]}$.

$\mathcal{R}_{n \in [\![0;c]\!]}$ is the sequence of computation steps which begins at element σ_0 as defined in section V. Hence, for all $n \in [\![0; c]\!]$, $\mathcal{R}_n = R_n$.

The sequence of computation steps on the second trace is defined as:

Definition 8: Sequence $\overline{\mathcal{R}}_{n \in [\![0;|\sigma|-c+1]\!]} \in (\mathcal{P}(Q))^2 \times (Q \longrightarrow \mathcal{P}(Q))^2$ is such that :

- Definition of the first two components of $\overline{\mathcal{R}}_n$ as following:
  - $\overline{\mathcal{R}}_{0|1} = Q$ (10)
  - $\forall n \in [\![0; |\sigma| + 1 - c]\!], \overline{\mathcal{R}}_{n|1} = \overline{\mathcal{R}}_{n-1|2}$. (11)
  - $\forall n \in [\![0; |\sigma|+1-c]\!], \forall r'_n \in \overline{\mathcal{R}}_{n|2}, \exists r_n \in \overline{\mathcal{R}}_{n|1}$, such that $(r_n, \sigma_{c+n+1}, r'_n) \in \rightarrow$ (12)
- The definition of the last two components of $\overline{\mathcal{R}}_n$ corresponds to the α function:

- $\forall n \in [\![0; |\sigma|+1-c]\!], \overline{\mathcal{R}}_{n|3} = \alpha_n,$ (13)
- $\forall n \in [\![1; |\sigma|+1-c]\!], \overline{\mathcal{R}}_{n|4} = \overline{\mathcal{R}}_{n+1|3},$ (14)
  - $\forall q \in Q,\ \alpha_0(q) = \{q\}.$ (15)
  - $\alpha_n(q) = \{q'' | \exists q' \in \alpha_{n-1}(q), (q'\sigma_{c+n+1}, q'') \in \rightarrow\}.$ (16)

Concretely, $\overline{\mathcal{R}}$ is the sequence of the execution of the Büchi automaton which begins at element index c and ends at element index $|\sigma|-1$. The initial set of states is Q. Function α_n records the link between the set of states at element $c+1$ and the set of states at element index $|\sigma|-1$.

Property 10 means that the sequence of computation steps is initialized with Q as the initial set of states. Properties 11 and 12 respectively ensure the consistency of the execution and th acceptance of a trace by the automaton. α is initialized with the identity function (property 15). For each state, α is recursively computed depending on activated transitions (property 16).

In figure 4a, the sequence will be:

$\mathcal{R}_0 = (\{A\}, \{A\})$
$\mathcal{R}_1 = (\{A\}, \{A, B\})$
$\mathcal{R}_2 = (\{A, B\}, \{A, B\})$
$\overline{\mathcal{R}}_0 = (\{A, B, C\}, \{A, B\}, \alpha = id, \{\alpha(A) = \emptyset, \alpha(B) = \{A, B\}, \alpha(C) = \{A, B\}\})$
$\overline{\mathcal{R}}_1 = (\{A, B\}, \{A, C\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A, B\}, \alpha(C) = \{A, B\}\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A, C\}, \alpha(C) = \{A, C\}\})$

Execution of the Büchi automaton on the first sub-trace can be performed independently of the execution on the second one. The results of the analysis of each sub-trace have to be merged using the α function.

Using the same approach, it is possible to divide the trace into more than two sub-traces. Actually, the prefix of the trace is classically analysed and we use the definition of $\overline{\mathcal{R}}$ for each other sub-trace.

C. Execution of a statistical deterministic Büchi automaton

Let us recall some notations: σ is an execution trace, $B_S = (Q, \Sigma, \rightarrow, q_0, F, \mathcal{C}_0)$ is a statistical deterministic Büchi automaton.

The execution sequences look like $\overline{\mathcal{R}}$ defined in section VI-B. An additional function is computed as α function, in order to built the operation applied on counters between the beginning and the end of each piece of trace.

$\mathcal{R}^S_{n \in [\![0;c]\!]}$ is the execution sequence which begins at trace state σ_0 as defined in section V-B. Hence for all $n \in [\![0; c]\!]$, $\mathcal{R}^S_n = R^S_n$.

The other sequence of computation steps is built as the previous-section one, taking into account the computation of statistical information.

Definition 9: $\overline{\mathcal{R}}^S_{n \in [\![0;|\sigma|+1-c]\!]} \in (\mathcal{P}(Q))^2 \times (Q \rightarrow \mathcal{P}(Q))^2 \times (Q \rightarrow \Lambda_{\mathcal{C}})^2$, where, for all $n \in [\![0; |\sigma|+1-c]\!]$:

- $\overline{\mathcal{R}}^S_{n|1} = \overline{\mathcal{R}}_{n|1}$, $\overline{\mathcal{R}}^S_{n|2} = \overline{\mathcal{R}}_{n|2}$
- $\overline{\mathcal{R}}^S_{n|3}$ (α function) is close to $\overline{\mathcal{R}}_{n|3}$. The last property about α defined at section VI-B is the only one which is modified:
  - $\forall q \in Q, \alpha_n(q) = \bigcup q''$ such that $\exists q \in \alpha_{n-1}, (q', \sigma_{c+n+1}, \lambda_{n,q'}, q'') \in \rightarrow$
  - property $\overline{\mathcal{R}}^S_{n|4}$=$\overline{\mathcal{R}}^S_{n+1|3}$ is preserved
- $\forall q \in Q, \overline{\mathcal{R}}^S_{0|5}(q) = Unknown.$
- $\forall q \in Q, \overline{\mathcal{R}}^S_{n|6}(q) = \lambda_{(n,q')}(\overline{\mathcal{R}}^S_{n|5}(q)).$
- $\forall q \in Q, \overline{\mathcal{R}}^S_{n|6}(q) = \overline{\mathcal{R}}^S_{n+1|5}(q).$

In figure 4b, if we take into account the statistical information, the sequence will be:

$\mathcal{R}_0 = (\{A\}, \{A\}, \mathcal{C}_0, \mathcal{C}_0)$
$\mathcal{R}_1 = (\{A\}, \{A, B\}, \mathcal{C}_0, \mathcal{C}_1)$
$\mathcal{R}_2 = (\{A, B\}, \{A, B\}, \mathcal{C}_1, \mathcal{C}_2)$
$\overline{\mathcal{R}}_0 = (\{A, B, C\}, \{A, B\}, \alpha = id, \{\alpha(A) = \emptyset, \alpha(B) = \{A\}, \alpha(C) = \{B\}\}, \mathcal{I}_0, \mathcal{I}_1)$
$\overline{\mathcal{R}}_1 = (\{A, B\}, \{A, C\}, \{\alpha(A) = \emptyset, \alpha(B) = \{A\}, \alpha(C) = \{B\}\}, \{\alpha(A) = \emptyset, \alpha(B) = \{C\}, \alpha(C) = \{A\}\}, \mathcal{I}_1, \mathcal{I}_2)$

If transition of X at Y implies the statistical computation $\lambda_{X,Y}$, then:

$\mathcal{I}_2(A) = Unknown$
$\mathcal{I}_2(B) = \lambda_{A,C}(\mathcal{I}_1(B)) = \lambda_{A,C}(\lambda_{B,A}(Unknown))$
$\mathcal{I}_2(C) = \lambda_{B,A}(\mathcal{I}_1(C)) = \lambda_{B,A}(\lambda_{C,B}(Unknown))\}$

D. The merging operation

After executing the Büchi automaton on each part of the trace, the result has to be generated using α function.

Let $\mathcal{F}$, the set of states at element $|\sigma|-1$. Hence $\mathcal{F} = \{q \in \overline{\mathcal{R}}_{|\sigma|-1|2}$, such that $\exists q' \in \mathcal{R}_{c|2}$ et $q \in \alpha_n(q')\}$.

Theorem 1: The Verification of a property by sequential execution of a Büchi automaton is equivalent to verification of a property by parallel execution of a Büchi automaton.

If the trace is split into more than two sub-traces, then the results will have to be sequentially merged.

Proof: Function α is constructed such that it allows the selection of states of $R_{|\sigma||1}$ coming from the set $\overline{\mathcal{R}}_0$. The fusion operation consists in determining states of $\overline{\mathcal{R}}_0$ which have to be kept. The states to be kept are those of the set $\mathcal{R}_c$. As a reminder, we have $\mathcal{R}_c \subset \overline{\mathcal{R}}_0 = Q$. Hence, the selection of states of $R_{|\sigma||1}$ coming from the set $\overline{\mathcal{R}}_0$ is equivalent to determining the generated states, for each state of $\mathcal{R}_c$. By definition of $\mathcal{F}$, therefore, we have $\mathcal{F} = R_{|\sigma||1}$. ■

Hence, the verification result can be computed using the end of trace algorithm defined in [16] and $\mathcal{F}$.

E. Limitations

In [16], specific operators on variables have been defined in order to use a counter of variable modifications or the value of a variable at a given number of its modification before the current trace state. See the examples given in section III-A.

Because these elements are recursively computed, parallel verification cannot be used. A possibility to handle these operators could consist in adding new variables inside the trace which map to these specific asked values.

VII. Experiments

In this section, the experiments results about the parallel-verification implemented approach are presented. Tests are performed on generic traces, for the reasons given in section I. The goal of this section is to test the efficiency of the approach, and the to compare it, in term of execution time, with the classic approach.

A. Implementation

Due to industrial confidentiality, it is not possible to provide the source code of the program which implements the approach of this paper. Nevertheless, some technical elements are given below. The prototype is written in C++ and is $32,900$ length (number of lines of code), with $13,000$ length of header file.

1) A distributed architecture: The implementation of both approaches (sequential and parallel) follows a distributed architecture, in order to efficiently share the available hardware resources. Indeed, the reading action of the trace is separated from the Büchi automaton execution. Figure 5 illustrates the relation between the processes of the sequential approach. The execution process and the reading process communicate through a TCP/IP interface. The execution process can load information from one or several readers. Actually, trace reading can be split and shared out among several reader processes. Figure 6 illustrates the communication relations between the processes of the parallel approach. The fusion operation is not presented in this figure. Each execution process can interact with one or several reader processes. In figure 6 only the case where each execution process maps with a reader process is presented.

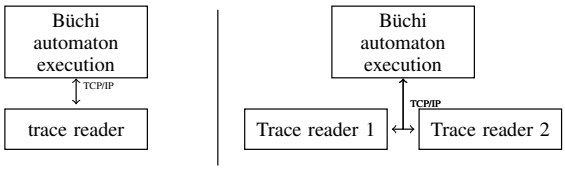


Fig. 5: Sequential implementation

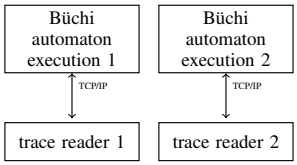


Fig. 6: Parallel implementation

2) The trace format: For this prototype, the trace format is based on Xml. The trace is split into several blocks which contain the same number of states $|b|$. Hence, when the execution process needs a state σ_i, it calls the reader process which returns the full block containing σ_i. The first state σ_i of this block contains the sub-trace $\sigma_i ... \sigma_{i+|b|-1}$. The use of blocks has two explanations.

Firstly, it limits the network latency. Test have shown that verification time increases dramatically if states are sent one by one. Secondly, Büchi automaton split follows the block segmentation. Actually, traces are partial[4]. But elements σ_i and $\sigma_{i+|b|-1}$ are complete[5] for each block b beginning at σ_i.

B. Load experiments

Two experiments are done: without statistical information, and with statistical information. The tests are done on the same generic execution trace for the two experiments. This trace has 10 million states and contains only one variable called x. For each trace state, the value of the variable changes. The domain of variation of the variable is $[\![-10; 10]\!]$. In order to do verification on different-sized traces, the verification can be stopped after a given number of states. Hence, we can verify a prefix of the given trace to simulate smaller traces.

Figure 8 gathers the results of the two experiments. The trace has been split into 2, 4, 8 or 10 sub-traces. The verification has been done for each sub-trace and the maximal time for each class of division has been placed in the graph. The memory process was launched in a 24Go-RAM computer, whereas the execution processes have been run on a 4Go-RAM computer with a Core-5i processor (2Ghz, 3Mo cache, 64-bits).

1) Without statistical information: The targeted property to verify on this trace is $\Box\ (x \leqslant 10 \wedge x \geqslant -10)$ which means that constraint $x \leqslant 10 \wedge x \geqslant -10$ has to be true for each state of the trace. The goal of this test is to compare a sequential verification with a parallel verification of this property on the trace.

The Büchi automaton of this property, which is obtained by *Ltl2ba*, is defined by figure 7a.

According to figure 8a, comparison with the sequential verification, verification time is reduced by 40% when the trace is split into two sub-traces. If the trace is split into ten sub-traces, verification time is reduced by 90%. The fusion time is weak: less than 0.01 seconds when the trace is divided into ten pieces.

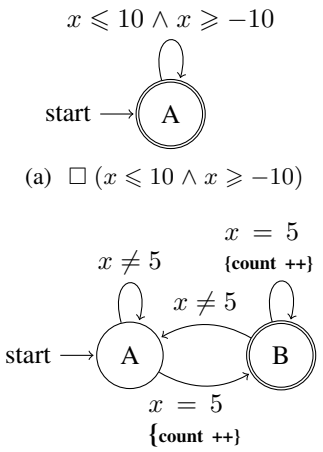
2) With statistical information: The targeted property to verify is $\Box\ (\Diamond\ (x = 5))$, which means that the constraint $x = 5$ occurs infinitely often. During the analysis of the trace, the number of states where the property $x = 5$ occurs is computed using the $count$ counter whose initial value is 0.

Figure 8b shows that the verification of the property is more efficient when the trace is split than when the trace is sequentially browsed. Time verification is divided by two when the trace is split into two pieces,and divided by four when the trace is split into four pieces of trace... The fusion time is weak: less than 0.01 of a second when the trace is divided into ten pieces.

This approach seems to be more efficient than the sequential one. The fusion operation which allows the merging of intermediate results requires less than 1 second. The next step of this work will consist in verifying traces which come from real industrial cases, with more complex properties, and traces with more variables. Actually, Büchi automata of complex

[4] An element only contains modified variables

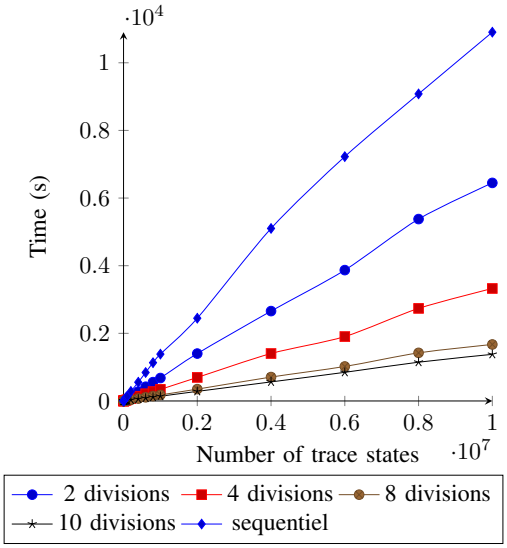
[5] All variables with their current values are inside these elements.



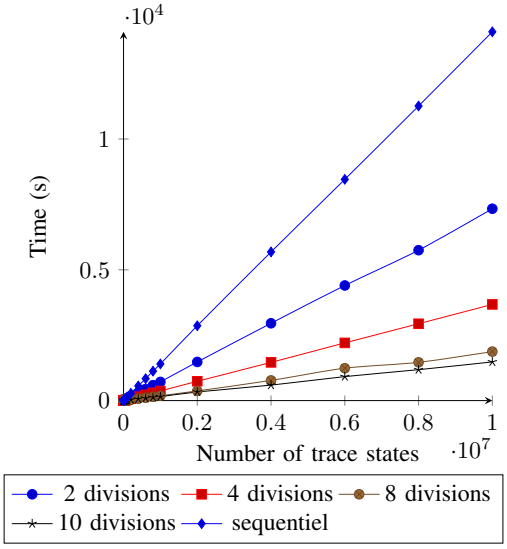
(a) $\square\,(x \leqslant 10 \wedge x \geqslant -10)$

(b) $\square\,(\Diamond\,(x = 5))$

Fig. 7: Büchi automata of the experiments



(a) without statistical information



(b) with statistical information

Fig. 8: Maximal time requirement for verification of a trace

property generally have more transitions and states than the experiments presented above.

VIII. Conclusion

In this paper, we formalize the execution on a trace of a non-deterministic Büchi automaton, of a deterministic one and of a statistical one. This formalizations are based on the classic formal definition of a Büchi automaton, which is not complete to entirely describe each computation step. This formalization is the basis of the main contribution of this paper which consists in parallelizing the execution of a Büchi automaton on a trace.

The analysed trace is split into several sub-traces. Then, the Büchi automaton is executed on each sub-trace. The execution on the first sub-trace is classic, whereas the execution on each other sub-trace is different. Indeed, the Büchi automaton is initialized with the set of automaton states instead of the initial state. During the execution of the Büchi automaton on a sub-trace, a short-cut is computed between an automaton state at the beginning of the execution and its generated set of automaton states at the end of the execution. After that, this short-cut is used to perform the fusion of the results of the first sub-trace with the results of the second one, in order to deduce the global result.

This new approach has been used to carry out experiments on generic traces of different sizes. The verification times of the parallel approach are compared to the ones of sequential approach. The verification time is approximatively divided by the number of sub-traces analysed. As a result, this new approach allows the splitting of the verification of a property on an execution trace to limit time and memory requirements. The next step consists in verifying a property on a real industrial case with the parallel approach and comparing time requirements with the sequential one.

A theoretical improvement moderation of this parallel approach is that the specific operators on variables developed in [16] cannot be used because they imply an on-the-fly computation which is not parallelizable. However, we show that the verification time is better with the parallel approach than with the classic one. In addition, this approach includes computation of statistical information on traces. The relevance of the divide-and-conquer strategy using parallel execution had to be proved in this context. The computing performance is a validation.

A perspective could be to enrich the trace with necessary data to allow the verification of properties with the specific operators. Another direction of research could be to determine after how many splits we no longer save verification time. Finally, this new parallel method has to be confronted with a real industrial case.

ACKNOWLEDGEMENT

The work is funded by the French national research agency (ANR) in the context of the PERFECT Project. This project is also supported by the French I-TRANS competitive pole.

REFERENCES

[1] European rail software applications. http://www.ersa-france.com/.

[2] J.-R. Abrial. The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA, 1996.

[3] F. Aguirre, M. Sallak, W. Schon, and F. Belmonte. Application of evidential networks in quantitative analysis of railway accidents. Proceedings of the Institution of Mechanical Engineers, Part O, Journal of Risk and Reliability, 227(4):368–384, Nov 2013.

[4] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Eagle does space efficient ltl monitoring. Technical report, Nasa, 2003.

[5] Shay Berkovich, Borzoo Bonakdarpour, and Sebastian Fischmeister. Gpu-based runtime verification. In Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IPDPS '13, pages 1025–1036, Washington, DC, USA, 2013. IEEE Computer Society.

[6] E. Bodden. Efficient hybrid typestate analysis by determining continuation-equivalent states. In Software Engineering, 2010 ACM/IEEE 32nd International Conference on, volume 1, pages 5–14, May 2010.

[7] Peter Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen, and Amelie Stainer. Monitor-based statistical model checking for weighted metric temporal logic. In Nikolaj Bjørner and Andrei Voronkov, editors, Logic for Programming, Artificial Intelligence, and Reasoning, volume 7180 of Lecture Notes in Computer Science, pages 168–182. Springer Berlin Heidelberg, 2012.

[8] J.Richard Büchi. On a decision method in restricted second order arithmetic. In Saunders Mac Lane and Dirk Siefkes, editors, The Collected Works of J. Richard Büchi, pages 425–435. Springer New York, 1990.

[9] CEA-LIST and INRIA-Saclay. The frama-c platform for static analysis of c programs, 2008.

[10] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst., 8(2):244–263, April 1986.

[11] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977.

[12] Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of omega-languages. In CAV'05, pages 364–378, 2005.

[13] Doron Drusinsky. The temporal rover and the atg rover. In K. Havelund, J. Penix, and W. Visser, editors, SPIN Model Checking and Software Verification, volume 1885 of Lecture Notes in Computer Science. Springer, 2000.

[14] M. Dwyer, G. Avrunin, and J. Corbett. Property specification patterns for finite-state verification. In Proceedings of the second workshop on Formal methods in software practice, FMSP '98. ACM, 1998.

[15] A. Ferlin and V. Wiels. Combination of static and dynamic analyses for the certification of avionics software. In Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on, pages 331–336, Nov.

[16] Antoine Ferlin. Vérification de propriétés temporelles sur des logiciels avioniques par analyse dynamique formelle. PhD thesis, Institut Supérieur de l'Aeronautique et de l'Espace (ISAE), Université de Toulouse, ED-MITT, September 2013.

[17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), volume 2102 of LNCS. Springer, 2001.

[18] Markus Geimer, Felix Wolf, BrianJ.N. Wylie, and Bernd Mohr. Scalable parallel trace-based performance analysis. In Bernd Mohr, Jesper-Larsson Träff, Joachim Worringen, and Jack Dongarra, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, volume 4192 of Lecture Notes in Computer Science, pages 303–312. Springer Berlin Heidelberg, 2006.

[19] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In Automated Software Engineering, 2001.

[20] K. Havelund and G. Rosu. Monitoring programs using rewriting. In Automated Software Engineering, pages 135 – 143, 2001.

[21] Klaus Havelund and Kestrel Technology. A rewriting-based approach to trace analysis. Automated Software Engineering, 12:2005, 2002.

[22] S C Kleene. Representation of events in nerve nets and finite automata. In In Automata Studies. Princeton University Press: Princeton, 1956.

[23] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the mop runtime verification framework. International Journal on Software Tools for Technology Transfer, 14:249–289, 2012.

[24] R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu. Hardware runtime monitoring for dependable cots-based real-time embedded systems. In Real-Time Systems Symposium, 2008, pages 481–491, Nov 2008.

[25] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In FM 2006: Formal Methods, volume 4085 of LNCS. Springer, 2006.

[26] Grzegorz Rozenberg and Arto Salomaa, editors. Handbook of Formal Languages, Vol. 1: Word, Language, Grammar. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[27] Volker Stolz and Eric Bodden. Temporal assertions using aspectj. Electronic Notes in Theoretical Computer Science, 144(4):109 – 124, 2006. Proceedings of the Fifth Workshop on Runtime Verification (RV 2005).

[28] Haitao Zhu, M.B. Dwyer, and S. Goddard. Predictable runtime monitoring. In Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on, pages 173–183, July 2009.

[29] C.B. Zilles and G.S. Sohi. A programmable co-processor for profiling. In High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on, pages 241–252, 2001.