Combining Genetic Programming and Model Checking to Generate Environment Assumptions

Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C. Briand, Yago Isasi Parache

***Abstract*—Software verification may yield spurious failures when environment assumptions are not accounted for. Environment assumptions are the expectations that a system or a component makes about its operational environment and are often specified in terms of conditions over the inputs of that system or component. In this article, we propose an approach to automatically infer environment assumptions for Cyber-Physical Systems (CPS). Our approach improves the state-of-the-art in three different ways: First, we learn assumptions for complex CPS models involving signal and numeric variables; second, the learned assumptions include arithmetic expressions defined over multiple variables; third, we identify the trade-off between soundness and coverage of environment assumptions and demonstrate the flexibility of our approach in prioritizing either of these criteria.**

We evaluate our approach using a public domain benchmark of CPS models from Lockheed Martin and a component of a satellite control system from LuxSpace, a satellite system provider. The results show that our approach outperforms state-of-the-art techniques on learning assumptions for CPS models, and further, when applied to our industrial CPS model, our approach is able to learn assumptions that are sufficiently close to the assumptions manually developed by engineers to be of practical value.

***Index Terms*—Environment assumptions, Model checking, Machine learning, Decision trees, Genetic programming, Search-based software testing**

I. Introduction

Software verification can be applied either at component-level or system-level; and be performed exhaustively using formal methods or partially via automated testing. Irrespective of the level at which verification is conducted or the technique used for verification, the results of verification may include spurious failures due to the violation of implicit or environment assumptions. Environment assumptions are the expectations that a system or a component makes about its operational environment and are often specified in terms of conditions over the inputs of that system or component. Any system or component is expected to operate correctly in its operational environment, which includes its physical environment and other systems and components that it interacts with. However, attempting to test or verify systems or components for a more general environment than their expected operational environment may lead to overly pessimistic results by detecting spurious failures [1].

Environment assumptions are rarely fully known for software systems [2]. Manual identification of environment assumptions is tedious and time-consuming. This problem is exacerbated for cyber-physical systems (CPS) that often have complex behavior. For CPS, we often need to apply verification at the component-level, e.g., when exhaustive and formal verification do not scale to the entire system. For most practical cases, engineers simply may not have sufficient information about the details of every component of the CPS under analysis and are not able to develop sufficiently detailed, useful and accurate environment assumptions for individual CPS components.

The ultimate use of environment assumptions is to help determine the validity of test inputs applied either at the system or component levels, or to know valid input ranges for exhaustive verification. The latter usage has been extensively studied in the area of formal verification where the goal is to use environment assumptions to automate compositional assume-guarantee reasoning (e.g., [3]–[7]). There have been approaches to automate the generation of environment assumptions in the context of assume-guarantee reasoning using an exact learning algorithm for regular languages and finite state automata [1], [8], [9]. These approaches, however, assume that the components under analysis and their environment can be specified as abstract finite-state machines. Such state machines are not expressive enough to capture CPS, and in particular quantitative and numerical CPS components and their continuous behavior. Besides, CPS components may not be readily specified in (abstract) state machine notations and specifying them into this notation may require considerable additional effort, which may not be feasible or beneficial.

In our earlier work, we proposed EPIcuRus [10] (assumPtIon geneRation approach for CPS) to automatically generate assumptions for CPS models. EPIcuRus addresses the limitations discussed above by generating assumptions for CPS models specified in Simulink®, which is a dynamic modeling language commonly used for CPS development and can specify complex mathematical and continuous functions over numeric and signal variables. EPIcuRus receives as input a CPS component M specified in Simulink® and a requirement ϕ. It automatically infers a set of environment assumptions (i.e., conditions) on the inputs of M such that M satisfies ϕ when its inputs are restricted by these conditions. EPIcuRus uses search-based testing to generate a set of test cases for M exercising requirement ϕ such that some test cases are passing

K. Gaaloul and C. Menghi are with University of Luxembourg, Luxembourg.
E-mail: {khouloud.gaaloul,claudio.menghi}@uni.lu
S. Nejati and L. Briand are with University of Ottawa, Canada and University of Luxembourg, Luxembourg.
E-mail: {snejati,lbriand}@uottawa.ca.
Y. I. Parache is with LuxSpace, Luxembourg.
E-mail: isasi@luxspace.lu

and others are failing. The generated test cases and their pass/fail results are then fed into a decision tree classification algorithm [11] to automatically infer an assumption $\mathtt{A}$ on the inputs of M such that M is likely to satisfy ϕ when its inputs are restricted by $\mathtt{A}$. Model checking is then used to validate the soundness of $\mathtt{A}$. Specifically, model checking determines if M provably satisfies ϕ when its inputs are constrained by $\mathtt{A}$. If so, $\mathtt{A}$ is a sound assumption. Otherwise, EPIcuRus continues until either a sound assumption is found or the search time budget runs out.

While EPIcuRus was effective in computing sound assumptions involving signal and numeric variables for industrial Simulink® models, the structure of the assumptions generated by EPIcuRus was rather simple. Specifically, EPIcuRus could only learn conjunctions of conditions where each condition compares exactly one signal or numeric variable with a constant using a relational operator. This is because EPIcuRus uses decision tree classifiers that can only infer such simple conditions. In our experience, however, assumptions produced by EPIcuRus, while being sound, do not have the largest coverage that can be learned for many CPS Simulink® models. In this paper, the goal is to learn an assumption that is not only sound (i.e., makes the component satisfy its requirements), but also has large coverage (i.e., is ideally the weakest assumption or among the weaker assumptions that make the component satisfy the requirement under analysis). For example, the actual assumption of our industrial case study—the attitude control component of the ESAIL maritime micro-satellite—is in the following form: $\mathtt{A}_1 ::= \forall t \in [0,1] : \alpha{\cdot}x(t){+}\beta{\cdot}y(t) < c$, where x and y are signals defined over the time domain $[0,1]$. But EPIcuRus, when relying on decision trees, is not able to learn any assumption in that form and instead learns assumptions in the following form: $\mathtt{A}_2 ::= \forall t \in [0,1] : \alpha' \cdot x(t) < c' \wedge \beta' \cdot y(t) < c''$. Assumptions in the latter form, even though sound, have lower coverage than the actual assumptions.

In this paper, we extend EPIcuRus to learn assumptions containing conditions that relate multiple signals by both arithmetic and relational operators. We do so using genetic programming (GP) [12]–[16]. Provided with a grammar for the assumptions that we want to learn, GP is able to generate assumptions that structurally conform to the grammar [17], and in addition, maximize objectives that increase the likelihood of the soundness and coverage of the generated assumptions. EPIcuRus still applies model checking to the assumptions learned by GP to conclusively verify their soundness. The coverage, however, is achieved through GP and partly depends on the structural complexity of the learned assumptions. Any assumption that can be structurally generated by our grammar is in the search space of GP. Therefore, EPIcuRus with GP has more flexibility compared to the old version of EPIcuRus and can search through a wider range of structurally different assumption formulas to build more expressive assumptions that are likely to have a larger coverage as well.

Note that, in the context of CPS, as assumptions have a larger coverage and become structurally complex, establishing their soundness becomes more difficult as well. As discussed above, soundness can only be established via exhaustive verification (e.g., model checking). In our experience with industrial CPS Simulink® models, model checkers fail to provide conclusive results by either proving or refuting a property when the assumption used to constrain the model inputs becomes structurally complex (e.g., when it involves arithmetic expressions over multiple variables). Therefore, the larger the coverage, the less effective exhaustive verification tools in proving their soundness, and vice-versa. Hence, if guaranteed soundness is a priority, engineers may have to put up with assumptions with lower coverage, and conversely, they can have assumptions with large coverage, whose soundness is not proven.

We evaluated EPIcuRus using two separate sets of models: First, we used a public-domain benchmark of Simulink® models provided by Lockheed Martin [18], a company working in the aerospace, defense, and security domains; second, we used a more complex model of the attitude control component of a microsatellite provided by LuxSpace [19], a satellite system provider. EPIcuRus successfully computed assumptions for 18 requirements of four benchmark models from Lockheed Martin [18] and one requirement of the attitude control component from LuxSpace. Note that, among all of our case study models, only these requirements needed to be augmented with environment assumptions to be verified by a model checker.

We check each requirement for model inputs conforming to different signal shapes, specifying different ways the values of input signals change over time. We refer to signal shapes as input profiles and define a number of specific input profiles in our evaluation. Note that not all input profiles are valid for every model and requirement. In total, we consider 32 combinations of requirements and input profiles. Our evaluation targets two questions: if genetic programming (GP) can outperform decision trees (DT) and random search (RS) in generating sound assumptions with large coverage (RQ1), and if the assumptions learned by EPIcuRus are useful in practice (RQ2). For RQ1, we considered all the 32 requirement and input profile combinations. Our results show that GP can learn a sound assumption for 31 out of 32 combinations of requirements and input profiles, while DT and RS can only learn assumptions for 26 and 22 combinations, respectively. The assumptions computed by GP have also a significantly (20% and 8%) larger coverage than those learned by DT and RS. For RQ2, we considered the attitude control component from LuxSpace since this is a representative and complex example of an industrial CPS component, and more importantly, in contrast to the public-domain benchmark, we could interact with the engineers that developed this component to evaluate how the assumptions computed by EPIcuRus compare with the assumptions they manually wrote. Our results show that, when EPIcuRus was configured to proritize coverage, as opposed to proving the soundness of assumptions, learned assumptions were syntactically and semantically close to those written by engineers. Conversely, when learning assumptions whose soundness can be verified was prioritized, EPIcuRus was able to generate sound assumptions in around six hours. Though simpler than the actual assumption, they provided useful, practical insights to engineers. We note that none of the existing techniques for learning environment assumptions is able to handle our attitude control component case study or learn assumptions that are structurally as complex as those required for this component.

Structure. Section II introduces the ESAIL running example. Section IV outlines EPIcuRus and its pre-requisites. Section III formalizes the assumption generation problem. Section V presents how EPIcuRus is implemented. Section VI evaluates EPIcuRus, and Section VI-E discusses the threats to validity. Section VII compares with the related work and Section VIII concludes the paper.

II. ESAIL Microsatellite Case Study

OUr case study system, the ESAIL maritime microsatellite, is developed by LuxSpace [19], our industrial partner, in collaboration with ESA [20] and ExactEarth [21]. ESAIL aims at enhancing the next generation of space-based services for the maritime sector. During the design phase of the satellite (i.e., development phases B-C [22]), the control logic of the ESAIL software is specified as a Simulink® [23] model. Before translating the Simulink® model into code that is going to be ultimately deployed on the ESAIL satellite, LuxSpace engineers need to ensure that the model satisfies its requirements.

The ESAIL Simulink® model is a large, complex and compute-intensive model [24]. It contains 115 components (Simulink® subsystems [25]) and a large number (2817) of Simulink® blocks of different types such as S-function blocks [26] containing Matlab code, and some MEX functions [27] executing C/C++ programs containing the behavior of external third party software components. Due to the above characteristics, exhaustive verification of the ESAIL Simulink® model (e.g., using model checking) is infeasible. For example, QVtrace [28], an industrial model checker developed by QRA Corp [29] and dedicated to model checking Simulink® models, cannot load the model of ESAIL since many components cannot be handled by the QVtrace model checker.

Even though the entire ESAIL Simulink® model cannot be verified using exhaustive verification, it is still desirable to identify critical components of ESAIL that are amenable to model checking. For example, the Attitude Control (AC) component of the attitude determination and control system (ADCS) of ESAIL monitors the environment in which the satellite is deployed and sends commands to its actuators according to classical control laws used for the implementation of satellites [30]. Specifically, it receives from the attitude determination system the estimated values of the speed, the attitude, the magnetic field and the sun measurements. It also receives commands from the guidance such as the target speed and attitude of the satellite. Then, it returns the commanded torque to the reaction wheel and the magnetic torquer. AC has ten inputs and four outputs that represent the commanded torque to be fed into the actuators. Note that some of the inputs and outputs are vectors containing several input signals, i.e., virtual vectors [31]. The inputs and outputs of AC are summarized in Table I. AC contains 1142 blocks. It can be loaded in QVtrace after replacing the 19 S-function blocks, that cannot be processed by QVtrace, with a set of Simulink®

TABLE I
Name of the Input, number of input signals in the virtual vector (NS), and description of the input.

	Name	NS	Description
Input	q_t	4	Target attitude of the satellite.
	q_e	4	Estimated attitude of the satellite.
	ω_t	3	Target speed of the satellite.
	ω_e	3	Estimated speed of the satellite.
	B	3	Measured Magnetic field.
	Md	1	The mode of the satellite.
	Rwh	4	Angular momentum of the reaction wheel.
	Ecl	1	Indicates if the satellite is in eclipse.
	SF	1	Indicates if the sun sensor is illuminated.
	SM	3	Sun sensor measurements.
Output	MT	4	magnetic dipole applied to the magnetorquers.
	MTc	4	Current applied to the magnetorquers.
	RW	3	Torque applied to the reaction wheel.
	RWa	3	The reaction wheel's torque acceleration.

TABLE II
Example requirements for the attitude control component.

ID	Requirement
ϕ_1	When the norm of the attitude error quaternion is less than 0.001, the torque commanded to the reaction wheel around the x-axis (with respect to its body frame) shall be within the range $[-0.001{,}0.001]\mathrm{N}\cdot\mathrm{m}$.
ϕ_2	The magnetic moment of each magnetorquer shall be within the range $[-15{,}15]\mathrm{A}\cdot\mathrm{m}^2$.
ϕ_3	The current applied to each magnetorquer shall be within the range $[-0.176{,}0.176]\mathrm{A}$.
ϕ_4	The acceleration of each of the reaction wheels shall be within the range $[-0.021{,}0.021]\mathrm{m}/\mathrm{s}^2$.

blocks supported by QVtrace. We manually performed this activity. This activity is time-consuming and error prone. Every time an S-function is replaced by a set of Simulink® blocks, to check for a discrepancy between the behaviors of the S-function and the newly added Simulink® blocks, a set of inputs is generated and, for each input, the outputs produced by the S-function and the Simulink® blocks is compared to check for dissimilarities. We did some testing to check for discrepancies between the behaviors of the S-function and the newly added Simulink® blocks. We did not detect any error. After all the S-function blocks are removed, the model can be loaded in QVtrace, and we can have a formal proof of correctness (or lack thereof) for AC, which is an important component of ESAIL.

However, some requirements may fail to hold on AC when it is evaluated as an independent component, while the same requirements would hold on AC when it is evaluated within the larger model it is extracted from. This is because in the latter case the AC inputs are constrained by the values that can be generated within the larger model, which is not the case when AC is running independently. As a result, we need to verify whether the conditions under which AC works are acceptable given the input values that can be generated by its larger model. This is addressed by learning assumptions guaranteeing that AC satisfies its requirements.

For example, the Simulink® model of the AC is expected to satisfy a number of requirements. Examples of these requirements are described in Table II. The requirement ϕ_1 ensures that AC does not command any torque about the x-axis of the body frame to the reaction wheel, when the satellite is already at the desired attitude. Reaction wheels are used to control the attitude, i.e., the orientation of the satellite, and ensure high pointing accuracy. They generate the twist applied to the satellite around a specific axis by acting on the acceleration of the reaction wheels. To determine whether, or not, AC satisfies requirement ϕ_1, we convert the requirement into a formal property and use QVtrace [28] to verify ϕ_1 over AC. However, it turns out that the requirement ϕ_1 does not hold on AC. Further, using QVtrace, we cannot show that AC satisfies $\neg\phi_1$, indicating that not all of its behaviors violate the requirement of interest. Therefore, for some inputs, AC violates ϕ_1, and for some, it satisfies ϕ_1. Note that if the model satisfies either ϕ_1 or $\neg\phi_1$, there is no need for generating an input assumption.

One of the reasons that the AC does not satisfy ϕ_1 is that, to ensure that the torque applied to the reaction wheel remains within $-0.001\,\mathrm{N}\cdot\mathrm{m}$ and $0.001\,\mathrm{N}\cdot\mathrm{m}$ when the norm of the attitude error quaternion is less than 0.001, we need to constrain the inputs of AC by the following assumption $\mathtt{A}_1$, which we elicited, in collaboration with the ESAIL engineers:

$$\mathtt{A}_1 ::= \forall t \in [0,1] : (\overbrace{\exp(t) + 1 \geq 0}^{\mathtt{P}_1(t)} \wedge \overbrace{\exp(t) - 1 \leq 0}^{\mathtt{P}_2(t)})$$

where:

$$\begin{aligned}\exp(t) = \quad & \overbrace{+783.3 \cdot \omega_e\_\mathrm{x}(t)}^{\mathtt{T1}}\overbrace{-332.6 \cdot \omega_e\_\mathrm{y}(t)}^{\mathtt{T2}}\overbrace{+3.5 \cdot \omega_e\_\mathrm{z}(t)}^{\mathtt{T3}} \\ & - 50.57 \cdot \omega_e\_\mathrm{x}(t) \cdot \omega_e\_y(t) - 4751.8 \cdot \omega_e\_x(t) \cdot \omega_e\_z(t) \\ & + 3588.7 \cdot \omega_e\_y(t) \cdot \omega_e\_z(t) \\ & + 1000 \cdot \mathtt{Rwh\_y}(t) \cdot \omega_e\_z(t) - 1000 \cdot \mathtt{Rwh\_z}(t) \cdot \omega_e\_\mathrm{y}(t) \\ & \underbrace{-54.8 \cdot \omega_e\_y(t)^2}_{\mathtt{T9}}\underbrace{+54.8 \cdot \omega_e\_z(t)^2}_{\mathtt{T10}}\end{aligned}$$

The elicitation of the actual assumption was performed by two of the authors in collaboration with the ESAIL engineers. This was done by analyzing the design document of the satellite. The design document contains the decisions engineers made during the satellite design and specification. The elicitation of the assumption was performed by analyzing the specifications and identifying under which assumption the requirement is satisfied. This was done by computing the transfer function of the system, that describes the input-output relations, and by analytically identifying the most covering assumption that ensures the satisfaction of the requirement. Assumption $\mathtt{A}_1$ constrains the values of the following variables within the time interval of $[0,1]$s: the estimated speed of the satellite (ω_e) and the angular momentum of the reaction wheel (Rwh) over the x-axis (ω_e_x and Rwh_x), the y-axis, (ω_e_y and Rwh_y) and the z-axis (ω_e_z and Rwh_z) of the body frame. This is done by forcing the value of exp to be between -1 and 1. Predicates $\mathtt{P}_1(t)$ and $\mathtt{P}_2(t)$ are composed of the ten terms T1, T2, ..., T10 of exp and the constants 1 and -1. For example, the term T3 of $\mathtt{P}_1(t)$ is $+3.5 \cdot \omega_e\_\mathrm{z}(t)$.

Assumption $\mathtt{A}_1$ is complex, and cannot be learned by our earlier work EPIcuRus [10] since it is a complex function that combines three input signals of ω_e and Rwh with arithmetic operators.

Requirement ϕ_2 in Table II constrains the magnetic moment commanded to the magnetorquers to be in the range $[-15, 15]\mathrm{A}\cdot\mathrm{m}^2$. Requirement ϕ_3 constrains the current applied to the magnetorquers to be in the range $[-0.176, 0.176]\mathrm{A}$. Finally, requirement ϕ_4 constrains the torque acceleration applied to each of the reaction wheels to be in the range $[-0.021, 0.021]\mathrm{m/s}^2$. Those requirements were provided by the manufacturers of the reaction wheel and magnetorquer (see for example [32]). According to the design documents, we expected these requirements to be satisfied for all possible input signals, i.e., without the need of adding any assumption.

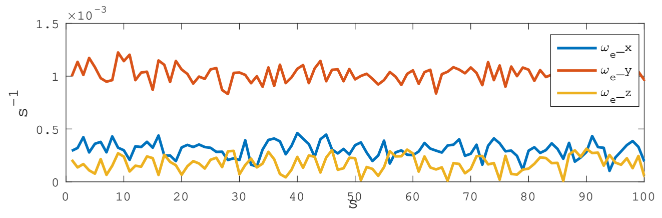
Objective. Without accounting for assumption $\mathtt{A}_1$, we may falsely conclude that the AC model is faulty as it does not satisfy ϕ_1. However, after restricting the inputs of AC with an appropriate assumption, we can show that it satisfies ϕ_1. Hence, there is no fault in the internal algorithm of AC.

In this paper, we extend EPIcuRus to provide an automated approach to infer *complex* environment assumptions for system components such that they, after being constrained by the assumptions, satisfy their requirements. Our extension is applicable under the pre-requisites **Prerequisite-1**, **Prerequisite-2**, and **Prerequisite-3** of EPIcuRus that are summarized in the following.

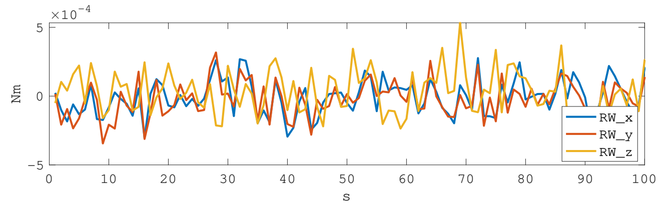
Prerequisite-1. *The component* M *to be analyzed is specified in the Simulink® language.* Simulink® [33], [34] is a well-known and widely-used language for specifying the behavior of cyber-physical systems such as those used in the automotive and aerospace domains. Each Simulink® model has a number of inputs and outputs. We denote a test input for M as $\overline{\mathrm{u}} = \{u_1, u_2 \ldots u_m\}$ where each u_i is a signal for an input of M, and a test output for M as $\overline{\mathrm{y}} = \{y_1, y_2 \ldots y_n\}$ where each y_i is a signal for some output of M. Simulink® models can be executed using a simulation engine that receives a model M and a test input $\overline{\mathrm{u}}$ consisting of signals over a time domain $\mathbb{T}$, and computes the test output $\overline{\mathrm{y}}$ consisting of signals over the same time domain $\mathbb{T}$. A *time domain* $\mathbb{T} = [0, b]$ is a non-singular bounded interval of $\mathbb{R}$. A *signal* is a function $f : \mathbb{T} \to \mathbb{R}$. A *simulation*, denoted by $\mathtt{H}(\overline{\mathrm{u}}, \mathtt{M}) = \overline{\mathrm{y}}$, receives a test input $\overline{\mathrm{u}}$ and produces a test output $\overline{\mathrm{y}}$.

For example, Fig. 1a shows the three input signals of ω_e, i.e., the estimated angular velocity of the satellite, over the time domain $[0, 100]$s, for the AC component. Fig. 1b shows three output signals obtained by simulating the model with these inputs, and representing the torque command (RW) AC applied to the reaction wheels of the satellite over the time domain $[0, 100]$s. Note that, when the behavior of a single component is analyzed, and the component is software, engineers significantly reduce the time domain. Indeed, software components promptly react to input changes. For example, the frequency of execution of AC is 8 Hz, i.e., AC is executed every 125 ms. Therefore, when AC is evaluated, $[0, 1]$s is a reasonable time domain, since it allows executing the control logic of AC eight times.[1] Furthermore, the inputs can be considered constant within this input domain.

[1]The simulation time step is $0.125/4$ms.



(a) Inputs



(b) Outputs

Fig. 1. Example of Input/Output signals of the AC model.

Prerequisite-2. *The requirement ϕ the component has to satisfy is specified in a logical language.* This is to ensure that the requirements under analysis can be evaluated by model checkers or converted into fitness functions required by search-based testing. Both model checking and search-based testing are part of EPIcuRus. In this work, we assume that requirements are expressed in an extension of Metric Temporal Logic (MTL) [35], named Signal Temporal Logic (STL) [36], where MTL propositions are used to constrain the values assumed by the signals over time. This is an expressive language that captures complex properties, such as invariance, stability, responsiveness, smoothness, and fairness.

Prerequisite-3. *The satisfaction of the requirements of interest over the component under analysis can be verified using a model checker.* In this work, we consider QVtrace [28] to exhaustively check whether a model M satisfies the requirement ϕ under the assumption A, i.e., $\langle \mathtt{A} \rangle \mathtt{M} \langle \phi \rangle$. QVtrace takes as input a Simulink® model and a requirement specified in QCT which is a logical language based on a fragment of first-order logic. In addition, QVtrace allows users to specify assumptions using QCT, and to verify whether a given requirement is satisfied for all the possible inputs that satisfy those assumptions. QVtrace uses SMT-based model checking (specifically Z3 BMC [37]) to verify Simulink® models. The QVtrace output can be one of the following: (1) *No violation exists* indicating that the assumption is valid (i.e., $\langle \mathtt{A} \rangle \mathtt{M} \langle \phi \rangle$ holds); (2) *No violation exists for* $0 \leq k \leq k_{max}$ indicating that the model satisfies the given requirement and assumption in the time interval $[0, k_{max}]$. However, there is no guarantee that a violation does not occur after k_{max}; (3) *Violations found* indicating that the assumption A does not hold on the model M; and (4) *Inconclusive* indicating that QVtrace is not able to check the validity of A due to scalability and incompatibility issues.

III. Assumption Generation Problem

In this section, we recall the definition of the assumption generation problem for Simulink® models introduced by Gaaloul et al. [10]. Let M be a Simulink® model. An *assumption* A for M constrains the inputs of M. Each assumption A is represented as a disjunction ($\mathtt{C}_1 \vee \mathtt{C}_2 \vee \ldots \vee \mathtt{C}_n$) of one or more constraints in $\mathcal{C} = \{\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_n\}$. Each constraint in $\mathcal{C}$ is a first-order formula in the following form:

$$\forall \mathtt{t} \in [\tau_1, \tau_1'] : \mathtt{P}_1(\mathtt{t}) \wedge \forall t \in [\tau_2, \tau_2'] : \mathtt{P}_2(\mathtt{t}) \wedge \ldots \wedge \forall t \in [\tau_n, \tau_n'] : \mathtt{P}_n(\mathtt{t})$$

where each $\mathtt{P}_i(\mathtt{t})$ is a predicate over the model input variables, and each $[\tau_i, \tau_i'] \subseteq \mathbb{T}$ is a time domain. The predicate $\mathtt{P}_i(\mathtt{t})$ is in the form $exp_s \sim 0$ where exp_s is an arithmetic expression over constant and signal values and $\sim \in \{<, \leq, >, \geq, =\}$. Recall from Section II that $\mathbb{T} = [0, 1]$s is the time domain used to simulate the model M. An example constraint for the AC model is the constraint $\mathtt{C}_1$ defined as follows:

$$\mathtt{C}_1 ::= \forall t \in [0, 1] : (1.0 \cdot \omega_e\_\mathrm{x}(t) + 1.0 \cdot \omega_e\_\mathrm{y}(t) - 0.0011) \leq 0$$

$\mathtt{C}_1$ constrains the sum of the values of two input signals $\omega_e\_\mathrm{x}(t)$ and $\omega_e\_\mathrm{y}(t)$ of the input ω_e of the AC model over the time domain $[0, 1]$s. These signals represent, respectively, the angular speed of the satellite over the x and y axes of the body frame.

Let $\overline{\mathtt{u}}$ be a test input for a Simulink® model M, and let C be a constraint over the inputs of M. We write $\overline{\mathtt{u}} \models \mathtt{C}$ to indicate that the input $\overline{\mathtt{u}}$ satisfies the constraint C. For example, the input $\overline{\mathtt{u}}$ for the AC model, which is described in Fig. 1, satisfies the constraint $\mathtt{C}_1$. Note that for Simulink® models, test inputs are described as functions over a time domain $\mathbb{T}$, and similarly, we define constraints C as a conjunction of predicates over the same time domain or its subdomains.

Let $\mathtt{A} = \mathtt{C}_1 \vee \mathtt{C}_2 \vee \ldots \vee \mathtt{C}_n$ be an assumption for model M, and let $\overline{\mathtt{u}}$ be a test input for M. The input $\overline{\mathtt{u}}$ satisfies the assumption A if $\overline{\mathtt{u}} \models \mathtt{A}$. For example, consider the assumption

$$\mathtt{A}_2 = \mathtt{C}_1 \vee \mathtt{C}_2$$

of AC where:

$$\mathtt{C}_1 ::= \forall t \in [0, 1] : (1.0 \cdot \omega_e\_\mathrm{x}(t) + 1.0 \cdot \omega_e\_\mathrm{y}(t) - 0.0011) \leq 0$$
$$\mathtt{C}_2 ::= \forall t \in [0, 1] : (1.0 \cdot \omega_e\_\mathrm{x}(t) + 1.0 \cdot \omega_e\_\mathrm{y}(t) + 1 \cdot \omega_e\_\mathrm{z}(t) - 0.0025) \leq 0$$

The input $\overline{\mathtt{u}}$ in Fig. 1 satisfies the assumption $\mathtt{A}_2$ since it satisfies the constraints $\mathtt{C}_1$ and $\mathtt{C}_2$.

Let A be an assumption, and let $\mathcal{U}$ be the set of all possible test inputs of M. We say $\mathtt{U} \subseteq \mathcal{U}$ is a *valid input set* of M restricted by the assumption A if for every input $\overline{\mathtt{u}} \in \mathtt{U}$, we have $\overline{\mathtt{u}} \models \mathtt{A}$. Let ϕ be a requirement for M that we intend to verify. For every test input $\overline{\mathtt{u}}$ and its corresponding test output $\overline{\mathtt{y}}$, we denote as $[\![\overline{\mathtt{u}}, \overline{\mathtt{y}}, \phi]\!]$ the Boolean verdict indicating whether ϕ is satisfied or violated when M is executed for test input $\overline{\mathtt{u}}$.[2]

Definition 1: Let A be an assumption, let ϕ be a requirement for M, let $\mathtt{U} \subseteq \mathcal{U}$ be a valid input set of M restricted by assumption A. We say the satisfaction of the requirement ϕ over model M restricted by the assumption A is v, i.e., $\langle \mathtt{A} \rangle \mathtt{M} \langle \phi \rangle = v$, if

$$v = \min_{\overline{\mathtt{u}} \in \mathtt{U}} [\![\overline{\mathtt{u}}, \overline{\mathtt{y}}, \phi]\!]$$

where $\overline{\mathtt{y}}$ is the test output generated by the test input $\overline{\mathtt{u}} \in \mathtt{U}$. For computing the min, we assume that *true* > *false*.

[2]The Boolean verdict is computed using existing approaches that extract the degree of violation or satisfaction of a property ϕ using existing techniques from the literature [24], [38], [39].

Definition 2: We say an assumption $\mathtt{A}$ is sound[3] for a model $\mathtt{M}$ and its requirement ϕ, if $\langle\mathtt{A}\rangle\mathtt{M}\langle\phi\rangle = \mathit{true}$.

For a given model $\mathtt{M}$ and a requirement ϕ, we may have several assumptions that are sound. We are typically interested in identifying the sound assumption that leads to the largest valid input set $\mathtt{U}$, and hence is less constraining. Let $\mathtt{A}_1$ and $\mathtt{A}_2$ be two different sound assumptions for a model $\mathtt{M}$ and its requirement ϕ, and let $\mathtt{U}_1$ and $\mathtt{U}_2$ be the valid input sets of $\mathtt{M}$ restricted by the assumptions $\mathtt{A}_1$ and $\mathtt{A}_2$. In our previous work [10], we defined $\mathtt{A}_1$ to be more informative than $\mathtt{A}_2$ if $\mathtt{A}_2 \Rightarrow \mathtt{A}_1$. However, this definition only allows to compare assumptions when there exists a logical implication between the two. To enable a wider comparison among assumptions in this work, we define a notion of *coverage*. We say that $\mathtt{A}_1$ has a larger *coverage* than $\mathtt{A}_2$ if $|\mathtt{U}_1| > |\mathtt{U}_2|$ where $|\mathtt{U}_1|$ and $|\mathtt{U}_2|$ are, respectively, the cardinalities of the valid input sets $\mathtt{U}_1$ and $\mathtt{U}_2$ associated with $\mathtt{A}_1$ and $\mathtt{A}_2$ (i.e., the number of inputs in the sets $\mathtt{U}_1$ and $\mathtt{U}_2$). Our definition is generic and the sets can contain infinitely many inputs. What is important is to have a metric space, i.e., a set together with a metric on the set. For example, a real interval $[0, 1]$ is a metric space. Given a real interval $[0, 1]$, the "size" (or "measure" or "length") of the interval is 1 (the metric is the difference between the upper and the lower bound of the interval). This definition can be generalized to hypervolumes of polytopes, by extending the metric from a one-dimensional space to a multidimensional space. Our notion of coverage is compliant with the notion of logical implication. Given two assumptions A_1 and A_2, such that $A_2 \Rightarrow A_1$, the assumption A_1 has a larger coverage than A_2 since it has a larger input set. Therefore, a more informative assumption (according to logical implication) has a larger coverage, i.e., a larger valid input set. The notion of coverage, however, is not limited to logical implication, and we can use it to compare assumptions that are not logically related, which is necessary in our context. We define the size of valid test inputs for an assumption as the number of valid test inputs within the test input set according to a given metric.

Note that computing the size of valid test inputs for our assumptions which are first-order formulas is in general infeasible. As we will discuss in Section VI, we provide an approximative method to compare the size of valid test inputs to be able to compare a given pair of assumptions based on our proposed coverage measure.

In practical applications, there is an intrinsic tension between coverage and soundness. The larger the coverage of the assumptions, the higher the chance that the assumptions are unsound. This is because by increasing the coverage of the assumptions, they may turn unsound. For example, suppose the actual assumption is $x < 2$ (i.e., a sound assumption with an optimal coverage). The assumption $x < 1$ is also sound but has suboptimal coverage. In an attempt to increase coverage, we may consider the assumption $x < 2.1$, which is unsound. Therefore, in industrial applications, users should find a practical *tradeoff* between coverage and soundness. We will evaluate this tradeoff for our satellite case study in Section VI.

In this paper, provided with a model $\mathtt{M}$, a requirement ϕ and a desired value v, our goal is to generate the sound assumption that provides the largest coverage. We note that our approach, while guaranteeing the generation of sound assumptions, does not guarantee that the generated assumptions have the largest coverage. Instead, we propose heuristics to maximize the chances of generating assumptions with the large coverage and evaluate our heuristics empirically in Section VI.

[3]Called v-safe in our previous work [10], where we define the degree of satisfaction v.

IV. EPIcuRus Overview

EPICURUS aims at solving the assumption generation problem. Fig. 2 shows an overview of EPIcuRus, which takes as input a Simulink® model $\mathtt{M}$ and a requirement ϕ, and computes an assumption ensuring that the model $\mathtt{M}$ satisfies the requirement ϕ when the assumption holds. The EPIcuRus components are reported in boxes labeled with blue background numbers. The *sanity check* (1) verifies whether the requirement ϕ is satisfied (or violated) on $\mathtt{M}$ for all the inputs and therefore the assumption should not be computed. If the requirement is neither satisfied nor violated for all inputs, EPIcuRus iteratively performs the three steps of the EPIcuRus Loop (see Algorithm 1) discussed in the following:

2 *Test generation* (GenSuite): returns a test suite $\mathtt{TS}$ of test cases that exercise $\mathtt{M}$ with respect to requirement ϕ. The goal is to generate a test suite $\mathtt{TS}$ that includes both passing (i.e., satisfying ϕ) and failing (i.e., violating ϕ) test cases;

3 *Assumption generation* (GenAssum): uses the test suite $\mathtt{TS}$ to compute an assumption $\mathtt{A}$ such that $\mathtt{M}$ restricted by $\mathtt{A}$ is likely to satisfy ϕ;

4 *Model checking* (Check): checks whether $\mathtt{M}$ restricted by $\mathtt{A}$ satisfies ϕ. We use the notation $\langle\mathtt{A}\rangle\mathtt{M}\langle\phi\rangle$ (borrowed from the compositional reasoning literature [9]) to indicate that $\mathtt{M}$ restricted by $\mathtt{A}$ satisfies ϕ. If our model checker can assert $\langle\mathtt{A}\rangle\mathtt{M}\langle\phi\rangle$, a sound assumption is found.

There are two stopping criteria that can be selected for EPIcuRus. The first stopping criterion stops EPIcuRus whenever the model checker can assert $\langle\mathtt{A}\rangle\mathtt{M}\langle\phi\rangle$. The second stopping criterion constrains the timeout and allows EPIcuRus to refine the computed assumption over consecutive iterations. The impact of the selection of the stopping criteria on the assumption produced by EPIcuRus is discussed in Section III.

A high-level description of each of these steps is presented in the following, a detailed description of the assumption generation procedures proposed in this work is provided in Section V.

A. Sanity Check

The sanity check (1) verifies whether the requirement ϕ is satisfied or violated for all the inputs. To do so, we use a model checker to respectively verify whether $\langle\top\rangle\mathtt{M}\langle\phi\rangle$ or $\langle\top\rangle\mathtt{M}\langle\neg\phi\rangle$ is true. We use the symbol $\top$ to indicate that no assumption is considered. If the requirement is satisfied for all inputs, no assumption is needed. If the requirement is violated for all inputs, then the model is faulty, and an assumption cannot be computed as there is no input that satisfies the requirement. The requirement *passes* the sanity check if some

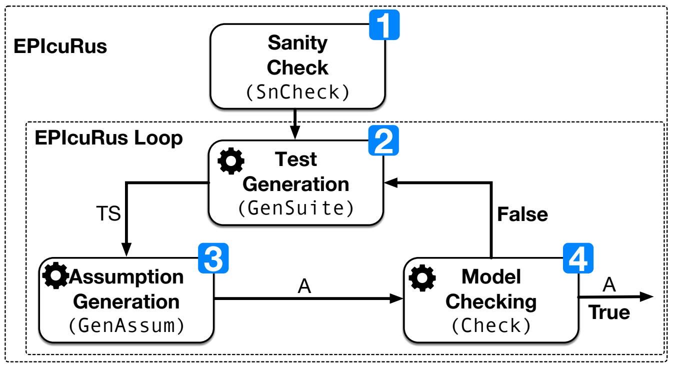


Fig. 2. EPIcuRus framework overview.

Algorithm 1 EPIcuRus Loop.

Inputs. M: the Simulink® model
ϕ: requirement of interest
opt: options
Outputs. A: assumption

```
1: function A=EPICURUSLOOP(M, $\phi$, opt)
2:   TS=[]; A=null;                        ▷ Variables Initialization
3:   do
4:     TS=GENSUITE(M, $\phi$, TS, opt)      ▷ Test Generation
5:     A=GENASSUM(TS,opt);                  ▷ Assum.Gen.
6:     A= CHECK(A, M, $\phi$)               ▷ Model Checking
7:   while not opt.Stop_Crt
8:   return A;
9: end function
```

inputs satisfy ϕ while others violate it, i.e., the requirement is neither satisfied nor violated for all inputs. In that case, the EPIcuRus loop is executed to compute an assumption.

We use QVtrace to implement our sanity check. QVtrace exhaustively verifies whether a Simulink® model M satisfies a requirement ϕ expressed in the QCT language, for all the inputs that satisfy the assumption A, i.e., $\langle \mathtt{A}\rangle \mathtt{M}\langle\phi\rangle$.

- To check whether the model M satisfies the requirement ϕ for all inputs, we check whether $\langle\top\rangle\mathtt{M}\langle\phi\rangle$. QVtrace generates four kinds of outputs (see Section II). When it returns "No violation exists", or "No violation exists for $0 \leq \mathrm{k} \leq \mathrm{k}_{max}$", we conclude that the model under analysis satisfies the given formal requirement ϕ without the need to consider assumptions.
- To check whether the model M violates requirement ϕ for all inputs, we check $\langle\top\rangle\mathtt{M}\langle\neg\phi\rangle$. If QVtrace returns "No violation exists", or "No violation exists for $0 \leq \mathrm{k} \leq \mathrm{k}_{max}$", we conclude that since $\neg\phi$ is satisfied, the model under analysis does not show any behavior that satisfies the requirement ϕ. Thus, the requirement ϕ is violated for any possible input, and the model is faulty.

In the two previous cases, EPIcuRus provides the user with a value indicating that all the outputs of the model either satisfy or violate ϕ. Otherwise, the EPIcuRus loop is executed to compute an assumption.

B. Test Generation

The goal of the test generation step (2) is to generate a test suite TS of test cases for M such that some test inputs lead to the violation of ϕ and some lead to the satisfaction of ϕ. Note that, while inputs that satisfy and violate the requirement of interest can also be extracted using model checkers, due to the large amount of data needed by machine learning (ML) to derive accurate assumptions, we rely on simulation-based testing for data generation. Further, it is usually faster to simulate models rather than to model check them. For example, performing a single simulation of AC and evaluating the satisfaction of ϕ_1 on the generated output takes 0.9s, while model checking AC against ϕ_1 takes approximately 21.06s. Hence, given a specific time budget, simulation-based testing leads to the generation of a larger amount of data compared to using model checking for data generation.

We use search-based testing techniques [40]–[42] for test generation and rely on simulations to run the test cases. Search-based testing allows us to guide the generation of test cases in very large search spaces. It further provides the flexibility to tune and guide the generation of test inputs based on the needs of our learning algorithm. For example, we can use an explorative search strategy if we want to sample test inputs uniformly or we can use an exploitative strategy if our goal is to generate more test inputs in certain areas of the search space. For each generated test input, the underlying Simulink® model is executed to compute the output. To generate input signals, we use the approach of Matlab [43], [44] that encodes signals using some parameters. Specifically, each signal u_u in $\overline{\mathrm{u}}$ is captured by an input profile $\langle int_u, R_u, n_u\rangle$, where int_u is the interpolation function, $R_u \subseteq \mathbb{R}$ is the input domain, and n_u is the number of control points. We assume that the number of control points is equal for all the input signals. Provided with the values for these three parameters, we generate a signal over time domain $\mathbb{T}$ as follows:

1) we generate n_u control points, i.e., $c_{u,1}$, $c_{u,2}$, …, c_{u,n_u}, equally distributed over the time domain $\mathbb{T} = [0, b]$, i.e., positioned at a fixed time distance $I = \frac{b}{n_u - 1}$. Let $c_{x,y}$ be a control point, x is the signal the control point refers to, and y is the *position* of the control point. The control points $c_{u,1}, c_{u,2}, \ldots, c_{u,n_u}$ respectively contain the values of the signal u at time instants $0, I, 2 \cdot I, \ldots, (n_u - 2) \cdot I, (n_u - 1) \cdot I$;
2) we assign randomly generated values within the domain R_u to each control point $c_{u,1}$, $c_{u,2}$, …, c_{u,n_u}; and
3) we use the interpolation function int_u to generate a signal that connects the control points. The interpolation functions provided by Matlab include, among others, linear, piecewise constant and piecewise cubic interpolations, but the user can also define custom interpolation functions.

To generate realistic inputs, the engineer should select an appropriate value for the number of control points (n_u) and choose an interpolation function that describes with reasonable accuracy the overall shape of the input signals for the model under analysis. Based on these inputs, the test generation procedure has to select which values $c_{u,1}$, $c_{u,2}$, …, c_{u,n_u} to assign to the control points for each input u_u.

The verdict of the requirement of interest (ϕ) is then evaluated by (i) using the values assigned to the control points and the interpolation functions to generate input signals $\overline{\mathtt{u}}$; (ii) simulating the behavior of the model for the generated input signals and recording the output signals $\overline{\mathtt{y}} = \mathtt{H}(\overline{\mathtt{u}}, \mathtt{M})$; (iii) evaluating the satisfaction $[\![\overline{\mathtt{u}}, \overline{\mathtt{y}}, \phi]\!]$ of ϕ on the output signals; and (iv) labelling the test case with the verdict value (*pass* or *fail*) depending on whether $[\![\overline{\mathtt{u}}, \overline{\mathtt{y}}, \phi]\!]$ is *true* or *false*.

The test generation step returns a test suite TS containing a set of test cases, each of which containing the values assigned to the control points of each input signal and the verdict value. Depending on the algorithm used to learn the assumption, EPIcuRus may or may not reinitialize the test suite TS at each iteration. In the latter case, the test cases that were generated in previous iterations remain in the new test suite TS that is also expanded with new test cases.

C. Assumption Generation

Given a requirement ϕ and a test suite TS, the goal of the assumption generation step is to infer an assumption A such that M restricted based on A is likely to satisfy ϕ. We use ML to derive an assumption based on test inputs labelled by binary verdict values. Specifically, the assumption generation procedure infers an assumption by learning patterns from the test suite data (TS). This is done by (i) running the ML algorithm that extracts an assumption defined over the control points of the input signals of the model under analysis; and (ii) transforming the assumption defined over the values assigned to the control points into an assumption defined over the values of the input signals such that it can be checked by QVtrace.

Different ML techniques used in the assumption generation step lead to different versions of EPIcuRus. In this work, we consider Decision Trees (DT), Genetic Programming (GP), and Random Search (RS) as alternative assumption generation policies. DT was recently used by Gaaloul et al. [10], while GP, and RS are described in Section V and are part of the contributions of this work.

D. Model Checking

This step checks whether the assumption A generated by the assumption generation step is sound. Note that the ML technique used in the assumption generation step, being a non-exhaustive learning algorithm, cannot ensure that assumption A guarantees the satisfaction of ϕ for M. Hence, in this step, we use a model checker for Simulink® models to check whether M, restricted by A, satisfies ϕ, i.e., whether $\langle \mathtt{A} \rangle \mathtt{M} \langle \phi \rangle$ holds. QVtrace returns four possible results; "*No violation exists*", "*No violation exists for* $0 \leq k \leq k_{max}$", "*Violations found*", and "*Inconclusive*". Specifically, when QVtrace returns "*No violation exists*" or "*No violation exists for* $0 \leq k \leq k_{max}$", we conclude that assumption A is sound, and hence ensures that M satisfies requirement ϕ. When QVtrace returns "*Violations found*", we conclude that A is *theoretically* unsound, since M violates requirement ϕ under the assumption A. When QVtrace returns "*Inconclusive*", the assumption can neither be proven sound nor theoretically unsound. In the remaining sections, we use the following terminology:

1) "*an assumption is sound*" denotes the cases in which the verdict "*No violation exists*" or "*No violation exists for* $0 \leq k \leq k_{max}$" is returned by the model checker, i.e., it is proven that the assumption is sound.
2) "*an assumption is theoretically unsound*" denotes the cases in which the "*Violations found*" verdict is returned by the model checker, i.e., it is proven that the assumption is unsound.
3) "*an assumption is inconclusive*" denotes the cases in which "*Inconclusive*" verdicts are returned

Algorithm 2 Genetic Programming (GP)

Inputs. TS: the test suite
    opt: values of the parameters of GP (Table III)
Outputs. A: assumption

```
1:  function A=GenAssum(TS,opt)
2:    t=0;
3:    P_0=Initialize(TS,opt);            ▷ Initialize Population
4:    P_0.Fit=Evaluate(TS,P_0);         ▷ Assumptions Evaluation
5:    while t<opt.Gen_Size do
6:      Off=Breed(P_t,opt);              ▷ New Offspring
7:      Off.Fit=Evaluate(TS,Off);        ▷ Evaluation
8:      P_{t+1}=Off;                     ▷ New Population
9:      t=t+1;
10:   endwhile
11:   A=BestAssum(P_0,...,P_{t+1});      ▷ Get Best Assum.
12:   return A;
13: end function
```

V. Assumption Generation Procedures

In this section, we describe our solution for learning assumptions. For a given Simulink® model M, we generate assumptions over individual control point variables of the signal inputs of M (Section V-A). We then provide a procedure to lift the generated assumptions that are defined over control points to those defined over signal variables (Section V-B). Our algorithm to generate assumptions uses Genetic Programming (GP) because we want to generate complex assumptions composed of arbitrary linear and non-linear arithmetic formulas. In addition, we introduce a baseline algorithm using Random Search (RS) for generating assumptions.

A. Learning Assumptions on Control Points with GP

Genetic programming (GP) is a technique for evolving programs from an initial randomly generated population in order to find fitter programs (i.e., those optimizing a desired fitness function). In our work, we use Strongly Typed Genetic Programming (STGP) [12], [17], a variation of GP designed to ensure that all the individuals within a population follow a set of syntactic rules specified by a grammar. The steps of our GP procedure are summarized by Algorithm 2. First (Initialize), the algorithm creates an initial population containing a set of possible solutions (a.k.a. individuals). A fitness measure is

TABLE III
PARAMETERS OF EPICURUS (EP) AND ITS GENETIC PROGRAMMING ALGORITHM (GP).

	Parameter	Description	Parameter	Description
EP	`SBA`	Search-based algorithm (`GP`, `DT`, `RS`).	`ST`	Simulink® simulation time.
	`TS_Size`	The number of the generated test cases per iteration.	`Stop_Crt`	Stopping criteria: sound assumption found (`MC`) or timeout (`Timeout`).
	`Timeout`	EPIcuRus timeout.	`Nbr_Runs`	Number of experiments to be executed.
GP	`Max_Conj`	Maximum number of conjunctions in an assumption.	`Max_Disj`	Maximum number of disjunctions in an assumption.
	`Const_Min`	Minimum constant value.	`Const_Max`	Maximum constant value.
	`Max_Depth`	Maximum depth of the syntax tree.	`Init_Ratio`	Percentage of the assumptions copied from the last population.
	`Pop_Size`	Number of individuals per population.	`Gen_Size`	Number of generations.
	`Sel_Crt`	The selection criterion.	`T_Size`	The number of individuals chosen for the tournament selection.
	`Mut_Rate`	Probability of applying the mutation operator.	`Cross_Rate`	Probability of applying the crossover operator.

```
or-exp   ::=  or-exp ∨ or-exp | and-exp
and-exp :: = and-exp ∧ and-exp | rel
rel        :: = exp (< | ≤ | > | ≥ | = ) 0
exp       :: = exp (+| − | ∗ |/ ) exp | const | cp
```

Fig. 3. Syntactic rules of the grammar that defines the assumptions on control points. The symbol | separates alternatives, const is a constant value, and cp is a variable that refers to a control point.

used to assess how well each individual solves the problem (EVALUATE). In the evolutionary part, the algorithm iteratively generates new populations (BREED). It extracts a set of parents individuals and generates an offspring set by applying genetic operators to the parents. The algorithm then evaluates the individuals of the offspring (EVALUATE) and uses the offspring set as the new population $\mathtt{P}_{t+1}$ for the next iteration. The breeding and evaluation steps are repeated for a given number of generations (`opt.Gen_Size`). Then, the algorithm finds among all the individuals of the generated populations the individual with the highest fitness (BESTASSUM). The algorithm returns the individual with the highest fitness (`A`).

In the following, we describe how we use Algorithm 2 to generate assumptions over individual control point variables of the input signals of `M`.

Representation of the Individuals. Each individual represents an assumption over individual control point variables of the input signals of `M`. Specifically, assumptions are defined according to the syntactic rules of the grammar provided in Fig. 3. Furthermore, we constrain each arithmetic expression to contain only signal control points in the same position. For example, the assumption

$$(c_{u_1,1} - c_{u_2,1} - 20 \leq 0) \vee ((c_{u_1,2} < 0) \wedge (c_{u_2,2} - 2.5 = 0))$$

is defined according to the grammar in Fig. 3. It constrains the values of the control points $c_{u_1,1}$, $c_{u_2,1}$, $c_{u_1,2}$, and $c_{u_2,2}$, and each arithmetic expression contains control points that refer to the same position. For example, $c_{u_1,1} + c_{u_2,1}$ contains the signal control points $c_{u_1,1}$, $c_{u_2,1}$ of the input signals u_1 and u_2 in position 1.

Initial Population. The initial population contains a set of `Pop_Size` individuals. The population size remains the same throughout the search. The method INITIALIZE generates the initial population. Recall from Section IV (see Algorithm 1) that GENASSUM is called within EPIcuRus iteratively.

The first time that INITIALIZE is called it generates an initial set of randomly generated individuals. We use the grow method [13] to randomly generate each individual in the initial population. The grow method generates a tree with a maximum depth `Max_Depth`. It first creates the root node of the tree labeled with the Boolean operator ∨ or ∧ or one among the relational operators $<,\leq,>,\geq$ and $=$. Then, it iteratively generates the child nodes as follows. If the node is not a terminal, the algorithm considers the production rule of the in Fig. 3 associated with the node type. One among the alternatives specified on the right side of the production rule is randomly selected and used to generate the child nodes. Then, the child nodes are considered. If the node is a terminal, depending on whether the node type is const or cp, a random constant value within the range $[\mathtt{Const\_Min}, \mathtt{Const\_Max}]$ or signal control point is chosen with equal probability among the set of all the control points, respectively. To ensure that the generated tree does not exceed the maximum depth (`Max_Depth`) the algorithm constrains the type of the nodes that can be considered as the size of the tree increases. The algorithm also forces each arithmetic expression (exp) to only use control points in the same position. When all the nodes are considered the individual is returned.

In the subsequent iterations, however, INITIALIZE copies a subset of individuals from the last population generated by the previous execution of GENASSUM. The number of copied individuals is determined by the initial ratio (`Init_Ratio`) in its initial population and then randomly generates the remaining elements required to reach the size `Pop_Size`. This allows GP to reuse some of the individuals generated previously instead of starting from a fully random population each time it is executed.

Fitness Measure. Our fitness measure is used by the EVALUATE method to assess the soundness and coverage of the assumption individuals in the current populations (see Section III). The fitness measure relies on the test cases contained in the test suite `TS`. We say a test case `tc` in `TS` satisfies an assumption `A` if `tc` is a satisfying value assignment for `A`. We compute the number of passing test cases in `TS` that satisfy `A` and denote it by `TP`. We also compute the number of failed test cases in `TS` that satisfy `A` and denote it by `TN`. We then compute the sound degree of an assumption `A` as follows:

$$\mathit{sound} = \frac{\mathtt{TP}}{\mathtt{TP} + \mathtt{TN}}$$

The variable *sound* assumes a value between 0 and 1. The higher the value, the more passing test cases in `TS` satisfy the assumption. When *sound* = 1, all test cases in `TS` that satisfy the assumption lead to a *pass* verdict. Since there is no failing

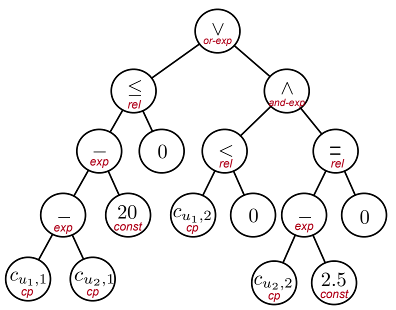


Fig. 4. Syntax tree associated with an individual.

test case in TS that satisfies the assumption, the assumption is likely to be *sound*.

To measure the coverage of an assumption, we compute the ratio of test cases in TS satisfying the assumption (TP+TN) over the total number of test cases in TS:

$$\textit{coverage} = \frac{\texttt{TP} + \texttt{TN}}{\texttt{TS}}$$

The higher *coverage*, the weaker the assumption, and the more useful it is when informing engineers about when a requirement is satisfied.

Having computed *sound* and *coverage* for an assumption A, we compute the fitness function for A as follows:

$$\textsc{Fn} = \textit{sound} + \lfloor \textit{sound} \rfloor \cdot \textit{coverage}$$

where $\lfloor \rfloor$ is the Matlab floor operator [45]. This operator returns 0 for all the values of *sound* within the interval $[0, 1)$ and returns 1 if *sound* is equal to 1. Function FN returns the *sound* value when *sound* is within the interval $[0, 1)$. When *sound* is equal 1 it returns the value $1 +$ *coverage*. Intuitively, FN starts considering the coverage value only when an assumption is sound. This fitness function guides the search toward the detection of sound assumptions (which is our primary goal) that provide larger coverage.

We note that our fitness provides one way to combine *sound* and *coverage* values that fitted our needs. Developers may identify other ways to combine these two values to prioritize either *sound* assumptions or assumptions with large *coverage*.

Parents Selection. It uses the fitness values to select parent individuals for crossover and mutation operations that will be used to generate a new population. We implemented the following standard selection criteria of GP: *Roulette Wheel Selection (RWS)*, *TouRnament Selection (TRS)* and *Rank Selection (RS)* [46].

Genetic Operators. The genetic operators of GP act on the syntax tree of individuals. For example, the syntax tree of

$$(c_{u_1,1} - c_{u_2,1} - 20 \leq 0) \vee ((c_{u_1,2} < 0) \wedge (c_{u_2,2} - 2.5 = 0))$$

is shown in Fig. 4. Each node of the syntax tree represents a portion of the individual and is labeled (italic red label) with the identifier of the corresponding syntactic rule of the grammar of Fig. 3.

The BREED method generates an offspring by

- either applying the *crossover* operator to generate two new individuals (with probability Cross_Rate) or randomly selecting an individual from $\texttt{P}_t$; and
- applying the *mutation* operator (with probability Mut_Rate) to the individuals returned by the previous step.

The *crossover* and *mutation* genetic operators are summarized in the following.

We use one-point crossover [47] as *crossover* operator. One-point crossover (i) randomly selects two parent individuals; (ii) randomly selects one subtree in each parent; and (iii) swaps the selected subtrees resulting in two child individuals. To ensure that the child individuals are compliant with our representation, we force the following constraints to hold:

- The type of the root nodes of the subtrees is the same;
- The depth of the child individuals does not exceed Max_Depth;
- The number of conjunctions and disjunctions of the child individuals does not exceed Max_Conj and Max_Disj, respectively;
- When the type of the root nodes of the subtrees is exp, all the signal control points of the subtrees are in the same positions.

We use point mutation [48] as a *mutation* operator. Point mutation mutates a child individual by randomly selecting one subtree and replacing it with a randomly generated tree. To create the randomly generated tree we adopt the procedure used within the INITIALIZE method. Additionally, to ensure that the mutated child individual is compliant with our representation, our implementation ensures the constraints specified for the crossover operator are also satisfied here.

The full set of GP parameters is summarized in Table III.

Random Search. It proceeds following the steps of Algorithm 2. However, at each iteration, a new set of individuals is randomly generated by adopting the same procedure used within the INITIALIZE method.

B. Control Points-Based to Signal-Based Assumptions

To use assumptions in QVtrace, it is necessary to translate assumptions that constrain control point values to assumptions that constrain signal values. To do so, we proceed as follows. Recall that control points $c_{u,1}$, $c_{u,2}$, $\ldots c_{u,n_u-1}$, c_{u,n_u} are respectively positioned at time instants $0, I, 2 \cdot I, \ldots, (n_u - 2) \cdot I, (n_u - 1) \cdot I$ and that any arithmetic expression contains only signal control points in the same position. Each expression exp that constrains the values of control points in position j is translated into an expression $\forall t \in [(j-1) \cdot I, j \cdot I) : \texttt{exp}'$ where $\texttt{exp}'$ is obtained by substituting each control point $c_{u_x,j}$ with the expression $u_x(t)$ modeling the input signal u_x at time t. Intuitively, this substitution specifies that the expression exp holds within the entire time interval $[(j-1) \cdot I, j \cdot I)$. For example, assuming that the control points 1, 2 and 3 are respectively positioned at time instant 0, 5 and 10, the assumption on the control points

$$(c_{u_1,1} - c_{u_2,1} - 20 \leq 0) \vee ((c_{u_1,1} < 0) \wedge (c_{u_2,2} - 2.5 = 0))$$

is translated into an assumption over the signal variables as follows:

$$(\forall t \in [0, 5) : u_1(t) - u_2(t) - 20 \leq 0) \;\; \vee$$

$((\forall t \in [0,5) : u_1(t) < 0) \ \wedge \ (\forall t \in [5,10) : u_2(t) - 2.5 = 0))$

Note that, our translation does not use the interpolation function of the input profile (see Section IV-B). Indeed, considering more complex interpolation functions may lead to assumptions that are less comprehensible and contain arithmetic functions that are more complex to interpret since they also relate signal values at different time instants. However, our approach can be extended to also consider the input profile for the translation from control point-based assumptions to signal-based ones.

VI. Evaluation

In this section, we evaluate our contributions by answering the following research questions:

• *RQ1 (Comparison of the search-based techniques). How does GP compare with DT and RS in generating sound assumptions over signal variables with a high coverage? (Section VI-A)*

To answer this question, we compared the different search-based techniques of EPIcuRus and empirically assessed whether GP learns sound assumptions that have a larger coverage than the ones learned by DT and RS. We are not aware of any tool other than EPIcuRus for computing signal-based assumptions that we could use as a baseline of comparison. To answer this question we relied on a public-domain set of representative models of CPS components [49] from Lockheed Martin [18]—a company working in the aerospace, defense, and security domains— and the model of our satellite case study (`AC`). Recall that EPIcuRus targets individual components, that can be analyzed using a model checker, and is generally not applicable to the entire industrial CPS models, such as the `ADCS` model (see sections I and II).

• *RQ2 (Usefulness). How useful are the assumptions learned by EPIcuRus?*

To answer this question, we empirically assessed whether EPIcuRus can learn assumptions that specify valid inputs of a CPS component by comparing them with assumptions engineers would manually develop based on their domain knowledge and without any automated assistance. We answer this question by using our best search technique, according to RQ1 results, and the `AC` component of `ADCS` since 1) this is a representative example of an industrial CPS component (see Section II), and 2) we could interact with the engineers that developed the `AC` component to evaluate how the assumptions computed by EPIcuRus compare with the assumptions they would manually write. To answer our research question, we considered the requirement ϕ_1 of `AC` (see Section II) since EPIcuRus confirmed that the requirements ϕ_2, ϕ_3, and ϕ_4, are satisfied for all possible input signals. Since there is, when dealing with complex components, a *tradeoff* among the coverage of the assumptions returned by EPIcuRus and the capability of QVtrace to confirm their soundness (see Section III), engineers often have the choice to either learn assumptions with large coverage, whose soundness cannot be confirmed by a solver like QVtrace, or alternatively learn simpler assumptions, which have lower coverage but whose soundness can be verified exhaustively. Our goal is to investigate such tradeoff when analyzing industrial CPS components. Therefore, to answer RQ2, we are considering two sub-questions:

- *RQ2-1. How useful are EPIcuRUS assumptions when demonstrating their soundness is not a priority? (Section VI-B)*
- *RQ2-2. How useful are EPIcuRUS assumptions when learning assumptions whose soundness can be verified is prioritized? (Section VI-C)*

Implementation and Data Availability. We extended the original Matlab implementation of EPIcuRus [10]. We decided to implement the procedure presented in Section V by reusing existing tools. Among the many tools available in the literature (e.g., Weka [50], GPLAB [51], GPTIPS [52], Matlab GP toolbox [15], [16]), we decided to rely on tools developed in Matlab. This facilitates the integration of our extension within EPIcuRus, and restricted our choice to GPLAB, GPTIPS, and the Matlab GP toolbox. Among these, we implemented our techniques on the top of GPLAB. We chose GPLAB since it allows the introduction of new genetic operators by adding new functions. We exploited this feature to implement the genetic operators of the procedure presented in Section V. Our implementation and results are publicly available [53].

A. RQ1 — Comparison of the Search-Based Techniques

To compare GP, DT, and RS, we considered 12 models of CPS components and 94 requirements [49]. These models include 11 models developed by Lockheed Martin [18] and the model of our satellite case study (`AC`). The models and requirements from Lockheed Martin were also recently used to compare model testing and model checking [54] and to evaluate our previous version of EPIcuRus [10]. Table IV contains the description, number of blocks, inputs, and outputs of each CPS component model. It also contains the simulation time and the number of requirements considered for each model.

Out of the 94 requirements, 27 could not be handled by QVtrace (violating **Prerequisite-3**). For 16 requirements, the Simulink® models of the CPS components were not supported by QVtrace. For 11 requirements, QVtrace returned an inconclusive verdict due to scalability issues. For 48 of the 67 requirements that can be handled by QVtrace, EPIcuRus did not pass the sanity check: QVtrace could prove 47 requirements and refuted one requirement. Therefore, to answer research question RQ1, we considered the 19 requirements of four models , that can be handled by QVtrace, pass the sanity check, and required the assumption generation procedure to be executed (column `#Reqs` of Table IV within round brackets).

Methodology and Experimental Setup. To answer our research question, we configured the parameters of GP in Table III according to values in Table V. We chose default values from the literature [13] for the population size (`Pop_size`), mutation rate (`Mut_rate`), crossover rate (`Cross_Rate`), and the max tree depth (`Max_Depth`) parameters. We set tournament selection (TRS) as selection criterion (`Sel_Crt`) since, when compared with other selection techniques, it leads to populations with higher fitness values [13]. We set the value of the tournament size (`T_Size`) according to the results of

TABLE IV
IDENTIFIER (ID), NAME, DESCRIPTION, NUMBER OF BLOCKS (#BK), INPUTS (#IN), OUTPUTS (#OUT), SIMULATION TIME (ST), NUMBER OF REQUIREMENTS (#REQS), AND THE NUMBER OF REQUIREMENTS WE USED TO ANSWER RESEARCH QUESTION RQ1, I.E., THEY PASS THE SANITY CHECK (#REQS WITHIN ROUND BRACKETS) OF EACH SIMULINK® MODEL OF THE COMPONENTS OF OUR STUDY SUBJECTS.

ID	Name	Description	#Bk	#In	#Out	ST(s)	#Reqs
TU	Tustin	A numeric model that computes integral over time.	57	5	10	10	5 (2)
EB	Effector Blender	A controller that computes the optimal effector configuration for a vehicle.	95	1	7	0	3 (0)
SW	Integrity Monitor	Monitors the airspeed and checks for hazardous situations.	164	7	5	10	2 (0)
FSM	Finite State Machine	Controls the autopilot mode in case of some environment hazard.	303	4	1	10	13 (2)
REG	Regulator	A typical PID controller.	308	12	5	10	10 (6)
NLG	Nonlinear Guidance	A guidance algorithm for an Unmanned Aerial Vehicles (UAV).	373	5	5	10	2 (0)
TX	Triplex	A redundancy management system.	481	5	4	10	4 (0)
TT	Two Tanks*	A controller regulating the incoming and outgoing flows of two tanks.	498	2	11	14	32 (8)
NN	Neural Network	A predictor neural network model with two hidden layers.	704	2	1	100	2 (0)
EU	Euler	Computes the rotation matrices for an inertial frame in a Euclidean space.	834	4	2	10	8 (0)
AP	Autopilot	A DeHavilland Beaver Airframe with Autopilot system.	1549	7	1	1000	11 (0)
AC	Attitude Control	Attitude control component of the ADCS of the ESAIL micro-satellite.	438	10	4	1	2 (1)

* does not support multiple control points.

TABLE V
VALUES FOR THE PARAMETERS OF TABLE III USED FOR RQ1.

	Parameter	Value	Parameter	Value
EP	SBA	DT/GP/RS	ST	see Table IV
	TS_Size	300	Stop_Crt	Timeout
	Timeout	1h	Nbr_Runs	100
GP	Max_Conj	3 or 4	Max_Disj	2
	Const_Min	−100	Const_Max	100
	Max_Depth	5	Init_Ratio	50%
	Pop_Size	500	Gen_Size	100
	Sel_Crt	*TRS*	T_Size	7
	Mut_Rate	0.1	Cross_Rate	0.9

* The values within the framed boxes 🟦, 🟩, and 🟧 are respectively from [13], [55], and [56]. The value within the framed box 🟨 is based on a preliminary analysis of the considered study subjects. The values within the framed boxes 🟥 are selected based on our domain knowledge on the considered study subjects.

an empirical study on ML parameter tuning [55]. We set the maximum number of generations (Gen_Size), the number of conjuntions (Max_Conj) and disjunctions (Max_Disj), and the initial ratio (Init_Ratio) based on the results of a preliminary analysis we conducted, over the considered study subjects, where we determined the average number of generations needed to reach a plateau. We assigned to Const_Min and Const_Max, respectively, the lowest and highest values the input signals can assume in our study subjects. We set the number of tests in the test suite (TS_Size) to 300, which was the value used to evaluate falsification-based testing tools in the ARCH-COMP 2019 and 2020 competitions [56], [57].

We configured RS by noticing that RS reuses part of the algorithm of GP. Therefore, for the parameters of RS, which are a subset of the parameters of GP, we assigned the same values considered for GP. Finally, we configured DT by considering the same values used by our earlier work, Gaaloul et al. [10], to evaluate EPIcuRus.

To answer RQ1, we performed the following experiment. We considered each of the 19 requirements under analysis and three different input profiles with respectively one (IP), two (IP′), and three (IP″) control points. We chose the number of control points of the input profiles based on the default input signals provided by the models of the CPS components. For the eight requirements of Two Tanks (TT), only the input profile IP was considered since this model only supports constant input signals.

Therefore, in total, we considered 32 requirement-profile combinations. For each combination, we ran EPIcuRus with GP, DT, and RS. We set a timeout of one hour, which is reasonable for this type of applications. As typically done in similar works (e.g., [24], [56]), we repeated each run 100 times to account for the randomness of the test case generation procedure. Therefore, in total, we executed 9600 runs[4]: 3200 runs ($32 \cdot 100$) for each of GP, DT, and RS. For each run, we recorded whether a sound assumption was returned. Furthermore, we computed the coverage value associated with the assumption. To compute *a coverage* value (COV_V), we need to compute the size of the valid input set for each assumption (see $|U|$ in Definition 1). We do so empirically. Specifically, we generated 100 different value assignments for *each* control point. These assignments are uniformly distributed within the value range of the control point. Then, we create the input set that we use to evaluate all the generated assumptions. When the assumption constrains more than one control point, the input set contains all the possible combinations of the value assignments of the control points. For example, to evaluate an assumption that constrains two control points, we create an input set with 10000 assignments ($100 \cdot 100$). Each input in the set is a combination of two assignments, each selected from the 100 assignments of each control point. We then compute the percentage of valid inputs (i.e., the inputs that satisfy the assumption) in the set. The higher this number, the larger the coverage provided by the assumption.

Results. The results of our comparison are reported in Figure 5. Each box plot reports the coverage value (COV_V) of the assumptions computed by GP, DT, and RS, and is labelled with the percentage of runs, across the 3200 runs, in which the technique was able to compute a sound assumption (the

[4]We executed our experiments on the HPC facilities of the University of Luxembourg [58].

value reported below the box plot). The average coverage values of the assumptions computed by GP, DT, and RS across their different runs are, respectively and approximately, 50%, 30% and 42%. Though there are variations across the different combinations of requirements and input profiles, GP can compute assumptions with a coverage value, that is, on average, 20% and 8% higher than that of the assumptions computed by DT and RS, respectively.

Across all the runs, GP was able, on average, to compute a sound assumption in 47.9% of the cases (1533 out of 3200), while DT, and RS were able to compute a sound assumption in respectively 46.7% (1495 out of 3200) and 48.9% (1565 out of 3200) of the cases. Therefore, GP is, on average, slightly more effective (1.2%) than DT, and slightly less effective than RS (1%) in computing sound assumptions. For each requirement-profile combination, Table VI reports the number of runs, among the 100 runs executed for the requirement-profile combination, in which GP, DT, and RS were able to compute a sound assumption. When GP was less effective than DT and RS, in many runs, across all the different iterations, GP was able to learn assumptions with large coverage that were close to actual sound assumptions, but theoretically unsound. Intuitively, in these cases, maximizing the coverage of the assumption, by searching for assumptions with a higher fitness, leads GP away from the generation of assumptions that are sound. For example, for the requirement ϕ_1 of Two Tanks (`TT`), GP was returning, in one of its runs, the assumption $\texttt{t1h} \leq 0.5855 \vee \texttt{t1h} \geq 2.0065$ for one of the inputs of Two Tanks. This assumption is theoretically unsound. However, the assumption $\texttt{t1h} < 0.58 \vee \texttt{t1h} \geq 2$ is sound. For the same requirement, RS and DT were respectively returning, in one of their runs, the assumptions $\texttt{t1h} \geq 2.0259$ and $\texttt{t1h} < 0.5655 \vee \texttt{t1h} \geq 2.0265$ which are sound but have much lower coverage. Indeed, the assumption $\texttt{t1h} \leq 0.5855 \vee \texttt{t1h} \geq 2.0065$ has a larger coverage than $\texttt{t1h} \geq 2.0259$, since any input that satisfies $\texttt{t1h} \geq 2.0259$ also satisfies $\texttt{t1h} \geq 2.0065$. The assumption $\texttt{t1h} \leq 0.5855 \vee \texttt{t1h} \geq 2.0065$ also has a larger coverage than $\texttt{t1h} < 0.5655 \vee \texttt{t1h} \geq 2.0265$, since any input that satisfies $\texttt{t1h} < 0.5655$, also satisfies $\texttt{t1h} < 0.5855$, and any input that satisfies $\texttt{t1h} \geq 2.0265$ also satisfies $\texttt{t1h} \geq 2.0065$. Therefore, GP learned assumptions that are the closest to the one that has the largest coverage.

Considering each of the 32 combinations of requirements and input profiles separately, GP computed a sound assumption for 31 combinations in at least one of the 100 runs. DT and RS computed, respectively, a sound assumption for 26 and 22 of the 32 combinations in at least one of the 100 runs. Note that, computing a sound assumption once in 100 runs is still acceptable, since running our tool 100 times takes a few hours, when parallelization is used to execute different runs. So, in the worst case, engineeers need to run our tool for a few hours to obtain assumptions, which is acceptable for our usage scenario. In the one case that all the three techniques failed to generate a sound assumption, all the generated assumptions were inconclusive. In the cases where only DT or RS could not compute a sound assumption, the assumption learned by GP has a complex structure and, therefore, could not be computed

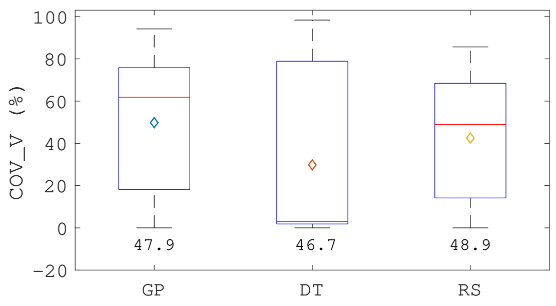


Fig. 5. Comparing GP, DT, and RS. The box plots show the coverage value of GP, DT, and RS (labels on the bottom of the figure). Diamonds depict the average. The value below the box plot is the percentage of runs, across all the runs, in which the technique was able to compute a sound assumption.

TABLE VI
NUMBER OF RUNS, AMONG THE 100 RUNS OF EACH REQUIREMENT-PROFILE COMBINATION, IN WHICH GP, DT AND RS WERE ABLE TO COMPUTE A SOUND ASSUMPTION

	IP			**IP′**			**IP″**		
Req.	**GP**	**DT**	**RS**	**GP**	**DT**	**RS**	**GP**	**DT**	**RS**
REG-ϕ_1	79	99	99	76	83	100	22	0	29
REG-ϕ_2	29	96	37	21	75	53	11	0	17
REG-ϕ_3	76	100	87	30	100	91	3	0	33
REG-ϕ_4	-	-	-	2	18	0	6	1	0
REG-ϕ_5	-	-	-	43	14	0	11	0	0
REG-ϕ_6	-	-	-	1	13	7	9	2	0
TU-ϕ_1	42	79	22	52	2	24	8	4	0
TU-ϕ_2	100	94	100	21	96	0	19	0	0
FSM-ϕ_1	-	-	-	-	-	-	100	86	100
FSM-ϕ_2	-	-	-	-	-	-	1	1	0
TT-ϕ_1	96	85	97						
TT-ϕ_2	93	62	89						
TT-ϕ_3	86	59	81						
TT-ϕ_4	93	55	93						
TT-ϕ_5	97	54	98						
TT-ϕ_6	92	55	89						
TT-ϕ_7	84	49	79						
TT-ϕ_8	88	94	85						
AC-ϕ_1	0	0	0						

* Symbol "-" marks entries related with input profiles for which the requirements are violated or satisfied.
For the eight requirements of Two Tanks (`TT`), only the input profile `IP` was considered since this model only supports constant input signals.
For `AC`, we considered the input profile `IP` suggested by our industrial partner.

by DT and was difficult to be generated by RS. To statistically compare the distributions of the coverage values generated by GP with those generated by DT and RS, we used the Wilcoxon rank sum test [59] with the level of significance (α) set to 0.05. In both cases, the test rejected the null hypothesis (p-values $<$ 0.05). Hence, assumptions learned by GP have a significantly larger coverage than those learned by RS and DT.

The box plots in Figure 6 depict the behavior of GP, DT, and RS across the input profiles `IP`, `IP′` and, `IP″`. Each box plot reports the coverage value of one tool for a given input profile and is labelled with the percentage of cases, across the runs associated with that input profile, in which the tool was able to compute a sound assumption (value reported below the box plot). The results confirm that GP generates assumptions with a larger coverage than DT and RS, however, with a negligible loss of capacity to compute sound assumptions. For GP and DT, the test returned p-values lower than 0.05 for `IP` and

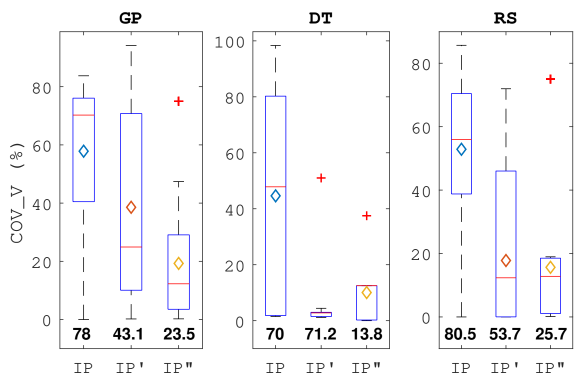


Fig. 6. Comparing GP, DT, and RS. The box plots show the coverage value of GP, DT, and RS for the input profiles IP, IP$'$ and, IP$''$ (labels on the bottom of the figure). Diamonds depict the average. The value below the box plot is the percentage of runs, across all the runs of each input profile, in which the technique was able to compute a sound assumption.

IP$'$. For IP$''$, the p-value is greater than 0.05 (i.e., 0.1) since the sample size is too small to reject the null hypothesis. For GP and RS, the test returned p-values lower than 0.05 for all the input profiles. The results show that, as expected, the more complex the input profile, the more difficult the computation of a sound assumption. Therefore, to handle more complex input profiles, developers should tune the values of the parameters in Table V, e.g., by increasing the timeout, the number of test cases, and the population size.

The answer to **RQ1** is that, on the considered study subjects, in contrast to DT and RS, GP can learn a sound assumption for 31 combinations out of 32 combinations of requirements and input profiles. The assumptions computed by GP have also a significantly larger coverage (20% and 8%) than those learned by DT and RS.

B. RQ2-1 — Usefulness of the assumptions with large coverage

To check whether GP can learn assumptions with large coverage similar to the one that would be manually defined by engineers, we empirically evaluated GP by considering the AC component of the ESAIL microsatellite. We analyzed the assumptions computed by GP in collaboration with the industrial CPS engineers that developed ESAIL. In this question, since we do not limit our analysis to sound assumptions, EPIcuRus was configured to return a sound assumption, if found, or the assumption computed in the last iteration of EPIcuRus, if none of the assumptions generated across the different iterations could be proven to be sound.

Methodology and Experimental Setup. To learn assumptions on the 27 input signals of the ten inputs of ESAIL (see Table I), we considered the parameter settings in Table VII. Cells marked with a gray background color denote parameter values that differ from the one considered for RQ1. We increased the values assigned to the test suite size (TS_Size) and the timeout (Timeout), since AC is significantly more complex than the other models. Recall from RQ1 that the parameter values in Table V did not lead to any sound assumption. The number of conjunctions (Max_Conj) and disjunctions (Max_Disj) are respectively set to 1 and 0 since, according to our industrial partner, the assumptions can be represented as a conjunction of two inequalities among complex arithmetic expressions (see Section II). In practice, engineers do not know a priori the assumption with the largest coverage of the system. However, they can select parameter values based on their domain knowledge combined with experiments. The values assigned to Const_Min and Const_Max are, respectively, the lowest and the highest values the input signals of AC can assume. We considered the input profile IP with a single control point since, given the time domain (see Section II) and according to the engineers of ESAIL, this input profile is sufficiently complex to represent changes in the inputs of AC over the considered time domain. We assumed the ranges $[-0.01, 0.01]$, $[-0.01, 0.01]$, $[0, 0.005]$, $[0, 0.005]$, and $[0, 0.005]$ as input domain for each of the input signals of the inputs $\mathtt{q}_t$, $\mathtt{q}_e$, ω_t, ω_e, and Rwh, respectively. We set the other inputs to constant values provided by our industrial partner.

We considered 100 runs of EPIcuRus and saved the assumptions computed for requirement ϕ_1 in each of these runs. Then, to assess whether EPIcuRus can learn assumptions similar to the ones that would be manually defined by engineers, we proceeded as follows. We elicited the assumption of AC for ϕ_1 in collaboration with the ESAIL engineers as described in Section II. This was done by consulting the Simulink® model design and the design documents of the satellite. We then compared the assumptions returned by EPIcuRus and the one we elicited in collaboration with the ESAIL engineers, i.e., the assumption $\mathtt{A}_1$ for ϕ_1. While eliciting this assumption, we found discrepancies between the model design and the design documents of the satellite. The problems were in the design documents of the satellite and were fixed by ESAIL engineers. We corrected our assumption accordingly to match the actual ESAIL design model. To assess the extent to which GP can learn sound assumptions similar to the ones that would be manually defined by engineers, we analyzed how many runs of EPIcuRus were able to learn each of the terms of the predicates $\mathtt{P}_1(t)$ and $\mathtt{P}_2(t)$. Note that, while the values of Const_Min and Const_Max are set to -0.1 and 0.1, coefficients of the terms with higher values (i.e., $+783.3$ in $\mathtt{A}_1$) can be generated by selecting small values for the constant terms, i.e., $+1$ and -1 in the assumption $\mathtt{A}_1$. For example, our algorithm can generate the assumption $0.783 \cdot \omega_e\_\mathtt{x}(t) < 0.001$ which is equivalent to $+783.3 \cdot \omega_e\_\mathtt{x}(t) < 1$. We relied on classical arithmetic properties to scale up and down the values of the coefficients. Furthermore, given the domain of the inputs, the minimum and the maximum value for each term in exp is reported in Table VIII. Given the ranges in the table, the term that can assume the highest value in exp is T1, followed by T2. For example, for term T2 the minimum value is -1.66 (i.e., $-332.6 \cdot 0.005$) and the maximum value is 0 (i.e., $-332.6 \cdot 0$).

Results. We obtained 100 assumptions from 100 runs of EPIcuRus. These assumptions were inconclusive due to the complexity of their mathematical expressions. We analyzed and compared the syntax and the semantics of these assumptions with respect to the actual assumption of AC ($\mathtt{A}_1$).

TABLE VII
VALUES FOR THE PARAMETERS OF TABLE III USED TO ASSESS WHETHER EPICURUS CAN LEARN ASSUMPTIONS SIMILAR TO THE ONE THAT WOULD BE MANUALLY DEFINED BY ENGINEERS.

	Parameter	**Value**	**Parameter**	**Value**
EP	`SBA`	GP	`ST`	$[0,1]$s
	`TS_Size`	3000	`Stop_Crt`	`Timeout`
	`Timeout`	5h	`Nbr_Runs`	100
GP	`Max_Conj`	1	`Max_Disj`	0
	`Const_Min`	-0.1	`Const_Max`	0.1
	`Max_Depth`	5	`Init_Ratio`	50%
	`Pop_Size`	500	`Gen_Size`	100
	`Sel_Crt`	*TRS*	`T_Size`	7
	`Mut_Rate`	0.1	`Cross_Rate`	0.9

TABLE VIII
MINIMUM AND MAXIMUM VALUE OF EACH TERM OF exp.

Term	**Min**	**Max**	**Term**	**Min**	**Max**
`T1`	0	3.91	`T2`	-1.66	0
`T3`	0	0.175	`T4`	-0.0012	0
`T5`	-0.1187	0	`T6`	0	0.0897
`T7`	0	0.025	`T8`	-0.025	0
`T9`	-0.0013	0	`T10`	0	0.0013

The results are shown in Table IX. Specifically, Table IX shows which of the terms `T1`, `T2`, ..., `T10` of $\texttt{P}_1$ and $\texttt{P}_2$, respectively, appear in the 100 assumptions computed by EPIcuRus. Each row of the table reports four metrics computed for one of the ten terms of both $\texttt{P}_1$ and $\texttt{P}_2$. For each term $\texttt{T}_i$ of $\texttt{P}_1$ or $\texttt{P}_2$, the tables' columns show the following:

- `N`-labelled columns. The number of runs in which the assumptions generated by EPIcuRus contain the term $\texttt{T}_i$ but not necessarily with the same coefficient as that appearing in the expression exp.
- `S`-labelled columns. The percentage of the runs in which the sign of $\texttt{T}_i$ is the same as its sign in the expression exp over all the runs where the generated assumption by EPIcuRus contained $\texttt{T}_i$.
- `D`-labelled columns. The average of the differences between the values of the coefficients of $\texttt{T}_i$ in exp and in the assumptions returned by EPIcuRus.
- `MaxD`-labelled columns. The maximum of the differences between the values of the coefficients of $\texttt{T}_i$ in exp and in the assumptions returned by EPIcuRus.

The column labeled by `C` in Table IX shows the values of the coefficients of the terms $\texttt{T}_i$ in exp.

The results in Table IX show that the terms `T1` and `T2` of $\texttt{P}_1$ and $\texttt{P}_2$, that can yield the highest values among other terms (see Table VIII), were contained in the assumptions returned by EPIcuRus in 59 and 69, and 88 and 74, out of the 100 runs, respectively. Note that GP learns assumptions with an arbitrary structure as it does not know the structure of the assumption a priori. For this reason, the terms `T1` and `T2` of $\texttt{P}_1$ are contained in a different number of assumptions than the terms `T1` and `T2` of $\texttt{P}_2$, respectively. The other terms of $\texttt{A}_1$, that yield negligible values compared with `T1` and `T2`, cannot be effectively learned by EPIcuRus, i.e., they are contained

TABLE IX
THE VALUE `C` OF THE COEFFICIENT OF THE TERM $\texttt{T}_i$ IN exp, THE NUMBER `N` OF RUNS IN WHICH EPICURUS WAS ABLE TO LEARN $\texttt{T}_i$ IN $\texttt{P}_1$ AND $\texttt{P}_2$. THE PERCENTAGE `S` OF RUNS IN WHICH THE SIGN OF $\texttt{T}_i$ WAS CORRECT. THE AVERAGE `D` AND THE MAXIMUM `MAXD` OF THE DIFFERENCE BETWEEN THE COEFFICIENT OF $\texttt{T}_i$ OF $\texttt{A}_1$ AND THE ONE RETURNED BY EPICURUS.

		$\texttt{P}_1$				**$\texttt{P}_2$**			
$\texttt{T}_i$	**C**	**N**	**S**	**D**	**MaxD**	**N**	**S**	**D**	**MaxD**
`T1`	783	59	71	679	4328	88	94	256	1550
`T2`	-332	69	91	247	1373	74	68	409	1331
`T3`	3	3	33	21	35	9	11	41	203
`T4`	-50	5	40	17682	87238	17	58	30778	242066
`T5`	-4751	1	0	4751	4751	3	33	4949	5345
`T6`	3588	1	0	3588	3588	4	25	3520	3998
`T7`	1000	0	-	-	0	0	-	-	0
`T8`	-1000	0	-	-	0	0	-	-	0
`T9`	-54	5	60	17536	87133	6	50	1585	4732
`T10`	54	0	-	-	0	1	0	54	54

in the assumptions returned by EPIcuRus in only a limited number of runs (< 17 each). This is an inherent property of the search as it cannot learn terms that have low values. Note that, in most of the runs (78 out of 100), all the terms learned by EPIcuRus are either part of $\texttt{P}_1$ or part of $\texttt{P}_2$. Only in 28 runs the assumptions produced by EPIcuRus contained a spurious term. Furthermore, in these cases, given the input domains, the values the spurious terms could yield are irrelevant compared to the other terms. To conclude, since the terms `T1` and `T2` of $\texttt{P}_1$ and $\texttt{P}_2$ were contained in the assumptions returned by EPIcuRus in 59 and 69, and 88 and 74, out of the 100 runs, engineers are very likely to learn an assumption that contains the terms `T1` and `T2` by executing EPIcuRus a few times, which would take less than a day. For example, for the term `T1` of $\texttt{P}_1$ the probability of finding it during the first, second, or third run is 0.9311 (i.e., $0.59 + (1 - 0.59) \cdot 0.59 + (1 - 0.59)^2 \cdot 0.59$). This shows that the term `T1` of $\texttt{P}_1$ is likely to be computed within three runs.

When the terms `T1` and `T2` were contained in the assumptions computed by EPIcuRus, the sign was correct in 71% and 91%, and 94% and 68% of the cases, for $\texttt{P}_1$ and $\texttt{P}_2$, respectively. The average of the differences between the coefficients of the terms `T1` and `T2` of $\texttt{P}_1$ and $\texttt{P}_2$ in $\texttt{A}_1$ and in the assumptions returned by EPIcuRus are 679 and 247, and 256 and 409, respectively. The maximum of the differences between the coefficients of the terms `T1`, and `T2` of $\texttt{P}_1$ and $\texttt{P}_2$ in $\texttt{A}_1$ and in the assumptions returned by EPIcuRus are 4328 and 1373, and 1550 and 1331, respectively. Note that the coefficients of the terms `T1`, and `T2` of $\texttt{P}_1$ and $\texttt{P}_2$ in $\texttt{A}_1$ are 783 and -332, respectively. As we can see, the coefficient values computed by EPIcuRus are not too different from the actual coefficient values for `T1`, and `T2`, even though EPIcuRus could technically select any arbitrary number as a coefficient for these terms. However, provided with such a large search space, EPIcuRus has been able to select coefficients for these terms that are in the same order of magnitude (i.e., number's nearest power of ten) as their actual coefficients. Such accuracy for coefficient estimates is acceptable when the coverage of the assumptions is prioritized and what matters is the identification by engineers of the assumptions' terms that yield the highest values among other terms in the actual assumption. A higher accuracy would not have significant practical benefits since the model checker would anyway not be able to confirm the soundness of the assumption.

To illustrate how useful EPIcuRus can be in practice, let us take three of the 100 runs for which EPIcuRus returned the following representative assumptions:

$$\begin{aligned}
\mathtt{A}_{r1}(t) = \quad & \underbrace{+1003.4 \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P1-T1}} \underbrace{-515.8 \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P1-T2}} +1 \geq 0 \wedge \\
& \underbrace{958.0 \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} \underbrace{-452.8 \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T2}} -1 \leq 0 \\
\mathtt{A}_{r2}(t) = \quad & \underbrace{+757.2 \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} \underbrace{-413.1 \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T2}} \\
& \underbrace{-4787.4 \cdot \omega_e\_\mathtt{x}(t) \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T4}} \\
& \underbrace{-4787.4 \cdot \omega_e\_\mathtt{y}(t)^2}_{\mathtt{P2-T9}} -1 \leq 0 \\
\mathtt{A}_{r3}(t) = \quad & \underbrace{+757.6 \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P1-T1}} \underbrace{-448.4 \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P1-T2}} \\
& + 448.4 \cdot \omega_e\_\mathtt{x}(t)^2 + 1 \geq 0 \wedge \\
& \underbrace{856.8 \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} \underbrace{-419.8 \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T2}} \\
& - 8.6 \cdot \mathtt{Rwh\_z}(t) - 1 \leq 0
\end{aligned}$$

where `P1-T1`, `P1-T2`, ... and `P2-T1`, `P2-T2`, ... label the terms `T1`, `T2`, ... of $\mathtt{P}_1(t)$ and $\mathtt{P}_2(t)$, respectively. EPIcuRus returns both assumptions that contain the terms `T1` and `T2` of both predicates (e.g., $\mathtt{A}_{r1}(t)$), and assumptions that contain the terms `T1` and `T2` of only one of the predicates (e.g., $\mathtt{A}_{r2}(t)$). Some assumptions contain only terms that are part of $\mathtt{A}_1$ (e.g., $\mathtt{A}_{r1}(t)$, $\mathtt{A}_{r2}(t)$), others contain additional spurious terms (e.g., $\mathtt{A}_{r3}(t)$), i.e., terms that are not part of $\mathtt{A}_1$ (e.g., $8.6 \cdot \mathtt{Rwh\_z}(t)$). Finally, note that, when EPIcuRus learns a term present in both $\mathtt{P}_1(t)$ and $\mathtt{P}_2(t)$ (e.g., `T1`), the coefficients of `P1-T1` and `P1-T2` are not necessarily equal.

Discussion. When EPIcuRus is configured to learn assumptions with large coverage that are not necessarily proven to be-sound, our results show that the resulting assumptions include the terms that yield the highest values among other terms in the actual assumption. Even though, in the generated assumptions, not all the terms are present, and the values of their coefficients are only estimates, ESAIL engineers confirmed that these assumptions are still useful and beneficial for designing CPS components, because they can help engineers identify flaws in their components.

Specifically, engineers generally know which input signals yield the highest values and expect to see those input signal variables in the assumptions generated by EPIcuRus. The absence of those variables in the generated assumptions may indicate flaws. For example, by analyzing $\mathtt{A}_{r1}(t)$, engineers understand that estimated speeds of the satellite across `x` and `y` axes have a significant impact on the satisfaction of ϕ_1, as expected. Furthermore, the assumption also provides high level and approximate information regarding the values of $\omega_e\_\mathtt{x}(t)$ and $\omega_e\_\mathtt{y}(t)$ that satisfy the requirement.

Engineers can execute several runs of EPIcuRus and obtain a report containing the information shown in Table IX. By consulting the data reported in the table, they can understand which terms are present in most of the assumptions computed by EPIcuRus and yield the highest values. Note that, the term of the assumption yielding the highest value is likely to be the one that has the highest impact on the property satisfaction. Running 100 runs of EPIcuRus requires approximately 20 days. However, the results can be obtained within approximately one day by running 20 instances of EPIcuRus in parallel. This is a reasonable solution for computing assumptions of critical CPS components.

The assumptions returned by EPIcuRus could not be learned by our previous version of EPIcuRus that relies on DT to learn assumptions. Finally, EPIcuRus is the only existing tool that is able to synthesize assumptions for CPS Simulink® components. Therefore, there is no alternative that engineers could consider to address their need.

The answer to **RQ2-1** is that, among the terms of the assumptions identified by the LuxSpace engineers, EPIcuRus configured with GP was able to learn the terms that yield the highest values among other terms in the actual assumption. These assumptions could not be learned with any other tool.

C. *RQ2-2 — Usefulness of the Sound Assumptions*

To check whether GP can learn sound assumptions similar to the one that would be manually defined by engineers, we configured EPIcuRus to increase the chances of computing simpler assumptions whose soundness can be verified by QVtrace.

Methodology and Experimental Setup. To check whether learning simpler assumptions allows QVtrace to prove that they are sound, we configured EPIcuRus using the values of the parameters in Table X. Compared with the values in Table VII, we decreased the timeout from five hours to one hour, the number of generations (Gen_Size) from 100 to 10, and set the values assigned to `Const_Min` and `Const_Max` to -0.001 and 0.001, respectively. We considered 100 runs of EPIcuRus and saved the assumptions computed for the requirement ϕ_1 in each of these runs. Then, we computed the percentage of the runs in which EPIcuRus was able to compute a sound assumption.

Results. Across the 100 runs, EPIcuRus was able to compute sound assumptions in 16 runs. On average, generating one sound assumption for `AC` took about 6 hours. All the 16 sound assumptions generated by EPIcuRus have lower coverage (simpler) than $\mathtt{A}_1$, the actual assumption of `AC`. We identified three distinct patterns to categorize the 16 generated assumptions. Below, we list the patterns and, for each one, we show which terms from the original assumption $\mathtt{A}_1$ appear in the pattern.

$$\begin{aligned}
\mathtt{A}_{s1}(t) = \quad & \underbrace{\mathtt{a} \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P1-T2}} -\mathtt{c} \leq 0 \wedge \\
& \underbrace{\mathtt{b} \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} -\mathtt{d} \leq 0 \\
\mathtt{A}_{s2}(t) = \quad & \underbrace{\mathtt{a} \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} \underbrace{+\mathtt{b} \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T2}} -\mathtt{c} \leq 0 \\
\mathtt{A}_{s3}(t) = \quad & \underbrace{\mathtt{a} \cdot \omega_e\_\mathtt{x}(t)}_{\mathtt{P2-T1}} \underbrace{+\mathtt{b} \cdot \omega_e\_\mathtt{y}(t)}_{\mathtt{P2-T2}} \underbrace{+\mathtt{c} \cdot \omega_e\_\mathtt{z}(t)}_{\mathtt{P2-T3}} \leq 0
\end{aligned}$$

TABLE X
VALUES FOR THE PARAMETERS OF TABLE III USED FOR RQ2-2.

	Parameter	Value	Parameter	Value
EP	SBA	GP	ST	$[0, 1]$s
	TS_Size	3000	Stop_Crt	Timeout
	Timeout	1h	Nbr_Runs	100
GP	Max_Conj	1	Max_Disj	0
	Const_Min	-0.001	Const_Max	0.001
	Max_Depth	5	Init_Ratio	50%
	Pop_Size	500	Gen_Size	10
	Sel_Crt	*TRS*	T_Size	7
	Mut_Rate	0.1	Cross_Rate	0.9

The above assumptions, although simpler than the original assumption $\mathtt{A}_1$, have sufficiently large coverage as confirmed by ESAIL engineers. For example, the $\mathtt{A}_{s1}(t)$ pattern indicates that, when the values of the estimated speed of the satellite along its x and y axes are low, requirement ϕ_1 is satisfied regardless of the values assigned to the other inputs. Similarly, the $\mathtt{A}_{s2}(t)$ and $\mathtt{A}_{s3}(t)$ patterns indicate that, when the sum of the speeds of the satellite along its x and y axes is low, requirement ϕ_1 is satisfied. In other words, despite being an oversimplification of reality, learned assumptions provide correct insights into the conditions leading to the satisfaction of requirements.

To verify that the satisfaction of the assumption generated by EPIcuRus entails the satisfaction of the baseline assumption, we compared the assumptions returned by EPIcuRus with the baseline assumption $\mathtt{A}_1$ of AC (see Section VI) using QVtrace. We loaded a Simulink® model representing the baseline assumption in QVtrace. The inputs of the model are the input signals of the assumptions. The output is a Boolean value: *true* if the assumption is satisfied, *false* otherwise. Then, we iteratively loaded each of the assumptions generated by EPIcuRus in QVtrace. For all the assumptions generated by EPIcuRus, QVtrace confirmed that the output of the Simulink® model is *true* for all the inputs that satisfy the assumption. This shows that the satisfaction of the assumption generated by EPIcuRus entails the satisfaction of the baseline assumption. Therefore, any input that satisfies the assumption generated by EPIcuRus also satisfies the actual (baseline) assumption of the AC component.

The answer to **RQ2-2** is that, when EPIcuRus was configured with parameters that lead to the generation of sound assumptions, EPIcuRus was able to generate sound assumptions in 16 runs out of 100. Therefore, through multiple runs, EPIcuRus can generate a sound assumption within approximately six hours. Though simpler than actual assumptions, these learned assumptions appear to provide correct and useful insights.

D. Discussion

In the following, we discuss (i) the impact of our results on the documentation practices of LuxSpace, and (ii) the impact of our choices regarding the design and configuration settings of EPIcuRus on our results.

Documentation Practices. LuxSpace engineers detail the behavior of ESAIL in a design document. This document is divided into several sections, one for every component of ESAIL. For every component, among various information, the specification document contains (i) the description of its inputs and outputs; (ii) the mathematical formulae that define the behavior of the component; (iii) the natural language explanation of these formulae; and (iv) a discussion of the design decisions made by the engineers to define the behavior of the component. The above information was also present in the specification documents of the benchmark models [54] of Lockheed Martin [18] we considered in Section II. Neither LuxSpace nor Lockheed Martin engineers included the assumptions of the software components in the specification documents. The ESAIL Simulink® model contains a set of assertion blocks [60] encoding simple component assumptions that check whether the values assumed by some of the signals are included within valid ranges. For the Lockheed Martin models, the input type (e.g., Boolean or Real) was described in the specification document, but the valid input ranges were not specified. None of the models included complex assumptions containing arithmetic expressions defined over multiple variables, such as assumption $\mathtt{A}_1$ presented in Section II. This confirms that manually identifying assumptions is difficult, especially when the component has many inputs and its behavior is defined by complex and non-linear equations. As confirmed by the results reported in our evaluation, EPIcuRus helps engineers in addressing this problem by automatically identifying complex and sound component assumptions.

Fitness Measure. Our fitness measure guides the search toward sound assumptions with large coverage. Coverage, as the core of the fitness function, might lead to assumptions that, accidentally and unnecessarily, correlate signals generated by different components. However, in our evaluation, this was not the case when soundness was a priority (see RQ2-2). When soundness was not a priority (see RQ2-1), the assumptions produced by EPIcuRus contained a spurious term in only 28 runs (over 100). Furthermore, in these cases, given the input domains, the values the spurious terms could yield are negligible compared to the other terms. Therefore, based on our results, the number of assumptions that, accidentally and unnecessarily, correlate signals generated by different components is limited and only concern the case in which soundness was not a priority.

Metric to Compute the Cardinalities of the Valid Input Sets. To compute the size of the valid input sets we decided not to prioritize any of the input dimensions (e.g., speed, distance, angle). This was done both in our search-based algorithm (see Section V) and in our evaluation (see Section VI). This is because we wanted to maximize coverage across all input dimensions equally, regardless of their types. The effectiveness of our metrics is confirmed by the results obtained for RQ2-1 and RQ2-2. If there is a need to focus coverage on certain input dimensions, engineers can modify these metrics, e.g., by giving a higher weight to some of the input types.

Control Points-Based to Signal-Based Assumptions. To apply model checking on the generated assumptions, we translated control point-based assumptions to signal-based ones.

Our translation is provided for a case where assumptions on control points are forced to hold continuously across the time interval between the control point and its preceding control point since it generates simpler assumptions that (a) are easier to be understood by engineers and (b) are more likely to lead to a conclusive verdict when analyzed by the model checker. Our evaluation showed that, our strategy successfully learnt sound assumptions with high coverage for all of our study subjects. However, more complex translations that encode correlations of signals in which the expression needs a time shift on the time series can be used.

Configuration Settings. Our fitness function guides the search towards sound assumptions with high coverage. Therefore, our fitness function learns constraints on the terms of the assumption $\mathtt{A}_1$ that can assume the highest values. As mentioned in Section VI-B, when soundness is not a priority, in 28 runs (over 100), the assumptions produced by EPIcuRus contained a spurious term. The configuration settings of EPIcuRus (see Table VII) influence the number of spurious terms learned by our tool. The higher the timeout (`Timeout`), the more likely is EPIcuRus to produce spurious terms since it performs more iterations, and therefore can learn assumptions that have a larger coverage (which may contain unnecessary correlation between input signals). The higher `Max_Dept`, the higher is the chance to have spurious terms in the assumption, since the tool can learn larger assumptions.

E. Threats To Validity

The set of models we selected for the evaluation and their features influence the generalizability of our results. Related to this thread, we note that: First, the public domain benchmark of Simulink® models we used in our study have been previously used in the literature on testing of CPS models [38], [54]; second, the models in the benchmark represent realistic and representative models of CPS components from different domains; third, our industry satellite model represents a realistic and representative CPS model for which we could develop assumptions manually by collaborating with the engineers who had developed this model; fourth, our results can be further generalized by additional experiments with diverse types of CPS components and by assessing EPIcuRus over those components.

VII. Related Work

This section compares EPIcuRus with the following threads of research: (i) verification, testing and monitoring CPS, (ii) compositional and assume-guarantee reasoning for CPS, (iii) learning assumptions for software components, and (iv) learning the values for unspecified parameters.

Verification, Testing and Monitoring of CPS. Approaches to verifying, testing, and monitoring CPS were proposed in the literature (e.g., [24], [38], [39], [61]–[70]). However, these approaches usually assume that the assumptions on the inputs of the CPS component under analysis are already specified. Those approaches verify, test, and monitor the behavior of the CPS component for the input signals that satisfy those assumptions. Our work is complementary to those and, in contrast, it automatically identifies (implicit) assumptions on test inputs. Considering those assumptions is an important prerequisite to ensure that testing and verification results are not overly pessimistic or spurious [8], [9], [71]–[74].

Compositional reasoning. Assume-guarantee and design by contract approaches were proposed in the literature to support hardware and software verification (e.g., [3]–[6], [75]–[80]). Assume-guarantee contracts represent the assumptions a CPS component makes about its environment, and the properties it satisfies when these assumptions hold, i.e., its guarantees. Some recent work discusses how to apply assume-guarantee to signal-based modeling formalisms, such as Simulink® and analog circuits (e.g., [81]–[83]). However, these frameworks assume that assumptions and guarantees are manually defined by the designers of the CPS components. Our work is complementary as the assumptions learned by EPIcuRus can be used within these existing frameworks. Finally, our work also differs from assume-guarantee testing, where the assumptions defined during software design are used to test the individual components of the system [74].

Learning Assumptions. The problem of automatically inferring assumptions of software components, a.k.a supervisory control problem, was widely studied in the literature (e.g., [1], [7]–[9], [71], [84]–[90]). However, the solutions proposed in the literature are solely focused on components specified in finite-state machines and are not applicable to signal-based formalisms (e.g., Simulink® models), that are widely used to specify CPS components (see Section I).

Kampmann et al. [91] proposed an approach to automatically determine under which circumstances a particular program behavior, such as a failure, takes place. However, this approach uses a decision tree learner to observe and learn which input features are associated with the particular program behavior under analysis. As such, for our usage scenario, this approach is going to inherit the same limitations of our earlier version of EPIcuRus.

Dynamic invariants generators (e.g., [92], [93]) infer conditions that hold at certain points of a program. They generate a set of candidate invariants and return the best candidates that hold over the observed program executions. In contrast, the goal of this work is to generate environment assumptions for signal-based modeling formalisms, such as Simulink® models. Environment assumptions can be considered as a specific type of invariant, but introduce specific challenges, such as the ones considered in this work.

Property inference aims at automatically detecting properties that hold in a given system. It was also recently applied to feed-forward neural network [94]. While many approaches for property inference were proposed in the literature (e.g., [95]–[98]), they do not consider signal-based modeling formalisms (e.g., Simulink®) which are the targets of this work.

Template-based specification mining are used to synthesize assumptions following with a certain structure [99], [100]. However, solutions from the literature (e.g., [99]–[101]) use LTL-GR(1) to express assumptions. These formalisms are substantially different and less expressive than the one considered in this work, and can not express the signal-based assumptions generated by EPIcuRus.

In this work, we combined model checking and model testing to learn assumptions. This idea was supported by a recent study [54] that analyzed the complementarity between model testing and model checking for fault detection purposes.

Finally, Sato et al. [102] recently considered the problem of finding an input signal under which the system's behavior satisfies a given requirement. This is a sub-problem of the assumption generation problem, where assumptions identify conditions on the input signals (and therefore sets of input signals) that ensure the satisfaction of a given requirement.

This paper significantly extends our previous version of EPIcuRus [10]. Our extension enables learning assumptions containing conditions defined over multiple signals related by both arithmetic and relational operators. Assumptions containing conditions defined over multiple signals related by both arithmetic and relational operators are common for industrial CPS components. This is confirmed by our industrial case study from the satellite domain. Differently than our previous work, we used genetic programming to synthesize complex assumptions of CPS components. Finally, we performed an extensive and thorough evaluation of the assumptions computed by the extended version of EPIcuRus using an industrial case study from the satellite domain. The assumptions computed by EPIcuRus were evaluated in collaboration with the engineers that developed the satellite.

Learning Parameters. The problem of learning (requirement) parameters from simulations was extensively studied in the literature [103]–[105]. Our work is significantly different from those, since it aims to learn assumptions on the input signals.

EPIcuRus extends counterexample-guided inductive synthesis [106] by exhaustively verifying the learned assumptions using an SMT-based model checker. Furthermore, differently from counterexample-guided inductive synthesis, EPIcuRus (i) targets signal-based formalisms, that are widely used in the CPS industry, (ii) extracts assumptions from test data, and (iii) uses test cases to efficiently produce a large amount of data to be fed in our machine learning algorithm to derive sound assumptions with large coverage.

VIII. Conclusion

This paper proposes a technique to learn complex assumptions for CPS systems and components. Our technique uses genetic programming (GP) to learn assumptions containing conditions defined over multiple signals related by both arithmetic and relational operators. Environment assumptions are required to ensure that the CPS under analysis meets its requirements and to avoid spurious failures during verification. We evaluated our approach using 12 models of CPS components with 94 requirements provided by Lockheed Martin [18] and the model of the attitude control component of a satellite with four requirements provided by LuxSpace [19]. Our evaluation shows that our approach can learn many more sound environment assumptions compared to the alternative baseline techniques. Further, our approach is able to learn assumptions that have a significantly larger coverage than those generated by existing techniques. Finally, for our industrial CPS model, our approach is able to generate assumptions that are sufficiently close to the assumptions manually developed by engineers to be of practical value.

Our work can be extended in many different ways, a few of them are summarized in the following:

1) Our approach considers assumptions represented as a disjunction of one or more constraints defined as in Section III. Although our results show that, for our case studies, EPIcuRus can learn sound assumptions with high coverage expressed in this form, learning assumptions of different forms can further increase the applicability of EPIcuRus. This would require extending the EPIcuRus *test generation* and *assumption generation* procedures and selecting a *model checker* that supports new forms of assumptions;
2) As discussed in Section V, we did not use the interpolation function, specified within the input profile, to translate control points-based to signal-based assumptions. Analyzing and defining more complex translations is part of our future work;
3) Alternative techniques, such as program synthesis [107], [108], or Support Vector Machines [109] can be used for implementing the assumption generation step. These techniques can further improve the effectiveness of EPIcuRus;
4) EPIcuRus uses a fixed set of test cases to evaluate assumptions. Search-based software testing (SBST) is an alternative technique, which requires performing additional computations, that can be used for evaluating the generated assumptions. We plan to extend EPIcuRus to support the usage of SBST to evaluate the generated assumptions, and to assess whether using SBST is beneficial in practical applications given the additional computational cost;
5) The sanity check and soundness check steps use model-checking to determine if the given requirement is satisfied or violated by the model inputs satisfying the assumption. When the requirement is not satisfied, we plan to use the counterexample to guide the test generation step.

Finally, the assumptions generated by EPIcuRus have many other potential usage scenarios, such as compositional reasoning, or supporting the detection of design flaws. Assessing how these assumptions support such usage scenarios is part of our future work.

Acknowledgement

This project has received funding from the Luxembourg National Research Fund under the grant BRIDGES18/IS/12632261, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), and NSERC of Canada under the Discovery and CRC programs.

The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [58](HPC @ Uni.lu).

We thank the anonymous reviewers for the useful comments.

References

[1] D. Giannakopoulou, C. S. Pasareanu, and H. Barringer, "Assumption generation for software component verification," in *International Conference on Automated Software Engineering*. IEEE, 2002, pp. 3–12.

[2] A. Schaap, G. Marks, V. Pantelic, M. Lawford, G. Selim, A. Wassyng, and L. Patcas, "Documenting simulink designs of embedded systems," in *International Conference on Model Driven Engineering Languages and Systems (MODELS): Companion Proceedings*. ACM, 2018, p. 47–51.

[3] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-physical system design contracts," in *International Conference on Cyber-Physical Systems*. ACM, 2013, pp. 109–118.

[4] "Taming dr. frankenstein: Contract-based design for cyber-physical systems*," vol. 18, no. 3, 2012, pp. 217 – 238.

[5] T. A. Henzinger, S. Qadeer, and S. K. Rajamani, "You assume, we guarantee: Methodology and case studies," in *Computer Aided Verification*. Springer, 1998, pp. 440–451.

[6] L. de Alfaro and T. A. Henzinger, "Interface theories for component-based design," in *Embedded Software*, T. A. Henzinger and C. M. Kirsch, Eds. Springer, 2001, pp. 148–165.

[7] D. G. Cavezza, D. Alrajeh, and A. György, "Minimal assumptions refinement for realizable specifications," in *International Conference on Formal Methods in Software Engineering (FormaliSE)*. ACM, 2020.

[8] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh, "Assume-guarantee verification of source code with design-level assumptions," in *International Conference on Software Engineering*. IEEE, 2004, pp. 211–220.

[9] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu, "Learning assumptions for compositional verification," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2003, pp. 331–346.

[10] K. Gaaloul, C. Menghi, S. Nejati, L. C. Briand, and D. Wolfe, "Mining assumptions for software components using machine learning," in *Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020.

[11] (2020) fitctree. [Online]. Available: https://nl.mathworks.com/help/stats/fitctree.html

[12] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[13] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[14] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming*. Springer, 1998.

[15] (2020) Matlab GP toolbox. [Online]. Available: https://it.mathworks.com/matlabcentral/fileexchange/47197-genetic-programming-matlab-toolbox

[16] J. Madár, J. Abonyi, and F. Szeifert, "Genetic programming for the identification of nonlinear input- output models," *Industrial & engineering chemistry research*, vol. 44, no. 9, pp. 3178–3186, 2005.

[17] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[18] (2020) Lockheed Martin. [Online]. Available: https://www.lockheedmartin.com/en-us/index.html

[19] "Luxspace," 2020. [Online]. Available: https://luxspace.lu/

[20] "The European Space Agency (ESA)," 2020. [Online]. Available: https://www.esa.int/

[21] "exactEarth," 2020. [Online]. Available: https://www.exactearth.com/

[22] ESA, "Building and testing spacecraft," 2020. [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/Building_and_testing_spacecraft

[23] "Mathworks," https://mathworks.com, 02 2019.

[24] C. Menghi, S. Nejati, L. C. Briand, and Y. I. Parache, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *International Conference on Software Engineering (ICSE)*. ACM, 2020.

[25] (2020) Subsystems. [Online]. Available: https://it.mathworks.com/help/simulink/subsystems.html

[26] (2020) S-Function. [Online]. Available: https://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html

[27] (2020) MEX function. [Online]. Available: https://it.mathworks.com/help/matlab/call-mex-file-functions.html

[28] (2020) QVtrace. [Online]. Available: https://qracorp.com/qvtrace/

[29] (2019) QRA corp. [Online]. Available: https://qracorp.com/

[30] B. Wie, *Space Vehicle Dynamics and Control*, ser. AIAA education series. American Institute of Aeronautics and Astronautics, 1998.

[31] (2020) Virtual vector. [Online]. Available: https://it.mathworks.com/help/simulink/ug/signal-types.html

[32] P. Acquatella, "Fast slew maneuvers for the high-torque-wheels biros satellite," *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 61, no. 2, pp. 79–86, 2018.

[33] D. K. Chaturvedi, *Modeling and Simulation of Systems Using MATLAB and Simulink*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 2009.

[34] (2020) Simulink. [Online]. Available: https://nl.mathworks.com/products/simulink.html

[35] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, p. 255–299, Oct. 1990. [Online]. Available: https://doi.org/10.1007/BF01995674

[36] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[37] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[38] C. Menghi, S. Nejati, K. Gaaloul, and L. C. Briand, "Generating automated and online test oracles for simulink models with continuous and uncertain behaviors," in *Foundations of Software Engineering (ESEC/SIGSOFT FSE)*. ACM, 2019.

[39] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[40] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia, "A survey on adaptive random testing." IEEE Transactions on Software Engineering, 2019, pp. 1–1.

[41] A. Arcuri and L. C. Briand, "Adaptive random testing: an illusion of effectiveness?" in *International Symposium on Software Testing and Analysis, ISSTA*. ACM, 2011, pp. 265–275.

[42] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Advances in Computer Science - ASIAN*. Springer, 2004, pp. 320–329.

[43] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, "Requirements-driven test generation for autonomous vehicles with machine learning components," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 265–280, 2020.

[44] A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, and G. Sagardui, "Pareto efficient multi-objective black-box test case selection for simulation-based testing," *Information and Software Technology*, vol. 114, pp. 137–154, 2019.

[45] (2020) floor. [Online]. Available: https://www.mathworks.com/help/matlab/ref/floor.html

[46] M. A. Lones, "Sean luke: essentials of metaheuristics," *Genet. Program. Evolvable Mach.*, vol. 12, no. 3, pp. 333–334, 2011.

[47] R. Poli and W. B. Langdon, "Genetic programming with one-point crossover," in *Soft Computing in Engineering Design and Manufacturing*. Springer, 1998, pp. 180–189.

[48] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," in *Evolutionary Computation*, 1998, vol. 6, no. 3, pp. 231–252.

[49] A. Mavridou, H. Bourbouh, D. Giannakopoulou, T. Pressburger, M. Hejase, P. L. Garoche, and J. Schumann, "The Ten Lockheed Martin Cyber-Physical Challenges: Formalized, Analyzed, and Explained," in *International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 300–310.

[50] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA workbench*. Morgan Kaufmann, 2016.

[51] S. Silva and J. Almeida, "Gplab-a genetic programming toolbox for Matlab," in *Proceedings of the Nordic MATLAB conference*. Citeseer, 2003, pp. 273–278.

[52] D. P. Searson, "GPTIPS 2: an open-source software platform for symbolic data mining," in *Handbook of genetic programming applications*. Springer, 2015, pp. 551–573.

[53] (2020) Additional material. [Online]. Available: https://github.com/SNTSVV/EPIcuRus

[54] S. Nejati, K. Gaaloul, C. Menghi, L. C. Briand, S. Foster, and D. Wolfe, "Evaluating model testing and model checking for finding requirements violations in simulink models," in *Foundations of Software Engineering*, ser. ESEC/FSE. ACM, 2019.

[55] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[56] G. Ernst, P. Arcaini, A. Donze, G. Fainekos, L. Mathesen, G. Pedrielli, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "ARCH-COMP 2019 cate-

gory report: Falsification." in *ARCH@ CPSIoTWeek*, 2019, pp. 129–140.

[57] G. Ernst, P. Arcaini, I. Bennani, A. Donze, G. Fainekos, G. Frehse, L. Mathesen, C. Menghi, G. Pedrielli, M. Pouzet, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "ARCH-COMP 2020 category report: Falsification," in *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, ser. EPiC Series in Computing, vol. 74. EasyChair, 2020, pp. 140–152.

[58] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an Academic HPC Cluster: The UL Experience," in *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy: IEEE, July 2014, pp. 959–967.

[59] J. H. McDonald, *Handbook of biological statistics*, 2009, vol. 2.

[60] (2021) assertionblock. [Online]. Available: https://nl.mathworks.com/help/simulink/slref/assertion.html

[61] A. Mavridou, H. Bourbouh, P.-L. Garoche, D. Giannakopoulou, T. Pressburger, and J. Schumann, "Bridging the gap between requirements and simulink model analysis," in *Requirements Engineering: Foundation for Software Quality (REFSQ), Companion Proceedings*. Springer, 2020.

[62] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable Verification of Hybrid Systems," in *Computer Aided Verification (CAV)*. Springer, 2011.

[63] A. Tiwari, "Hybridsal relational abstracter," in *Computer Aided Verification (CAV)*. Springer, 2012.

[64] C. Boufaied, C. Menghi, D. Bianculli, L. Briand, and Y. Isasi-Parache, "Trace-checking signal-based temporal properties: A model-driven approach," in *International Conference on Automated Software Engineering (ASE 2020)*. IEEE, 2020.

[65] C. Menghi, E. Viganò, D. Bianculli, and L. C. Briand, "Trace-Checking CPS Properties: Bridging the Cyber-Physical Gap," in *International Conference on Software Engineering (ICSE)*. ACM, 2021.

[66] M. Borg, R. B. Abdessalem, S. Nejati, F.-X. Jegeden, and D. Shin, "Digital twins are not monozygotic–cross-replicating adas testing in two industry-grade automotive simulators," *arXiv preprint arXiv:2012.06822*, 2020.

[67] U. Markiegi, A. Arrieta, L. Etxeberria, and G. Sagardui, "Test case selection using structural coverage in software product lines for time-budget constrained scenarios," in *SIGAPP Symposium on Applied Computin (SAC)*. ACM, 2019, p. 2362–2371.

[68] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Search-based test case generation for cyber-physical systems," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2017, pp. 688–697.

[69] S. Y. Shin, K. Chaouch, S. Nejati, M. Sabetzadeh, L. C. Briand, and F. Zimmer, "Uncertainty-aware specification and analysis for hardware-in-the-loop testing of cyber-physical systems," *Journal of Systems and Software*, vol. 171, p. 110813, 2020.

[70] D. Giannakopoulou, F. Howar, M. Isberner, T. Lauderdale, Z. Rakamarić, and V. Raman, "Taming test inputs for separation assurance," in *International Conference on Automated Software Engineering (ASE)*. ACM/IEEE, 2014, p. 373–384.

[71] H. Barringer and D. Giannakopoulou, "Proof rules for automated compositional verification through learning," in *In Proc. SAVCBS Workshop*, 2003, pp. 14–21.

[72] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[73] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 11:1–11:61, 2012.

[74] D. Giannakopoulou, C. S. Pasareanu, and C. Blundell, "Assume-guarantee testing for software components," *IET Software*, vol. 2, no. 6, pp. 547–562, 2008.

[75] A. Arrieta, J. A. Agirre, and G. Sagardui, "A tool for the automatic generation of test cases and oracles for simulation models based on functional requirements," in *Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 1–5.

[76] C. Menghi, P. Spoletini, M. Chechik, and C. Ghezzi, "A verification-driven framework for iterative design of controllers," *Formal Aspects of Computing*, vol. 31, no. 5, pp. 459–502, 2019.

[77] C. Menghi, P. Spoletini, M. Checkik, and G. Carlo, "Supporting verification-driven incremental distributed design of components," in *Fundamental Approaches to Software Engineering (FASE)*. Springer, 2018.

[78] L. de Alfaro and T. A. Henzinger, "Interface automata," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, Sep. 2001. [Online]. Available: https://doi.org/10.1145/503271.503226

[79] M. Bernaerts, B. Oakes, K. Vanherpen, B. Aelvoet, H. Vangheluwe, and J. Denil, "Validating industrial requirements with a contract-based approach," in *International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 18–27.

[80] A. Arrieta, J. A. Agirre, and G. Sagardui, "Seeding strategies for multi-objective test case selection: An application on simulation-based testing," in *Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 2020, p. 1222–1231.

[81] X. Sun, P. Nuzzo, C. Wu, and A. Sangiovanni-Vincentelli, "Contract-based system-level composition of analog circuits," in *Design Automation Conference*. ACM, 2009.

[82] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia, "A contract-based methodology for aircraft electric power system design," *IEEE Access*, vol. 2, pp. 1–25, 2014.

[83] P. Nuzzo, J. B. Finn, A. Iannopollo, and A. L. Sangiovanni-Vincentelli, "Contract-based design of control protocols for safety-critical cyber-physical systems," in *Design, Automation & Test in Europe Conference & Exhibition, (DATE)*. European Design and Automation Association, 2014.

[84] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems, Second Edition*. Springer, 2008.

[85] M. Gheorghiu Bobaru, C. S. Păsăreanu, and D. Giannakopoulou, "Automated assume-guarantee reasoning by abstraction refinement," in *International Conference on Computer Aided Verification (CAV)*. Springer-Verlag, 2008, p. 135–148.

[86] S. Maoz, J. O. Ringert, and R. Shalom, "Symbolic repairs for GR(1) specifications," in *International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1016–1026.

[87] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM journal on control and optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[88] P. J. Ramadge and M. Wonham W, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[89] N. Piterman, M. Keegan, V. Braberman, N. D'Ippolito, and S. Uchitel, "Control and discovery of reactive system environments," in *University of Leicester*, 2020.

[90] S. Lin, E. Andre, Y. Liu, J. Sun, and J. Dong, "Learning assumptions for compositional verification of timed systems," *IEEE Transactions on Software Engineering*, vol. 40, no. 02, pp. 137–153, feb 2014.

[91] A. Kampmann, N. Havrikov, E. O. Soremekun, and A. Zeller, "When does my program do this? learning circumstances of software behavior," in *Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020.

[92] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, vol. 69, no. 1, pp. 35 – 45, 2007.

[93] Y. Chen, C. M. Poskitt, and J. Sun, "Towards learning and verifying invariants of cyber-physical systems by code mutation," in *International Symposium on Formal Methods*. Springer, 2016, pp. 155–163.

[94] D. Gopinath, H. Converse, C. S. Păsăreanu, and A. Taly, "Property inference for deep neural networks," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, p. 797–809.

[95] C. Flanagan and K. R. M. Leino, "Houdini, an Annotation Assistant for ESC/Java," in *International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity (FME)*. Springer-Verlag, 2001, p. 500–517.

[96] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The Daikon System for Dynamic Detection of Likely Invariants," *Science of computer programming*, vol. 69, no. 1–3, p. 35–45, 2007.

[97] P. Garg, C. L"oding, P. Madhusudan, and D. Neider, "ICE: A robust framework for learning invariants," in *Computer Aided Verification (CAV)*. Springer, 2014.

[98] P. Garg, D. Neider, P. Madhusudan, and D. Roth, "Learning invariants using decision trees and implication counterexamples," *SIGPLAN Not.*, vol. 51, no. 1, p. 499–512, 2016.

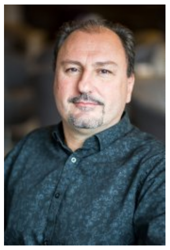
[99] W. Li, L. Dworkin, and S. A. Seshia, "Mining assumptions for synthesis," in *International Conference on Formal Methods and Models*. IEEE, 2011, pp. 43–50.

[100] R. Alur, S. Moarref, and U. Topcu, "Counter-strategy guided refinement of GR(1) temporal logic specifications," in *Formal Methods in Computer-Aided Design, FMCAD*. IEEE, 2013, pp. 26–33.
[101] M. Keegan, V. A. Braberman, N. D'Ippolito, N. Piterman, and S. Uchitel, "Control and discovery of environment behaviour," *IEEE Transactions on Software Engineering*, 2020.
[102] S. Sato, M. Waga, and I. Hasuo, "Constrained optimization for falsification and conjunctive synthesis," *arXiv preprint arXiv:2012.00319*, 2020.
[103] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.
[104] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," in *International conference on Hybrid systems: computation and control, HSCC*. ACM, 2013.
[105] R. V. Borges, A. d'Avila Garcez, L. C. Lamb, and B. Nuseibeh, "Learning to adapt requirements specifications of evolving systems," in *International Conference on Software Engineering (ICSE)*. IEEE, 2011.
[106] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat, "Combinatorial sketching for finite programs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2006.
[107] E. Kitzelmann, "Inductive programming: A survey of program synthesis techniques," in *International workshop on approaches and applications of inductive programming*. Springer, 2009, pp. 50–73.
[108] S. Gulwani, O. Polozov, R. Singh *et al.*, "Program synthesis," *Foundations and Trends® in Programming Languages*, vol. 4, no. 1-2, pp. 1–119, 2017.
[109] H. Byun and S.-W. Lee, "Applications of support vector machines for pattern recognition: A survey," in *International workshop on support vector machines*. Springer, 2002, pp. 213–236.

Khouloud Gaaloul is a Ph.D. candidate at the Inter-disciplinary Centre for Security, Reliability and Trust(SnT), University of Luxembourg. She received her Engineering degree from the Higher National School of Engineering of Tunis (ENSIT) in 2016. Her research interests include software engineering, focusing on search-based testing, empirical verification methods, machine learning and analysis of Cyber-Physical Systems.

Claudio Menghi is a Research Associate at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), at the University of Luxembourg. After receiving his Ph.D. at Politecnico di Milano in 2015, he was a post-doctoral researcher at Chalmers University of Technology and University of Gothenburg. His research interests are in software engineering, with a special interest in cyber-physical systems (CPS) and formal verification.

Shiva Nejati is an Associate Professor at the School of Electrical Engineering and Computer Science of the University of Ottawa and a part-time Research Scientist at the SnT Centre for Security, Reliability, and Trust, University of Luxembourg. From 2009 to 2012, she was a researcher at Simula Research Laboratory in Norway. She received her Ph.D. degree from the University of Toronto, Canada in 2008. Nejati's research interests are in software engineering, focusing on model-based development, software testing, analysis of cyber-physical systems, search-based software engineering and formal and empirical software engineering methods. Nejati has coauthored over 60 journal and conference papers, and regularly serves on the program committees of international conferences in the area of software engineering. She has for the past ten years been conducting her research in close collaboration with industry partners in telecommunication, maritime, energy, automotive and aerospace sectors.



Lionel C. Briand is professor of software engineering and has shared appointments between (1) The University of Ottawa, Canada and (2) The SnT Centre for Security, Reliability, and Trust, University of Luxembourg. He is currently running multiple collaborative research projects with companies in the automotive, satellite, financial, and legal domains. Lionel has held various engineering, academic, and leading positions in six countries. He was one of the founders of the ICST conference (IEEE Int. Conf. on Software Testing,Verification, and Validation, a CORE A event) and its first general chair. He was also EiC of Empirical Software Engineering (Springer) for 13 years and led, in collaboration with first Victor Basili and then Tom Zimmermann, the journal to the top tier of the very best publication venues in software engineering. Lionel was elevated to the grades of IEEE and ACM Fellow. He was also granted the IEEE Computer Society Harlan Mills award and the IEEE Reliability Society engineer-of-the-year award for his work on model-based verification and testing, respectively in 2012 and 2013. He received an ERC Advanced grant in 2016 – on the topic of modeling and testing cyber-physical systems – which is the most prestigious individual research award in the European Union. More recently, he was awarded a Canada Research Chair (Tier 1) on "Intelligent Software Dependability and Compliance". His research interests include: software testing and verification, model-driven software development, applications of AI in software engineering, and empirical software engineering.

Yago Isasi Parache is Head of the Software division at LuxSpace Sàrl, and leads the development of satellite simulators, satellite on-board software, satellite data applications, software product assurance, and software tooling for space software development. He holds a Diploma of Advanced Studies (DEA - Master Degree) in Applied Mathematics from Universitat Politècnica de Catalunya, a Diploma of Advanced Studies (DEA - Master Degree) in Astrophysics, Particle Physics, and Cosmology from Universita de Barcelona, and a degree in Industrial Engineering from Universitat Politècnica de Catalunya.