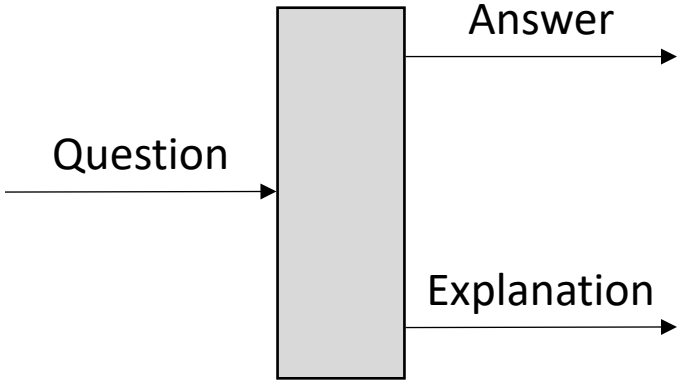
2nd International Workshop on Explainability of Real-time Systems and their Analysis

at the IEEE Real-Time Systems Symposium

Taipei, Taiwan, December 5, 2023



It is our pleasure to welcome you to the 2nd International Workshop on Explainability of Real-time Systems and their Analysis (ERSA). This workshop is held on December 5, 2023 in Taipei, Taiwan in conjunction with IEEE Real-Time Systems Symposium (RTSS). Last year we started this workshop because we wanted to explore whether the notion of explainability is helpful for the real-time system research community in order to deliver more value to software practitioners—in particular those involved in certification. This document is the workshop proceeding for ERSA'23.

We thank several individuals and institutions without whom this workshop would not have been possible. This includes:

1. the authors for providing technical content;
2. the members of the technical program committee for evaluating the content and providing constructive feedback to the authors;
3. the organizers of RTSS that gave "go-ahead" for ERSA to take place; this includes Angeliki Kritikakou (Hot-Topics Day Chair of RTSS);
4. the National Taiwan University for supporting ERSA;
5. people working behind the scene to provide (digital and physical) infrastructure, advice, and proof reading.

The papers in this workshop proceeding provide new ideas on explainability. Some of the papers build on ideas from last year. This is a testament to forward progress of the area and our community. We believe and hope you will find the papers interesting; and that they will help you and help us all in defining this new area of research.

Sincerely,

Chi-Sheng (Daniel) Shih
Bjorn Andersson
Co-Chairs of ERSA'23

Program Co-Chairs

Chi-Sheng (Daniel) Shih, National Taiwan University, Taiwan
Bjorn Andersson, SEI/CMU, USA

Program Committee

Ahlem Mifdaoui, UToulouse, France
Al Mok, UTexas, USA
C. Michael Holloway, NASA, USA
Carol Smith, SEI/CMU, USA
Chung-Wei Lin, NTU, Taiwan
David Cole, DanLAW, USA
Dionisio de Niz, SEI/CMU, USA
George Romanski, FAA, USA
Gernot Heiser, UNSW Sydney, Australia
Guillem Bernat, Rapita, UK
Hyoseung Kim, UCR, USA
Iain Bate, UYork, UK
Isaac Amundson, Collins Aerospace, USA
John Lehoczky, CMU, USA
Manabu Tsukada, UTokyo, Japan
Mark Klein, SEI/CMU, USA
Sanjoy Baruah, WUSTL, USA
Shambwaditya Saha, SEI/CMU, USA
Shige Wang, Motional, USA
Willie Fitzpatrick, AvMC, USA
Violet Turri, SEI/CMU, USA

Explaining Quadratic Boundedness for Latency Mitigation and Safety Assurance in Edge-Cloud Computing

Raffaele Romagnoli

***Abstract*— A novel paradigm of control relocates process controls to the edge-cloud, offering increased computational resources and proximity to sensors. However, network latency remains a key limitation, treated as missing updates in control loops. This study explores the system's robustness to missing updates, leveraging Lyapunov-like methods such as quadratic boundedness. The goal is to identify Lyapunov level sets that remain bounded despite missing control updates. This approach, initially designed for edge-cloud control, has broader applications, including aerospace. This note presents our approach, focusing on linear time-invariant (LTI) systems.**

I. Introduction

Modern industry is currently undergoing a convergence of two pivotal domains: information technology (IT) systems and operational technology (OT) systems [1]. IT systems encompass the technologies and infrastructure employed for processing, storing, and transmitting information to users, applications, and other IT systems. In contrast, OT systems are responsible for overseeing and controlling physical processes. While maintaining a separation between IT and OT systems remains crucial for safety considerations, contemporary industry demands rapid responsiveness. Flexibility within manufacturing is paramount, and a key component of achieving this goal lies in integrating IT and OT systems. In this vein, a groundbreaking paradigm of networked control has emerged, aiming to shift process controls to the edge-cloud—an infrastructure situated in close proximity to process sensors and actuators [2]. This innovative edge-cloud architecture augments computational resources, enabling the implementation of more advanced applications that were previously unattainable with standard industrial devices like programmable logic controllers (PLCs). Simultaneously, the proximity to sensors and actuators guarantees availability and determinism for the controller hosted within the edge-cloud.

One primary limitation of this novel control architecture pertains to network latency and reliability. Delays within the control loop pose significant risks as they can compromise system stability. In this context, these delays are treated as missing updates to control inputs. When a control update is absent, the last successfully transmitted control input is retained until a new, timely update arrives [3].

Our primary objectives involve assessing the system's robustness concerning missing updates and establishing safety conditions. Our chosen approach involves harnessing Lyapunov-like methods, specifically quadratic boundedness (QB) [4], [5]. We model the missing updates as external disturbances, with the goal of identifying a Lyapunov level set that is quadratically bounded concerning a sequence of missing control updates. This implies the identification of a set where all system trajectories, subjected to bounded disturbances, remain within.

The idea of modeling missing updates as disturbances has been previously introduced in our work [6], where we discussed the potential benefits of developing explainable tools for assessing the safety of a cyber-physical system (CPS). In this paper, we reframe the concept of quadratic boundedness in relation to the maximum sequence of admissible missing updates required to ensure system safety. Additionally, we introduce an algorithm capable of determining the parameters that confirm the quadratic boundedness of systems with respect to missing updates.

The issue we investigate in this study can be situated within the framework of weakly-hard real-time (WHRT) constraints [3], [7], [8]. While these papers tackle the stability problem, our work is primarily centered on safety, which is generally considered to be a less stringent requirement than stability. This distinction serves as one of the motivations behind our utilization of quadratic boundedness [6].

While this paper is primarily motivated by a novel control paradigm rooted in edge-cloud computing, the general problem exposition and the approach presented here for analyzing the effects of missing updates can be extended to other scenarios, including aerospace applications. In this concise note, we present our approach within the context of linear time-invariant (LTI) systems.

II. Problem Statement

We consider the sampled data version of an LTI dynamical system represented as follows:

$$x_{k+1} = Ax_k + Bu_k, \tag{1}$$

where $x_k \in \mathbb{R}^n$ represents the state vector of the system with n components, and $u_k \in \mathbb{R}^p$ represents the input vector with p components. The matrices $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times p}$ describe the sampled-data dynamics and inputs. The subscript $k \geq 0$ corresponds to the sampled time step with $k \in \mathbb{N}$.

We employ the following state feedback controller to ensure the asymptotic stability of (1):

$$u_k = -Kx_k, \tag{2}$$

where $K \in \mathbb{R}^{p\times n}$. If this controller remains unaffected

R. Romagnoli is with the Dept. of Electrical and Computer Engineering, Carnegie Mellon University (CMU), Pittsburgh, PA, USA 15235. Email: {rromagno}@andrew.cmu.edu

by missing control updates, then the closed-loop system, obtained by substituting (2) into (1), becomes:

$$x_{k+1} = (A - BK)x_k, \tag{3}$$

where $(A-BK)$ forms a Schur matrix, indicating that all eigenvalues reside within the unitary circle in the complex plane.

While the controller (2) can be readily implemented in a PLC (Programmable Logic Controller), we use it as a proof of concept in this paper. The approach presented here can be extended to more advanced controllers that are not feasible for common PLCs.

In the event of network delays or disruptions, the control input is not updated, rendering the closed-loop formulation (3) invalid. Specifically, assuming that the last control input was applied at $k = 0$, the evolution of the sampled-data system is described as:

$$x_{k+1} = Ax_k - BKx_0, \tag{4}$$

where $-BKx_0$ remains a constant value until a new valid update of the control law $-Kx_k$ arrives, and the system can then be described again using (3).

A. Model Formulation

One way to address (4) is by expressing it as follows, introducing the term $\pm BKx_k$:

$$x_{k+1} = (A - BK)x_k + BK(x_k - x_0), \tag{5}$$

where the second term can be viewed as an external disturbance $d_k \triangleq K(x_k - x_0)$. This formulation enables us to utilize the quadratic Lyapunov function associated with the closed-loop system (3):

$$V(x_k) = x_k^T P x_k, \tag{6}$$

where $P > 0$ is a symmetric positive definite matrix that satisfies the Lyapunov equation:

$$(A - BK)^T P(A - BK) - P = -Q, \tag{7}$$

with $Q > 0$ being a given symmetric positive definite matrix. Since (3) is asymptotically stable, according to Lyapunov's theory [9], we have:

$$V(x_k) - V(x_{k-1}) < 0 \quad \text{for all } k \geq 0.$$

In summary, during consecutive missing updates, the system can be described as an asymptotically stable system subjected to an external disturbance:

$$x_{k+1} = A_f x_k + Bd_k, \tag{8}$$

where $A_f \triangleq (A - BK)$. This formulation allows us to leverage the Lyapunov function (6) to define Lyapunov level sets, which manifest as ellipsoids:

$$\mathcal{E}_c = \left\{ x \in \mathbb{R}^n \mid x^T P x \leq c \right\}, \tag{9}$$

where $c \in \mathbb{R}$. In cases where $d_k = 0$, a Lyapunov level set is "positively invariant" [9]. Given that $V(x_k)$ continually decreases, if the system initiates from any point on the border of (9), its behavior remains confined within (9). However, in instances of missing control updates ($d_k \neq 0$), this perturbation may disrupt the positive invariance of (9). To explore this scenario, we introduce quadratic boundedness [4], which is further discussed in the subsequent subsection.

B. Quadratic boundedness

Consider the system (4), with the assumption that $\|d_k\|_2 \leq \delta$ for all $k \geq 0$. A system is considered "quadratically bounded" if there exists a scalar $\eta > 0$ and a symmetric positive definite matrix P satisfying the following conditions:

$$\begin{aligned} &x_k^T P x_k > \eta \quad \rightarrow \\ &(A_f x_k + Bd_k)^T P(A_f x_k + Bd_k) - x_k^T P x_k < 0, \\ &\qquad \forall \|d\|_2 \leq \delta, \ \forall k \geq 0. \end{aligned} \tag{10}$$

In this context, the ellipsoid $\mathcal{E}_\eta$ defines a positively invariant set. The objective is to associate $\mathcal{E}_\eta$ with a maximum number of missing control updates, ensuring that the system remains confined within the ellipsoid. This guarantees the controller's robustness against network delays. Using the new formulation (4), the matrix $P > 0$ can be computed by solving the Lyapunov equation (7). Additionally, as demonstrated in [4], it is possible to compute an overapproximation of η, provided that δ is known.

In practice, δ is often unknown and even if a reasonable approximation can be obtained, the resulting values of η may be excessively conservative. This means that the obtained ellipsoid may not fit the real application requirements. In the following section, we introduce an alternative approach to describe quadratic boundedness in cases involving missing control input updates, along with a procedure for finding a practical η.

III. Proposed Solution

Building upon the previous formulation regarding quadratic boundedness, we explore an alternative method to demonstrate the system's quadratic boundedness concerning missing control updates for a specific value of η.

We commence by considering a positive scalar, denoted as $\eta' > 0$, which defines $\mathcal{E}_{\eta'}$ in alignment with the practical specifications of the problem. Subsequently, we contemplate any initial condition, denoted as x_0, situated outside of $\mathcal{E}_{\eta'}$. Furthermore, we assume that the control input undergoes an update, yielding $x_1 = A_f x_0$. This leads us to the following inequality, utilizing expression (6):

$$V(x_0) > V(x_1). \tag{11}$$

From this point onwards, we shift our focus to a sequence encompassing $\bar{k}$ consecutive missing control updates. As per equation (5):

$$x_{k+1} = A_f x_k - BK(x_k - x_0) \tag{12}$$

where k takes values in the range $k \in [1, ..., \bar{k}]$. Subsequently, we make the assumption that at $\bar{k}+1$, the control input is updated. Therefore, $x_{\bar{k}+1}$ represents the new x_0, and this initiates the repetition of the missing sequence of control inputs. Within the framework that solely considers one update following $\bar{k}$ missing updates, the system is said to be quadratically bounded concerning a sequence of consecutive missing $\bar{k}$ control updates for a given $\eta' > 0$ if the following condition is satisfied for all x_0:

$$\forall x_0 : V(x_0) > \eta' \rightarrow V(x_{\bar{k}+1}) < V(x_0). \tag{13}$$

Additionally, the above definition implies:

$$\forall x_0 : V(x_0) < \eta' \rightarrow V(x_{\bar{k}+1}) < \eta'. \tag{14}$$

Ultimately, when considering a domain, denoted as $\mathcal{D} \subset \mathbb{R}^n$, which encompasses $\mathcal{E}_{\eta'}$, if the subsequent condition holds:

$$\forall x_0 \in \mathcal{D} \rightarrow V(x_{\bar{k}+1}) < V(x_0). \tag{15}$$

Then, the system is deemed asymptotically stable, directly implying quadratic boundedness.

The underlying rationale for the aforementioned definitions is that there exists a value of η' such that for any x_0 where $V(x_0) > \eta'$, the constant control input $-Kx_0$ maintains its stabilizing effect for a certain duration during network disruptions. At a certain point, when the error $\|x_k - x_0\|_2$ becomes too substantial to sustain that stabilizing effect, the Lyapunov function begins to increase. However, this increment remains smaller than the decrease induced by the lack of an updated control input. If the initial condition x_0 satisfies $V(x_0) < \eta'$, then the potential increase in the Lyapunov function cannot exceed η' for a given $\bar{k}$. This behavior is illustrated in Fig. 1.

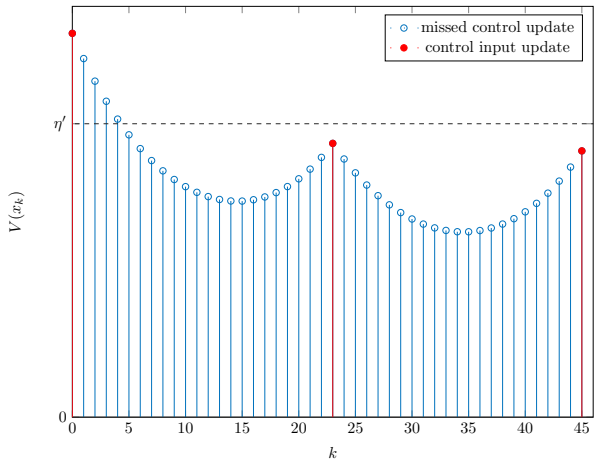


Fig. 1: Behavior of $V(x_k)$ updating the control input after $\bar{k} = 22$ consecutive missing updates.

A. LTI Systems

For LTI systems, we consider a generic initial condition $x_0 \in \mathcal{D}$, and we assume that the controls are regularly updated, leading to:

$$x_1 = (A - BK)x_0 = A_f x_0. \tag{16}$$

Subsequently, we delve into the evolution of the system (4) with $\bar{k}$ consecutive missing control updates:

$$\begin{aligned} x_2 &= Ax_1 - BKx_0 \\ &= (AA_f - BK)x_0 \\ x_3 &= Ax_2 - BKx_0 \\ &= (A^2A_f - ABK - BK)x_0. \end{aligned}$$

Generalizing for $\bar{k}$, we have:

$$x_{\bar{k}+1} = \left(A^{\bar{k}} A_f - \sum_{j=0}^{\bar{k}-1} A^j BK \right) x_0 = A_1(\bar{k}) x_0. \tag{17}$$

Considering the asymptotic stability condition (15):

$$V(x_{\bar{k}+1}) - V(x_0) < 0. \tag{18}$$

By using (17), this condition is represented as:

$$x_0^T A_1(\bar{k})^T P A_1(\bar{k}) x_0 - x_0^T P x_0 < 0. \tag{19}$$

Remarkably, this condition to be verified is independent of the initial condition x_0. It is formulated as:

$$A_1(\bar{k})^T P A_1(\bar{k}) - P < 0. \tag{20}$$

This result is more in the spirit of the WHRT framework. In this work, we aim to use a less conservative result based on quadratic boundedness, which is represented by the following condition:

$$x_0^T A_1(\bar{k})^T P A_1(\bar{k}) x_0 - \eta' \leq 0. \tag{21}$$

In this formulation, (21) has to be evaluated for any initial condition outside of $\mathcal{E}_{\eta'}$. In the subsequent subsection, we describe a procedure to determine the maximum value of $\bar{k}$ for a given η'.

B. Quadratic Boundedness Procedure

In order to address the issue of initial conditions, we present a procedure based on reach set analysis. Let us consider $\mathcal{E}_1 \in \mathcal{D}$ as a maximal ellipsoid contained in $\mathcal{D}$. We approximate it with a polytope:

$$\mathcal{P}_1 = \left\{ x \in \mathbb{R}^n : x = \sum_{j=1}^{m} \alpha_j x_j, 0 \leq \alpha_j \leq 1, \sum_{j=1}^{m} \alpha_j = 1 \right\}, \tag{22}$$

such that $\mathcal{E}_1 \subseteq \mathcal{P}_1$. The points $x_j \in \mathbb{R}^n$ represent the vertices of the polytope. By selecting $0 < \eta' < 1$, in the case of an LTI system, we can evaluate (17) for each vertex x_j as initial conditions. A pseudo-algorithm to find $k_{\max}$ is outlined in Algorithm 1.

Algorithm 1 Determine $k_{\max}$

Input: η', $\{x_1, x_2, \ldots, x_m\}$ **Output:** $k_{\max}$

for $j = 1$ to m **do**
  $x_0 \leftarrow x_j$
  $\bar{k} \leftarrow 1$
  while $V(x_{\bar{k}+1}) \leq \eta' \,||\, V(x_{\bar{k}+1}) \leq V(x_{\bar{k}})$ **do**
    $\bar{k} \leftarrow \bar{k} + 1$
    $x_{\bar{k}+1} \leftarrow$ evaluate (17)
  end while
  $\bar{k}_j \leftarrow \bar{k}$
end for
$k_{\max} \leftarrow \min(\bar{k}_j)$

Fig. 2 shows the starting point of the algorithm. The ellipsoid $\mathcal{E}_1$ represents an inner approximation of $\mathcal{D}$. The polytope $\mathcal{P}_1$ is an over-approximation of $\mathcal{E}_1$, and the relationship holds: $\mathcal{E}_1 \subset \mathcal{P}_1 \subset \mathcal{D}$.

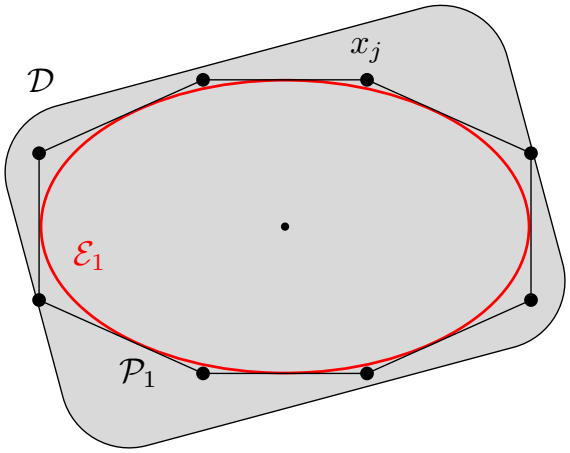


Fig. 2: Initialization of Algorithm 1. The domain $\mathcal{D}$ is approximated with $\mathcal{E}_1$, whose over-approximation is the polytope $\mathcal{P}_1$.

We consider $\eta' < 1$, and for each vertex x_j, we evaluate (17). Initially, the control input update has a stabilizing effect. To capture this effect for the subsequent missing control updates, the condition to be evaluated for each initial condition is:

$$V(x_{\bar{k}+1}) \leq \eta' \,||\, V(x_{\bar{k}+1}) \leq V(x_{\bar{k}}). \tag{23}$$

At the beginning, the state of the system can still be outside of $\mathcal{E}_{\eta'}$, but its behavior is moving towards it, meaning that $V(\cdot)$ is decreasing. Once it is not decreasing, we need to evaluate if the state of the system is still inside $\mathcal{E}_{\eta'}$, which is the reason for the OR condition. Fig. 3 describes what happens to the polytope $\mathcal{P}_1$ when Algorithm 1 provides a value of $k_{\max} > 1$. Thanks to the stabilizing effect of the control update, each vertex moves towards $\mathcal{E}_{\eta'}$. This behavior is maintained during the missing control updates until it reaches the smallest polytope inside $\mathcal{E}_{\eta'}$. After that, each vertex begins to diverge until at least one of them reaches the border of $\mathcal{E}_{\eta'}$.

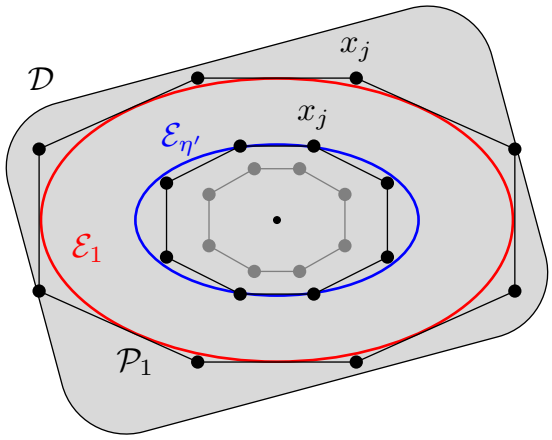


Fig. 3: Behavior of $\mathcal{P}_1$ when Algorithm 1 provide $k_{\max} > 1$ for a given η'. The gray polytope is the smallest obtained by the stabilizing effect of the initial control update.

Algorithm 1 may not provide a solution $k_{\max} > 1$. In that case, different values of η' should be considered. The algorithm also considers (17), but this method is more general because it can handle more sophisticated controls and models. In contrast, the condition (20) is valid only for assessing asymptotic stability for LTI systems with a linear state feedback controller (2).

IV. Conclusions

In this note, we have provided both a theoretical analysis and a practical procedure for ascertaining the quadratic boundedness of a system that experiences intermittent control updates. Our motivation for this endeavor stems from an innovative control framework rooted in edge-cloud computing.

References

[1] P. Ackerman, *Industrial Cybersecurity: Efficiently monitor the cybersecurity posture of your ICS environment*. Packt Publishing Ltd, 2021.

[2] M. Giani, N. Frank, and A. Verl, "Towards industrial control from the edge-cloud: A structural analysis of adoption challenges according to industrial experts," in *Proceedings of the 12th International Conference on the Internet of Things*, pp. 17–24, 2022.

[3] M. Hertneck, S. Linsenmayer, and F. Allgöwer, "Stability analysis for nonlinear weakly hard real-time control systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 2594–2599, 2020.

[4] M. L. Brockman and M. Corless, "Quadratic boundedness of nonlinear dynamical systems," in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, vol. 1, pp. 504–509, IEEE, 1995.

[5] M. L. Brockman and M. Corless, "Quadratic boundedness of nominally linear systems," *International Journal of Control*, vol. 71, no. 6, pp. 1105–1117, 1998.

[6] R. Romagnoli, "Understanding safety of linear real-time systems from lyapunov theory and quadratic boundedness," 2022.

[7] S. Linsenmayer, M. Hertneck, and F. Allgöwer, "Linear weakly hard real-time control systems: Time-and event-triggered stabilization," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1932–1939, 2020.

[8] N. Vreman, P. Pazzaglia, V. Magron, J. Wang, and M. Maggio, "Stability of linear systems under extended weakly-hard constraints," *IEEE Control Systems Letters*, 2022.

[9] H. K. Khalil, *Nonlinear systems; 3rd ed.* Upper Saddle River, NJ: Prentice-Hall, 2002.

Efficient Explainability of Real-Time Schedulability

Sanjoy Baruah
Washington University in Saint Louis
baruah@wustl.edu

Pontus Ekberg
Uppsala University
pontus.ekberg@it.uu.se

***Abstract*—We had previously [1] proposed a rigorous definition of what it means for a safety property to be *efficiently explainable*: there should exist a certificate, whose validity may be checked by a deterministic algorithm in polynomial time, attesting to the safety of any system satisfying the property. Here we explore a more generalized notion of efficient explainability in which the validity of the certificate may be checked in a probabilistic sense (rather than only a deterministic one).**

***Index Terms*—Schedulability; Polynomial-time Verifiability; Randomized Verification.**

The first edition of this workshop threw up several alternative, all valid, interpretations of 'EXPLAINABILITY,' ranging from Andersson's rather informal perspective [2] that an explanation should be understandable to a non-expert, to far more formal interpretations such as the one articulated by Brandenburg during a panel discussion at the workshop that an explanation should be expressible in, and hence rigorously verifiable within, some machine-checkable formalism such as Prosa [3]. The authors of this submission had also provided a rigorous and formal perspective [1] as to what constitutes an acceptable explanation for a claim that a particular system satisfies a particular safety property; the central idea in [1] may be summarized as follows.

> By interpreting the set of all system specifications that satisfy a particular safety property as a *formal language* [4], and the explanation for a particular input system specification as a *certificate* attesting to the membership of that system specification in this formal language, the existence of explanations becomes closely related to several well-studied problems in computational complexity theory [5], [6]. In particular, explainable safety properties are exactly those for which the associated verification problems belong to well-defined complexity classes.

(From this perspective the Halting Problem [7] —the problem of determining, from a description of an arbitrary computer program P and an input e, whether the program will halt when executed upon this input— is explainable: for a given program P on input e, an acceptable certificate is simply the total number of steps that P executes on e before completing and halting. However, the complementary problem, that of determining whether P executes without halting on input e, is not explainable, as it is well-known that the complement of the halting problem is not recursively enumerable.)

This particular interpretation of explainability was investigated further in [8] by the authors of this submission, with a focus on *efficient* explainability: what are the safety properties for which there exist explanations that can be efficiently verified as being correct (or rejected for being erroneous – for failing to actually establish safety)? The central idea in [1], [8] can be summarized in the following proposition. Let us define a safety property to be *efficiently explainable* if for any system satisfying the safety property, there exists an explanation of this fact that can be validated for correctness by a deterministic procedure in time no worse than polynomial in the representation of the system; this definition simply equates efficient explainability with the complexity class NP.

Proposition 1.

- Any safety property for which the associated verification problem belongs to the complexity class NP is efficiently explainable; and
- Showing that the verification problem associated with a safety property is hard for a complexity class that is unlikely to be contained within NP offers strong evidence of that property not being efficiently explainable. □

The application of Proposition 1 was illustrated in our prior work [1], [8] upon several example problems concerning real-time schedulability analysis, including in particular (i) preemptive uniprocessor fixed-priority (FP) schedulability of constrained-deadline sporadic task systems (see, e.g., [9]–[11] for a description of this problem), and (ii) preemptive uniprocessor earliest-deadline-first (EDF) schedulability of sporadic task systems (this problem is described in, e.g., [12], [13]). It was pointed out that since FP-schedulability of constrained-deadline sporadic task systems is NP-complete [14], [15], it is in NP and hence efficiently explainable. In contrast, EDF-schedulability of sporadic task systems is known to be coNP-complete [16] and hence coNP-hard; since it is widely believed that coNP $\not\subseteq$ NP, this immediately implies that EDF-schedulability of sporadic task systems is unlikely to be efficiently explainable. In [8] this issue was addressed both via identifying efficiently explainable (NP) subsets of this coNP-hard problem, and via considering a couple of wider notions of efficient explainability, the latter in the form of pseudo-polynomial time verifiability (as captured by the class pseudoNP) and fully-polynomial time verification approximation schemes (FTPVAS). In this note we want to make the case for another intriguing possibility, the explainability of real-time schedulability using interactive proof systems.

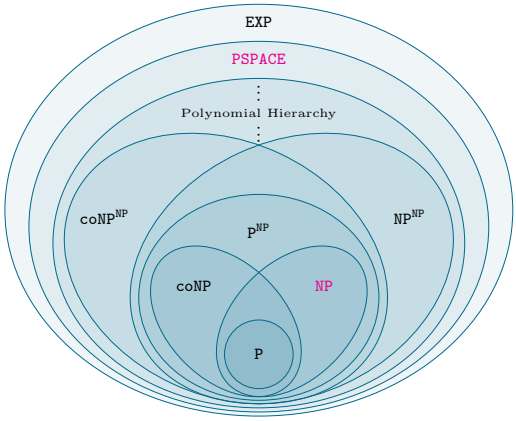


Fig. 1. Some computational complexity classes

Randomized verification

Verification of safety-critical software has traditionally been a conservative endeavor, particularly when performed as part of a certification process in highly regulated domains such as civilian aviation or nuclear power control. It is interesting to speculate on the possibilities that open up if we were to settle for *randomized*, rather than purely deterministic, verification of safety claims. That is, rather than requiring, as current safety standards tend to do, that the correctness of a system be validated with absolute certainty, what if we would settle for a safety argument that convinces us of a system's safety at an arbitrarily high probability (that is strictly smaller than one — say, $(1 - 10^{-6})$)? If such randomized verification were to be considered acceptable, this opens up the possibility that rather than being a statically-generated certificate, an explanation be permitted to be of the form of an interactive dialog whereby a verifier makes repeated queries in order to develop adequate confidence in the veracity of a claim that a system satisfies a particular safety property. The reason why this would be a significant development builds upon a well-known result in complexity theory from the 1990's [17], establishing that the class of decision problems that can be verified in polynomial time by such an interactive randomized verifier (which communicates with a *prover* that is not polynomially bounded) is exactly the complexity class PSPACE. Hence if randomized interactive verification of safety properties were to be considered acceptable practice for the purposes of safety verification, then the class of efficiently explainable properties (i.e., the safety properties for which there exist polynomial-time verifiable interactive explanations for all systems satisfying the safety property) becomes the class of all safety properties for which the associated verification problem belongs to PSPACE. And as we can see in Figure 1, this complexity class is considerably larger than the class NP (which, recall, is the class of safety properties currently considered efficiently explainable – see Propsition 1. For instance, EDF-schedulability of sporadic task systems was shown to not be efficiently explainable under the prior definition (of deterministic verification); however since EDF-schedulability of sporadic task systems is coNP-complete and coNP $\subseteq$ PSPACE, it can be verified in polynomial time by a randomized interactive verifier. In a similar vein, global EDF- and FP-schedulability for sporadic task systems are both known to be in PSPACE [18], [19], and schedulability analysis for conditional DAG tasks is PSPACE-complete [20], [21]; hence such schedulability, too, can be verified in polynomial time by randomized interactive verifiers.

The possibility of interactive randomized verification of safety-critical systems becoming accepted practice opens up several interesting avenues of research, amongst them being the derivation of interactive proofs for important schedulability analysis problems that, in addition to having polynomially-bounded running time, are computationally reasonably efficient in practice. We point out that there are other, related, ongoing research efforts in this direction; see, e.g. [22].

References

[1] S. Baruah and P. Ekberg, "Certificates of real-time schedulability," in *International Workshop on Explainability of Real-time Systems and their Analysis (ERSA)*, 2022.

[2] B. Andersson, "The case for explainability of real-time systems," 2022, presentation at International Workshop on Explainability of Real-time Systems and their Analysis (ERSA); PowerPoint slides available at https://sites.google.com/view/ersa22 (Date checked: 2023/ 09/13).

[3] F. Cerqueira, F. Stutz, and B. B. Brandenburg, "PROSA: A case for readable mechanized schedulability analysis," in *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 273–284.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction To Automata Theory, Languages, And Computation*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[5] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.

[6] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. [Online]. Available: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264

[7] A. Turing, "On computable numbers, with an application to the Entscheidungsproblem," in *Proceedings of the London Mathematical Society*, ser. 2, no. 42, 1936, pp. 230–265, correction in 43 (1937), pages 544–546.

[8] S. Baruah and P. Ekberg, "Towards Efficient Explainability of Schedulability Properties in Real-Time Systems," in *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), A. V. Papadopoulos, Ed., vol. 262. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 2:1–2:20. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2023/18031

[9] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, Oct. 1986.

[10] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proceedings of the 10th Real-Time Systems Symposium (RTSS)*. Santa Monica, California, USA: IEEE Computer Society Press, Dec. 1989, pp. 166–171.

[11] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach," in *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, May 1991, pp. 127–132.

[12] J. Y.-T. Leung and M. Merrill, "A note on the preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, pp. 115–118, 1980.

[13] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*. Orlando, Florida: IEEE Computer Society Press, 1990, pp. 182–190.

[14] F. Eisenbrand and T. Rothvoß, "Static-priority real-time scheduling: Response time computation is NP-hard," in *Proceedings of the 29th Real-Time Systems Symposium (RTSS)*. Barcelona: IEEE Computer Society Press, December 2008.

[15] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard," in *Proceedings of the 38th Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 2017, pp. 139–146. [Online]. Available: https://doi.org/10.1109/RTSS.2017.00020

[16] F. Eisenbrand and T. Rothvoß, "EDF-schedulability of synchronous periodic task systems is coNP-hard," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2010.

[17] A. Shamir, "IP = PSPACE," *Journal of the ACM*, vol. 39, no. 4, pp. 869–877, oct 1992. [Online]. Available: https://doi.org/10.1145/146585.146609

[18] V. Bonifaci and A. Marchetti-Spaccamela, "Feasibility analysis of sporadic real-time multiprocessor task systems," *Algorithmica*, vol. 63, no. 4, pp. 763–780, 2012. [Online]. Available: https://doi.org/10.1007/s00453-011-9505-6

[19] G. Geeraerts, J. Goossens, and M. Lindström, "Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm," *Real Time Systems*, vol. 49, no. 2, pp. 171–218, 2013. [Online]. Available: https://doi.org/10.1007/s11241-012-9172-y

[20] S. Baruah and A. Marchetti-Spaccamela, "Feasibility Analysis of Conditional DAG Tasks," in *Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 12:1–12:17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13943

[21] ——, "The computational complexity of feasibility analysis for conditional DAG tasks," *ACM Trans. Parallel Comput.*, jul 2023. [Online]. Available: https://doi.org/10.1145/3606342

[22] E. Couillard, P. Czerner, J. Esparza, and R. Majumdar, "Making IP = PSPACE practical: Efficient interactive protocols for BDD algorithms," in *Computer Aided Verification*, C. Enea and A. Lal, Eds. Cham: Springer Nature Switzerland, 2023, pp. 437–458.

Budget-Based Explainable Schedulability Analysis for Automotive Applications

Muhammad Tanveer Ali Ahmad
Jesus Pestana
Pro2Future GmbH
Graz, Austria
tanveer.ali-ahmad@pro2future.at
jesus.pestana@pro2future.at

Leandro Batista Ribeiro
Marcel Baunach
Graz University of Technology
Graz, Austria
lbatistaribeiro@tugraz.at
baunach@tugraz.at

***Abstract*—Schedulability analysis is commonly performed considering that all instances of a task (jobs) run with a fixed execution time, i.e., the WCET. This strategy can be overly pessimistic because not all jobs need the entire WCET to finish. In the AUTOSAR context, the so-called sequencer tasks consist of runnables, which are elementary execution units of an application. Each job of a sequencer task might execute a different number of runnables. We take advantage of this characteristic to perform a less pessimistic schedulability analysis by considering the task dynamic execution time, i.e., different WCET for different jobs. A job WCET value is directly obtained from the set of runnables it executes. Our approach focuses on the Fixed-Priority Preemptive Scheduling (FPPS) policy, and we demonstrate the schedulability analysis on an AUTOSAR-based application model. We propose a budget-based explainable schedulability analysis that precisely identifies the task, the job, the point in time when a schedulability violation occurs, and the additional budget requirement. For a schedulable system, our results are beneficial for software extensions, e.g., the remaining budget can be used to add a runnable or task without reanalyzing the schedulability of the entire system.**

***Index Terms*—Explainable Schedulability Analysis, Runnable to Task Mapping, Classic AUTOSAR.**

I. Introduction

Modern embedded automotive applications are designed and implemented according to the AUTomotive Open System ARchitecture (AUTOSAR) standards [2]. Each AUTOSAR application is decomposed into Software Components (SWCs) [3]. The internal behavior of each SWC is realized using a set of runnables (R). Runnables are the elementary units of the application and implement specific functions. Runnables offsets are calculated to minimize the response time of the embedded system [5], [6], [14], [8], [13].

In practice, there are numerous runnables in an embedded application. Therefore, for better utilization of hardware resources [10], [5], runnables are grouped into Sequencer Tasks (τ), using mapping techniques mentioned in [10], [7], [15]. A sequencer task (τ_m), is scheduled by the Operating System (OS), and it is used to sequentially execute the mapped runnables in a fixed order according to their periods and offsets [5]. In this work, we consider only the basic sequencer task, i.e., without the waiting state [4]. Each activation of a task is called a Job (J). From now on, we use *runnables* to refer to mapped runnables in a sequencer task and *tasks* to refer to sequencer tasks.

This work has been supported by the FFG under contract No. 881844: "Pro²Future" (Products and Production Systems of the Future), Graz University of Technology (TU Graz), and Elektrobit Automotive GmbH in the joint research project CompEAS.

Application Model								
Runnable Model				Sequencer Task Model				
ID	Period in ms	Offset in ms	WCET in us	ID	Period in ms	Offset in ms	Priority	WCET in us
R_1	6	0	1000	τ_1	2	0	2	2000
R_2	6	2	1000					
R_3	8	4	1500	τ_2	4	0	1	3000
R_4	8	0	1500					

Fig. 1. Application model, including mapping of runnables to sequencer tasks.

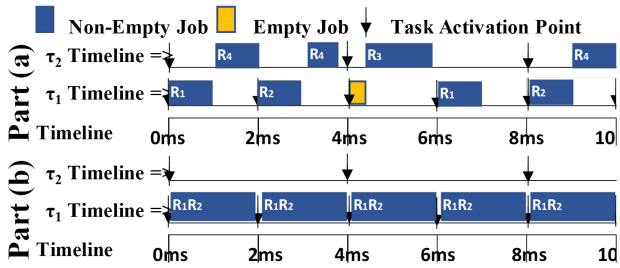


Fig. 2. System schedulability analysis for the AM in Fig. 1: (a) our approach with different WCETs for different jobs, (b) jobs with the same WCETs.

An example of our Application Model (AM), which is a simplified version of the automotive AM, is shown in Fig. 1 (more details in section II). In our AM, the runnables are specified by their period and worst-case execution time (WCET). The other parameters of our AM can be either specified manually or calculated automatically as follows. For the runnables, the offsets are derived using the Least Loaded (LL) heuristic [13], which minimizes response times, and runnables are mapped to tasks using the Multiple Periodic Solution (MPS) technique [10]. For the tasks, the period is calculated from the periods and offsets of the runnables using Equ. 1 [5], and the priority is derived by Rate-Monotonic Scheduling (RMS). Our approach does not use the task WCET. In related works, the task WCET is the sum of the WCETs of its runnables [10].

Related Work and Paper Contributions. Previous work

on real-time OSs with FPPS [12], [11], [9], perform schedulability analysis considering that all instances of a task (jobs) run with a fixed execution time, i.e., the WCET. In contrast, our approach considers that a job can execute a different number of runnables, which results in different WCETs for different jobs of the same task. Considering the AM specified in Fig. 1, the pessimistic usage of jobs with the same WCETs [12], [11], [9] determines the AM as un-schedulable, because the higher priority task τ_1 would run permanently and never gives τ_2 a chance to run (Fig. 2(b)). Our job-specific WCET allows less pessimistic schedulability analysis and it shows the AM is schedulable (Fig. 2(a)).

In [1], the authors considered the dynamic execution time of jobs, primarily due to dependencies among tasks. In our AM, we do not consider task dependencies but we still have dynamic execution time. In our model, this is due to allowing varying numbers of executing runnables inside jobs of the same task, see job description in section II.

In addition, we consider the clock frequency, which is not directly considered in the above-cited works. This is important because un-schedulable systems may become schedulable by increasing the clock frequency.

In this paper, we introduce a novel budget-based algorithm for schedulability analysis, which considers: (i) task dynamic execution time, (ii) OS context switch costs, and (iii) the clock frequency of the embedded system. For our AM, the runnables to task mapping are provided by the system designer, and we do not propose any (re)mapping technique. Our approach provides a schedulability analysis of all jobs in the AM, and enhances explainability through our novel budget list concept as follows: First, for un-schedulable systems, we identify the task, the job, the point in time when a schedulability violation occurs, and the additional budget requirement. Second, for schedulable systems, we provide the remaining budgets at the end of the analysis which, e.g., can be used for adding a new task without reanalyzing schedulability (see Example 1 in Section IV).

II. Application Model

Here, we define the variables of our AUTOSAR-based AM.

Runnable. Let R be the set of all periodic runnables. Each runnable can be represented as $R_i = (C_i, T_i, \phi_i)$ where C_i is the WCET, T_i is the period, and ϕ_i is the offset, which specifies the release time delay of its first job. For all runnables, $0 \leq \phi_i < T_i$. Each runnable can be mapped to only one sequencer task.

Sequencer Task. A task can be represented as $\tau_m = (W_m, P_m, O_m, TR_m)$, where W_m is the execution time, P_m is the period, O_m is the offset, and TR_m is the subset of runnables mapped to τ_m. For deterministic and precise time triggering of mapped runnables, P_m must consider the release time and period of every runnable [5]. If $TR_m = R_1, R_2, \ldots, R_k$, where k is the number of runnables, then:

$$P_m = GCD(T_1, \ldots, T_k, \phi_1, \ldots, \phi_k) \quad (1)$$

Job. A single activation of a task is a job. For different jobs of the same task, the number of executing runnables can vary from 0 to k, where k is the number of mapped runnables, which enables task-level dynamic execution time. The term Empty Job (EJ) refers to a job in which no runnable executes ($k = 0$). Every task τ_m releases a job at each time instant $(k \cdot P_m + O_m)$, which has a deadline at $((k+1) \cdot P_m + O_m)$, for $k \in \mathbb{N}$. Alg. 1 [5], shows the implementation of tasks. Each task uses a counter, α_m, initialized to zero, to identify which runnables to execute. The pattern of execution of mapped runnables on a task τ_m repeats after N_m jobs, where $N_m = LCM(T_1, T_2, \ldots T_k)/P_m$.

Algorithm 1 Counter-Based Runnables Activation Handling

```
1: procedure SequencerTask()
2:     for R_k ∈ τ_m do
3:         if ((α_m mod T_k/P_m) = φ_k/P_m) then
4:             Call R_k
5:     α_m = (α_m + 1) mod N_m
```

Assumptions. (i) Each task or runnable is periodic, (ii) if the clock frequency of the embedded board scales, the number of ticks for every instruction remains constant.

III. Proposed Approach

The flow chart of the proposed budget-list-based algorithm for schedulability analysis is shown in Fig. 3. The following paragraphs explain each step in detail:

Inputs. The following three input models are considered. (i) The application model (AM), as shown in Fig. 1, where the execution time of runnables are in ticks. (ii) The embedded platform model which represents a set of tuples of core ID E_k, and core frequency F_k. (iii) OS model, that represents a constant task context switch cost in ticks, H.

Initialization. There are four initialization steps: **(i) Length of Analysis.** For the periodic system the pattern of execution of all runnables in AM starts repeating after a major cycle, M, which is the LCM of the periods of runnables. Therefore, schedulability analysis for the time window [0, M] is considered in our algorithm. **(ii) Length of The Budget List.** In the proposed algorithm, the budget list covers the schedulability analysis over one major cycle, M. To calculate the number of slots L, we use step-1 of Fig. 3. Afterward, we create the budget list with L number of slots. **(iii) Budget of Each Slot.** Jobs are scheduled in slots as per their task period. Each slot budget value, B, represents a time length which represents the minimum time elapsed between any two job activations in AM. Therefore it is GCD of runnables periods and offset (Fig. 3, step-2). Moreover, we initialize each slot of the budget list with this B value in ticks. To convert a variable unit from time to ticks, its value is multiplied by the board clock frequency. For example, a board with 5MHz clock frequency has 5ticks=1μs. Therefore, if B is 3ms, its value in ticks is $B = 3 \cdot 1000\mu s = 3 \cdot 1000 \cdot 5ticks = 15000ticks$. In summary, each slot of the budget list represents (a) a

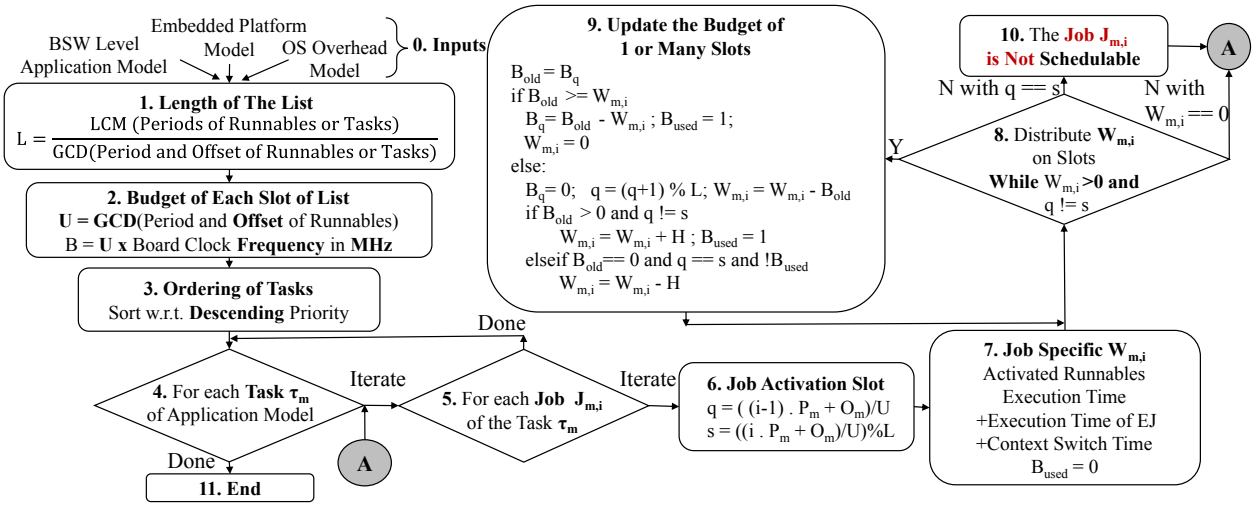


Fig. 3. Flow chart of the proposed algorithm. Using the budget list we analyze the schedulability of every job of every task.

time unit, e.g., 3ms, and (b) a tick budget, e.g., $15000 ticks$. Ticks are the atomic computation unit, which enables us to discretely allocate the execution time of jobs to slots in the budget list and to incorporate the role of the clock frequency in the schedulability analysis. **(iv) Ordering of Tasks.** In FPPS, lower priority tasks never preempt higher priority ones. Therefore, in our algorithm, we start with the highest priority task (Fig. 3, step-3). Then, we assign its jobs to specific slots of the budget list and update the slots budgets. Once all jobs of the highest-priority task are assigned, we sequentially move to the jobs of lower-priority tasks. As an example, see the timelines in Fig. 2(a), this process is detailed in the next paragraph.

Simulation of Activation and Execution of Jobs. For schedulability analysis we use steps 4-10 in Fig. 3 to simulate the activation and execution of all jobs, allocate their execution times in the budget list, and determine whether they are schedulable. In step-4, we start with the highest priority task and sequentially move to lower priority tasks in the application model. In step-5, for each task τ_m, there are M/P_m jobs within the length of analysis M. In step-6, for each job $J_{m,i}$ of a task τ_m, we calculate the index, q, of the budget list where we have to place its i^{th} activation using the task's period and offset. We also calculate the index, s, of the list that shows the activation point of the subsequent job of the same task. For the job $J_{m,i}$, the budget list index range is [q, (s-1)]. In step-7, we calculate the i^{th} job-specific execution time, $W_{m,i}$, using Equ. 2.

$$W_{m,i} = W_{m,r} + W_{m,e} + H \cdot d \tag{2}$$

All variables of Equ. 2 are in ticks. The OS task context switch time, H, is required every time a new slot is used for the same job. The number of context switches, d, is initialized to 1. $W_{m,e}$ is the execution time of the Empty Job (EJ), i.e., which is the execution time of Alg. 1 with line 4 excluded. $W_{m,r}$ is the sum of WCETs of the runnables that will be executed inside this i^{th} job, using Alg. 2. In step-8, the job-specific execution time, $W_{m,i}$, of a job, $J_{m,i}$, can be distributed among a number of consecutive slots of the list, starting from index q till (s-1) based on the available budget in each slot. In step-9, we allocate all or a portion of the remaining $W_{m,i}$ to the current slot q, with available budget B_q. If the job, $J_{m,i}$, has execution time remaining, $W_{m,i} > 0$, then we go to the next consecutive slot, and $W_{m,i}$ is increased by H if B_q was > 0. The variables are updated as shown in Fig. 3, step-9.

Algorithm 2 Job Specific Execution Time Calculation Based on Activated Mapped Runnables and Slot Index of Budget List

Input q, TR_m, P_m
Output $W_{m,r}$

1: **procedure** DYNAMICEXECUTIONTIMECALCULATOR()
2: $W_{m,r} = 0$
3: **for all** $R_k \in \{TR_m\}$ **do**
4: **if** $(q \bmod \frac{T_k}{P_m}) = \frac{\phi_k}{P_m}$ **then**
5: $W_{m,r} = W_{m,r} + C_k$

Schedulability Condition. If we are able to allocate the job-specific execution time, $W_{m,i}$, in the consecutive slots with index ranges [q, (s-1)] of the budget list, then the job $J_{m,i}$ is schedulable, otherwise it is not (step-10).

Once we have allocated the current job-specific execution time, $W_{m,i}$, and determined its status of schedulability, we sequentially repeat the same procedure for all jobs of the same task, τ_m. Moreover, we sequentially move from higher priority tasks to lower priority tasks, and in a similar manner perform the schedulability analysis of every job of all tasks. Finally, if any single job is not schedulable, then the system is not schedulable. Through our sequential schedulability analyses of every job, we can precisely identify every job for which a schedulability violation occurs. For each of these un-schedulable jobs, we identify the task, the job, the point in time when the schedulability violation occurs, and the additional budget requirement. The time complexity of Alg. 1 and Alg. 2 is $\mathcal{O}(k)$, where k is the number of mapped runnables. For the algorithm in Fig. 3, it is $\mathcal{O}(N \cdot L \cdot k)$, where N is the number

of tasks in the AM and L is the length of the budget list.

IV. Demonstration

Now, we use our approach (Sec. III) to determine whether the AM shown in Fig. 1 is schedulable, and we are interested in the result along with the provided detailed explanation. For the embedded platform model, we have a single core system with clock frequency, $F_1 = 1MHz$. For OS overhead modeling, we have context switch cost in ticks, $H = 10ticks$. The EJ execution time is $W_{1,e} = W_{2,e} = 6ticks$.

Explainable Schedulability Analysis – Example 1. When we execute our algorithm, it simulates the timeline for one major cycle of application, i.e., $L = 24ms$. Our algorithm generates the following trace Fig. 4 of schedulability analysis for job-level details output to explain the details of each step. In short, the algorithm proceeds as follows. For each task, τ_m, in descending order of priority, it simulates the activation of all jobs. For each job, $J_{m,i}$, based on the task period, offset and job count, i, it calculates the start and end index of the budget list, i.e., $[q, s-1]$. Afterward, it calculates the job-specific execution time based on the index, q, and mapped runnable periods and offsets using Alg. 2. Then, it allocates this execution time to a set of consecutive slots of the budget list that resides within indexes $[q, s-1]$. If the allocation of the execution time, $W_{m,i}$, is successful, then this job, $J_{m,i}$, is schedulable, otherwise it is not schedulable.

For the given application, OS overhead, and platform model, our algorithm shows that the system is schedulable (Fig. 4).

Furthermore, the last state of the budget list informs how much margin remains in each $2ms$ time slot. This information is useful as follows. If we want to introduce a new periodic or sporadic task, we can analyze what should be its period/inter-arrival time and its maximum worst-case execution time. For example, if we add a task of period $6ms$ and offset $0ms$, then it activates at every 3^{rd} slot, and its maximum available budget per activation $W_{m,i}$ is the minimum sum of each 3 consecutive slots of the last state of the budget in Fig. 4, i.e. [910 2426 910 2426]. Hence, for the system to remain schedulable, this task with period 6ms needs to have $W_{m,i} \leq 910ticks$ (Equ. 2).

```
****************************** Start of Analysis ******************************
Length of Schedulability Analysis ==> 24 ms
Duration of Each Slot => 2 ms  Slot Budget => 2000 Ticks
Current Status of Budget List: [2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000]
Task Context Switch Cost = H => 10 Ticks
---Analysis of Task ID= 1 , Priority= 10 , Period=  2 ms, and Offset= 0 ms
EJ Execution Time = Wm,e =>  6 Ticks
Schedulability Analysis of Job No.  1
Runnable of Name R1 , Period 6 ms, Offset 0 ms, and  WCET 1000 ticks will be executed.
Distribution of job execution time:
      Allocated  1016  Ticks from List Index  0 .
Job  1 Final Execution Time = Wm,i = Wm,r + Wm,e + H . d => 1016 Ticks
Updated Budget List =>  [ 984 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000]
**Schedulable Job  1 of Task 1 . **
...
****************************** End of Analysis ******************************
Last State of Budget List => [   0  442  468  984   0  1442    0  442  468  984  0  1442]
***System is Schedulable***
```

Fig. 4. Details of our algorithm-generated analysis for a schedulable system.

Explainable Schedulability Analysis – Example 2. To demonstrate the working of our algorithm in the case when the system is not schedulable, we modify the AM in Fig. 1 by gradually changing the WCET of runnable R_3 and repeating the schedulability analysis until the system is determined as un-schedulable. This occurs, at the WCET value of $1943ticks$. The resulting trace along with the provided detailed explanation are shown in Fig. 5.

In this figure, the 4^{th} job of τ_2 has missed its $4ms$ deadline by $1tick$. This job's execution time $W_{m,i}$ is $1969ticks$. Since each slot represents a ticks budget for $2ms$ and τ_2 has a period of $4ms$, the execution time of the 4^{th} job of τ_2 has to be allocated on the list indexes $[6, 7]$. Before allocating this job, the latest state of the budget for these list indexes is [984 984]. If we consume all available budgets, we still cannot meet the $1969ticks$ execution time demand. Furthermore, we cannot take budget from the 8^{th} index of the list, because the $4ms$ deadline (period of τ_2) does not permit it. We also cannot take budget from the 5^{th} slot either, because the 4^{th} job of τ_2 is not activated at that time. Therefore, this job is un-schedulable. Hence, the whole system is un-schedulable.

```
******************** Start of Analysis********************
Length of Schedulability Analysis ==> 24 ms
...
Updated Budget List =>  [ 0  442   25  984  0  1442  984  984 1984  984  984 1984]
**Schedulable Job  3 of Task 2 . **
Schedulability Analysis of Job No.  4
Runnable of Name R3 , Period 8 ms, Offset 4 ms, and  WCET 1943 ticks will be executed.
Distribution of job execution time:
      Allocated  984  Ticks at List Index  6 .
      Allocated  984  Ticks at List Index  7 .
Job  4 Final Execution Time = Wm,i= Wm,r + Wm,e + S x d => 1969 Ticks
Updated Budget List =>  [ 0  442   25  984    0 1442  0  0  1984  984  984 1984]
!!Un-Schedulable Job  4  of Task 2 by  1 tick(s). !!
...
******************** End of Analysis********************
Last State of Budget List => [  0  442  25  984  0  1442  0    0  468  984   0  999]
!!!System is Un-Schedulable!!!
Because of [TaskID, Job ID, Lack Of Tick] ==> [[2, 4, 1]]
```

Fig. 5. Details of our budget-based algorithm-generated analysis. In this case, the algorithm output explains that job 4 of task 2 is unschedulable, because it requires 1 tick of additional budget at the list indexes 6 and 7 (or between the time of 12 ms and 16 ms). Therefore the system is un-schedulable.

Effect of Clock Frequency. If we want to make the previous system schedulable, one possible way is to increase the clock frequency, e.g., to $F_1 = 2MHz$. This change in frequency results in an increased budget for each slot from 2000 to 4000 ticks, and in a schedulable system.

V. Conclusion and Future Work

In this paper, we have presented our budget-based approach that focuses on the Fixed-Priority Preemptive Scheduling (FPPS) policy, and we have demonstrated the schedulability analysis on an AUTOSAR-based application model.

Our approach provides detailed insights, due to the budget list and through the sequential schedulability analyses of every job of every task. For un-schedulable systems, our approach explains which jobs have caused schedulability violations, along with related informations. For schedulable systems, the budget list is useful, e.g., to determine a bound on the WCET of a new task to be added.

Our schedulability analysis can be applied to application models that only contain the task model, i.e., with fixed task WCETs. Additionally, since our analysis is based on pessimistic assumptions, i.e., the execution time always equals the WCET of runnables, and we do not consider inter-task dependencies, the schedulability analysis result remains valid in case of reduced execution times. While this paper considers only periodic tasks, other task models and scheduling policies will be addressed in future work.

REFERENCES

[1] Björn Andersson, Hyoseung Kim, Dionisio De Niz, Mark Klein, Ragunathan Rajkumar, and John Lehoczky. Schedulability analysis of tasks with corunner-dependent execution times. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(3):1–29, 2018.

[2] AUTOSAR. AUTOSAR Layered Software Architecture, R22-11 , 2022.

[3] AUTOSAR. AUTOSAR Software Component Template, R22-11 , 2022.

[4] AUTOSAR. AUTOSAR specification of operating system, r22-11, 2022.

[5] AUTOSAR. AUTOSAR specification of RTE software, r22-11, 2022.

[6] Daeho Choi, Tae-Wook Kim, and Jong-Chan Kim. AUTOSAR runnable periods optimization for dag-based complex automobile applications. *Applied Sciences*, 10(17), 2020.

[7] Klaus et al. Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[8] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost. In *4th European Congress on Embedded Real Time Software (ERTS)*, 2008.

[9] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[10] Fouad Khenfri, Khaled Chaaban, and Maryline Chetto. Efficient mapping of runnables to tasks for embedded AUTOSAR applications. *Journal of Systems Architecture*, 110:101800, 2020.

[11] Joseph Y-T Leung, Merrill, and ML. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

[12] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[13] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multisource software on multicore automotive ecus—combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, 2012.

[14] Shiva Nejati, Morayo Adedjouma, Lionel C Briand, Jonathan Hellebaut, Julien Begey, and Yves Clement. Minimizing CPU time shortage risks in integrated embedded software. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013.

[15] Miloš Panić, Sebastian Kehr, Eduardo Quiñones, Bert Boddecker, Jaume Abella, and Francisco J Cazorla. Runpar: an allocation algorithm for automotive applications exploiting runnable parallelism in multicores. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, New York, NY, USA, 2014. Association for Computing Machinery.

Transforming Logic into Language: Bridging the Gap with Large Language Models

Ruben Martins
Carnegie Mellon University
Pittsburgh, USA
rubenm@andrew.cmu.edu

***Abstract*—Formal verification is used in safety-critical systems to prevent catastrophic errors. In such domains, certification is paramount, demanding rigorous verification processes to obtain the elusive seal of approval. Despite advancements in verification tools, analysts remain apprehensive when a tool declares a system "verified" without offering a meaningful human explanation.**

This vision paper addresses this critical challenge by introducing a novel idea to enhance the explainability of formal verification tools. Our method leverages the power of large language models (LLMs) to transform intricate logical formulations into comprehensible natural language explanations. By bridging the chasm between logic and language, our proposed framework will enable the generation of intelligible explanations that analysts can readily consume. This, in turn, will engender a heightened confidence in the correctness of safety-critical systems, fostering a symbiotic relationship between human expertise and computational rigor in the pursuit of certifiable trustworthiness.

***Index Terms*—formal methods, explainability, LLMs**

I. Introduction

Safety-critical systems [1], exemplified by railway control systems, aviation systems, and electric power grid systems, play an indispensable role in modern society. They rely heavily on intricate software components to ensure their reliable operation. Ensuring the software's correctness is crucial for the systems' functioning. The use of formal methods has become a strategy to ensure confidence in their performance, allowing the software to be rigorously proven to have defined properties.

Formal verification tools have enhanced trust in safety-critical systems, but a significant challenge remains: the absence of explanations accompanying the assertion of program correctness. In essence, formal verification tools, though adept at establishing the veracity of a program, often fall short in explaining why a program is correct. In contexts such as aviation, where stringent certification guidance documents like DO-178C [2], DO-254 [3] and DO-330 [4] describe human analyst involvement, a pertinent question arises: *"Why should I place trust in the verification tool when it merely asserts correctness without furnishing compelling evidence?"* [5]

Recognizing the critical need for transparency in the formal verification process, prior work [5] has emphasized that formal verification tools should not confine their focus solely to correctness and scalability. Instead, they should figure out why a program is flawed or sound.

In response to this imperative, this vision paper endeavors to tackle the challenge head-on by concentrating on enhancing explainability within formal methods approaches. We propose to *transform complex logical formulations into readily understandable natural language constructs facilitated by the deployment of large language models* [6]. By bridging the gap between intricate logic and human-friendly language, we empower analysts to scrutinize and comprehend the underlying logic, thereby nurturing a heightened level of trust in the overall system. This endeavor represents a significant step towards a more symbiotic relationship between formal verification tools and human analysts in pursuing safety-critical system reliability and certification.

This vision paper makes the first step towards using large language models to improve the explainability of verification tools. Although the ideas presented in this paper are still in a very preliminary stage, we hope to encourage researchers in the real-time systems community and in the formal methods community to pursue these ideas further and improve the explainability of formal methods tools. This paper is organized as follows. First, we show an example of a logical formula and how large language models can be used to explain its meaning. Next, we present the overall framework for transforming logic into language. Finally, we show potential applications of this framework such as to certification of safety-critical systems.

II. Motivation

Consider a simple real-time system problem of task partitioning [7], where we have n processors and k threads and want to partition the threads by each processor such that (1) one thread must be allocated to one processor, and (2) each processor can have at most one thread.

If we query ChatGPT (version 4) about this problem and ask it to explain why we cannot assign 5 threads to 4 processors, ChatGPT clearly explains why the problem has no solution as shown in Figure 2. [1] The answer given by ChatGPT is great and clear! However, can ChatGPT do a good job when logically formulating this problem? If we encode the above problem into SAT [8] then we will have the representation depicted in Figure 1.

Each number corresponds to a variable, and each line corresponds to a disjunction of facts. For instance, "1 2 3 4 5 0" corresponds to having one of the five threads allocated to the first processor. By mapping the meaning of each variable

[1] https://chat.openai.com/share/a0fa84de-1a5b-48c4-b21c-350fd6eab2c3

```
p cnf 20 49
1 2 3 4 5 0       c thread#1 is in processor#1 OR thread#2 is in processor#1 OR ...
6 7 8 9 10 0      c thread#1 is in processor#2 OR thread#2 is in processor#2 OR ...
11 12 13 14 15 0 c thread#1 is in processor#3 OR thread#2 is in processor#3 OR ...
16 17 18 19 20 0 c thread#1 is in processor#4 OR thread#2 is in processor#4 OR ...
1 6 11 16 0       c thread#1 is in processor#1 OR thread#1 is in processor#2 OR ...
2 7 12 17 0       c thread#2 is in processor#1 OR thread#2 is in processor#2 OR ...
3 8 13 18 0       c thread#3 is in processor#1 OR thread#3 is in processor#2 OR ...
4 9 14 19 0       c thread#4 is in processor#1 OR thread#4 is in processor#2 OR ...
5 10 15 20 0      c thread#5 is in processor#1 OR thread#5 is in processor#2 OR ...
-1 -2 0           c thread#1 is not in processor#1 OR thread#2 is not in processor#1
-1 -3 0           c thread#1 is not in processor#1 OR thread#3 is not in processor#1
-1 -4 0           c thread#1 is not in processor#1 OR thread#4 is not in processor#1
-1 -5 0           c thread#1 is not in processor#1 OR thread#5 is not in processor#1
...
```

Fig. 1. SAT formula for assigning 5 threads to 4 processors

5 Threads (Tasks): T1, T2, T3, T4, T5
4 Processors (Workers): P1, P2, P3, P4

If you assign one thread to each processor, it might look like this: P1: T1, P2: T2, P3: T3, P4: T4

Now, you have successfully assigned 4 threads to 4 processors. However, you are left with one thread (T5) without a processor to assign it to, because each processor is already busy handling one thread, and they cannot handle more than one thread at a time.

...

You would need at least 5 processors to assign 5 threads following these rules.

Fig. 2. Answer from ChatGPT to the English description of the problem

to natural language, we can have a language representation of the formula as shown in the comments in each line of Figure 1.

Suppose we feed just the formula to ChatGPT. In that case, since the logical encoding does not have information about what represents threads and processors, ChatGPT answer will be incorrect and it will claim that it does not find any clear contradiction in the formula.[2] However, if we instead provide the natural language representation and ask for a text summarization of why there is no solution to this problem then ChatGPT provides a good explanation as shown in Figure 3.[3] While this serves as a straightforward illustration, it underscores the potential of Large Language Models (LLMs) in the realm of text summarization for logical formulas. It also highlights their capacity to elucidate the reasons behind the unsatisfiability of a formula, a capability that holds significant promise for enhancing the comprehensibility of formal verification tools.

III. Transforming Logic into Language

The traditional workflow of a verification tool, as depicted in Figure 4, involves taking a specification and a system model as input and returning one of two outcomes: "verified" or "failed." While this binary outcome provides a decisive verdict on the system's correctness, it often lacks the granularity

[2]https://chat.openai.com/share/78f6cc37-ea0f-41f7-92ee-d50284049de0

[3]https://chat.openai.com/share/37807b38-a3af-4faf-acbb-94e319c0a30a

Here's a summarized explanation of why there is no solution:

1) **Existence on a Processor:** Each thread must be on one of the processors (as per the constraints like "thread#1 is in processor#1 OR thread#1 is in processor#2 OR thread#1 is in processor#3 OR thread#1 is in processor#4").
2) **Mutual Exclusivity:** No two threads can solely occupy the same processor. For each pair of threads, there's a constraint ensuring that at least one of them is not present on each processor (e.g., "thread#1 is not in processor#1 OR thread#2 is not in processor#1").
3) **General Distribution:** Every processor must have at least one of the threads (e.g., "thread#1 is in processor#1 OR thread#2 is in processor#1 OR thread#3 is in processor#1 OR thread#4 is in processor#1 OR thread#5 is in processor#1").

Considering these constraints, there's a contradiction that makes finding a solution impossible. For each processor, while there must be a thread assigned, the mutual exclusivity constraints make it so that it's impossible for all threads to find a unique processor without violating any conditions.

Fig. 3. Answer from ChatGPT to the natural language representation of the logical constraints

necessary for in-depth understanding. In response to this challenge, we propose an extended pipeline that augments this conventional approach with a layer of intelligibility.

Our proposal hinges on the extraction of a logical formulation generated by the formal verification tool when it returns a "verified" result. This logical formulation encapsulates the rationale behind the verification outcome and serves as the gateway for unlocking the black box. Subsequently, we leverage the capabilities of LLMs to transform this logical formulation into natural language. The resulting natural language explanation can then be readily interpreted by an analyst, shedding light on the reason the system is deemed "verified."

However, our approach presents several significant challenges that require careful consideration.

1) **Transforming Logic to Language:** The initial challenge is to transform complex logical formulas into clear and human-readable natural language sentences. We can begin by associat-

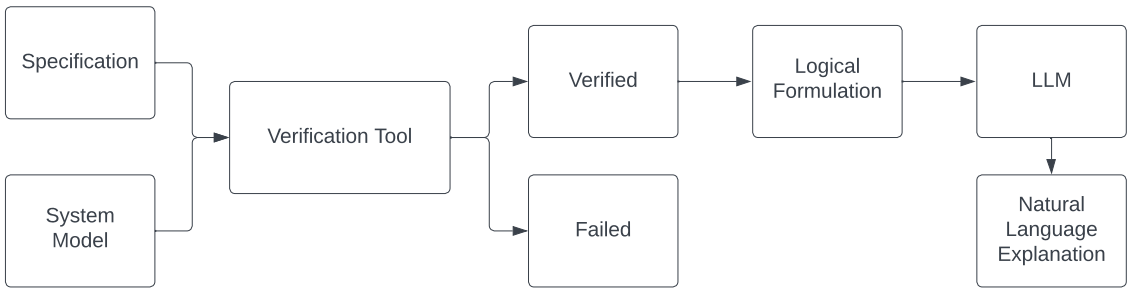


Fig. 4. Transforming Logic into Language using LLMs

ing logical variables with their semantic interpretations. Next, we create statements for individual clauses in the formula by combining the meanings of these variables through disjunctions, as shown in the motivation section. While this method offers basic explanations, it might not fully generate English-like sentences that are both informative and easily understood.
2) **Guidance for LLMs:** The choice of prompt provided alongside the natural language description significantly influences the comprehensibility of the output generated by the Language Model (LLM). For example, requesting a text summarization yields considerably superior results compared to relying on the LLM for logical reasoning tasks. This is because LLMs excel in text summarization tasks, showcasing their strength in this domain compared to logical reasoning.
3) **Confidence in the Explanation:** LLM explanations may not always be accurate. To enhance confidence in LLMs explanation, one can query multiple times, seeking consistency in responses. LLMs are also more likely to explain well if they use known concepts. For instance, the task partitioning problem in Section II can be seen as a variant of the pigeonhole problem [9]. When interacting with ChatGPT, some answers apply the pigeonhole principle to our motivating problem.[4] Leveraging established arguments relevant to safety-critical systems in conjunction with the chain-of-thought method [10] can make explanations from LLM more reliable and insightful.
4) **Selecting the Appropriate Logical Formula:** The fourth challenge pertains to selecting the most suitable logical formula for explanation. One approach involves focusing on the original formula, as demonstrated in the motivating example. However, alternative strategies warrant exploration. For instance, considering an unsatisfiable subformula, smaller than the original might offer a more accessible starting point for explanation. Another option is to harness the proofs emitted by constraint solvers and task large LLMs by elucidating these logical formulations instead of the original formula.

These challenges underscore the multidimensionality of our approach. By addressing how to transform logic into natural language effectively and selecting the most appropriate logical formulation for explanation, we aim to create a symbiotic relationship between formal verification tools and LLMs. The ultimate goal is to empower analysts with clear, interpretable explanations, enhancing their capacity to discern genuine verifiability from spurious claims and fostering confidence in the output of verification tools.

[4]https://chat.openai.com/share/3ba78d98-d428-416b-afde-3fb194e487ad

IV. Applications

Enhancing Certification of Safety-Critical Systems with Natural Language Explanations. The certification of safety-critical systems [1], particularly in domains like aviation, railway control, and power grid management, demands a meticulous and reliable process to ensure that these systems adhere to stringent guidance documents and specifications. Formal verification tools have played a pivotal role in this context by rigorously assessing the properties of software components. However, a glaring challenge persists regarding the interpretability of the verification results.

In formal verification, it is not uncommon for a verification tool to confidently assert that a system is "verified" without providing any human-readable output that an analyst can comprehend. This lack of transparency is concerning for several reasons. Firstly, it opens the possibility that the verification tool itself might have a bug [11]–[13], leading to erroneous claims of system correctness. Secondly, and perhaps more commonly, it raises the specter of logical specification errors. In such cases, the system may indeed be verified, but concerning a specification that contains a subtle logical error or an unintended interpretation [14]. In both scenarios, the absence of clear, human-understandable explanations poses a significant challenge for trusting the output of the tool.

The transformative potential of translating complex logic into natural language empowers analysts. It simplifies the scrutiny and comprehension of underlying logic and verification processes, facilitating error detection, whether tool or specification-related, and enhancing the certification process.

Consider a scenario where a logical specification error exists, leading to an incorrect verification result. If the verification tool can provide a natural language explanation of the logic it assessed, the analyst is better equipped to spot the erroneous specification. Similarly, if the explanation aligns with the analyst's understanding of the system's behavior, it

fosters confidence in the tool's output. Transforming logic into natural language improves the certification process, enabling analysts to engage actively, identify issues, and ensure the safety and reliability of critical systems.

Leveraging Large Language Models for Debugging Logical Formulas. Creating and refining logical formulas is a fundamental aspect of formal methods and, by extension, the verification of critical systems. However, it's an endeavor fraught with complexity that often tests the limits of human precision. Crafting logical formulas is notorious for its inherent difficulty and proneness to errors. Even the most experienced logicians can inadvertently introduce subtle flaws into their formulations. In the context of formal verification, the output of a verification tool holds profound significance. When such a tool declares that a system is "verified," it essentially asserts that the logical formula representing the system's behavior is unsatisfiable. In other words, the tool did not encounter a counterexample, which would indicate a violation of the system's specification. However, this binary outcome—verified or not—merely scratches the surface of the underlying complexity. There are various potential reasons why a formula might be unsatisfiable. While the absence of a counterexample is a positive sign, it does not unequivocally validate the correctness of the system's behavior. It could be that the formula is unsatisfiable for reasons unrelated to the system's actual behavior. For instance, the encoding of the specification itself may contain a logical error, rendering the entire verification process invalid. Debugging logical formulations in the presence of such intricate variables is a formidable challenge [15]. It is often a time-consuming and labor-intensive endeavor that demands exceptional expertise.

However, the integration of large language models into this process introduces a transformative dimension. By transforming complex logical formulas into natural language descriptions, large language models can serve as invaluable aids in the debugging process. These models possess the capacity to concisely summarize the essence of a logical formula in a human-understandable manner. In doing so, they enable analysts to scrutinize the logic, identify potential discrepancies, and pinpoint the root causes of any unsatisfiability. This shift from abstract logic to accessible language offers a critical advantage, as it significantly reduces the cognitive gap between the analyst and the system's formal representation.

Enhancing Constraint Solver Performance through Lemma Summarization. Constraint solvers [8], [16], integral components of formal verification tools, are pivotal in exploring solution spaces for complex logical problems. These solvers actively acquire knowledge through lemmas during the search process. These lemmas serve as critical insights, enabling the solver to prune the search space efficiently and converge toward a solution. However, a fundamental challenge arises when managing the vast and dynamic corpus of lemmas generated during the search. To maintain reasonable memory usage, constraint solvers must make selective decisions regarding which lemmas to retain and which to discard. Traditionally, heuristic methods [17] have been employed to tackle this dilemma, prioritizing lemmas based on predefined criteria, such as recency or perceived importance. The advent of large language models has ushered in new opportunities for handling and summarizing complex data [18]. These models have demonstrated remarkable proficiency in data summarization tasks, distilling vast amounts of information into concise and informative representations. Given this capability, a compelling question emerges: *can we harness the power of large language models to summarize the lemmas learned by constraint solvers?* Moreover, can these summaries be leveraged to discern the relative importance of individual lemmas, thus guiding the decision-making process of lemma retention? This novel approach holds the potential to revolutionize the performance of constraint solvers. We can distill the essential insights gleaned during the search by employing large language models for lemma summarization, effectively preserving the most pertinent information while conserving memory resources. These summarized representations can then inform the development of novel heuristics, allowing constraint solvers to make more informed decisions regarding lemma retention. Enhanced heuristics, informed by summarization techniques, could lead to significant performance gains in constraint solving and improve verification of safety-critical systems and other complex logical domains.

V. Conclusions and Future Directions

In conclusion, the absence of explainability reduces the widespread adoption of formal methods in safety-critical systems [1] and real-time scheduling [19]–[21]. This vision paper has illuminated a promising path forward by advocating for *integrating large language models to bridge the gap between intricate logical formulations and human understanding*.

We envision many far-reaching applications by harnessing the power of language models to articulate complex logic in natural language. Even though our examples used verification using constraint solvers, we believe this approach is general and can be widely applied to other formal verification approaches. Although we focused on improving the explainability of verified systems, the same approach could also be used to improve the explainability of buggy systems. When a bug is found, most verification tools can return a counterexample that can be used to explain it. However, it may be the case that this counterexample is hard to analyze, and an explanation generated by LLMs could improve its interpretability.

In closing, we invite the formal methods and real-time systems communities to engage with this vision. Together, we can foster the development of explainable tools that empower analysts to certify the most intricate and critical systems confidently. As we continue on this trajectory, we anticipate a future where the marriage of formal methods and explainability leads to safer, more reliable, and more understandable critical systems, ushering in a new era of trust and dependability in complex software-intensive domains.

ACKNOWLEDGMENTS

The author would like to express gratitude for the valuable and insightful discussions with Björn Anderson concerning the explainability of formal methods approaches. In the process of crafting this article, ChatGPT (GPT-3.5 and GPT-4), OpenAI's advanced language-generation model, was employed to assist in generating and refining the writing style. The author meticulously reviewed, edited, and tailored the text generated by ChatGPT to align with personal preferences and standards, and thus assumes full responsibility for the content presented in this publication. This work was partially supported by CMU-Portugal project ANI 045917 funded by FEDER and FCT. All statements are those of the author, and do not necessarily reflect the views of any funding agency.

REFERENCES

[1] J. Bowen and V. Stavridou, "Safety-critical systems, formal methods and standards," *Software engineering journal*, vol. 8, no. 4, pp. 189–209, 1993.

[2] L. Rierson, *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance*. CRC Press, 2017.

[3] V. Hilderman and T. Baghi, *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*. Avionics Comm., 2007.

[4] J. Marques and A. M. da Cunha, "Cots tool qualification using rtca do-330: Common pitfalls," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–6.

[5] R. Martins, "Towards explainable formal verification," *ERSA*, 2022.

[6] OpenAI, "Gpt-4 technical report," 2023.

[7] N. Fisher, J. H. Anderson, and S. K. Baruah, "Task partitioning upon memory-constrained multiprocessors," in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005), 17-19 August 2005, Hong Kong, China*. IEEE Computer Society, 2005, pp. 416–421. [Online]. Available: https://doi.org/10.1109/RTCSA.2005.97

[8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, vol. 336. [Online]. Available: https://doi.org/10.3233/FAIA336

[9] W. A. Trybulec, "Pigeon hole principle," *Journal of Formalized Mathematics*, vol. 2, no. 199, p. 0, 1990.

[10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[11] M. N. Mansur, M. Christakis, V. Wüstholz, and F. Zhang, "Detecting critical bugs in SMT solvers using blackbox mutational fuzzing," in *FSE*. ACM, 2020, pp. 701–712.

[12] D. Winterer, C. Zhang, and Z. Su, "On the unusual effectiveness of type-aware operator mutations for testing SMT solvers," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–25, 2020.

[13] J. Park, D. Winterer, C. Zhang, and Z. Su, "Generative type-aware mutation for testing SMT solvers," *Proceedings of the ACM on Programming Languages*, vol. 5, no. OOPSLA, pp. 1–19, 2021.

[14] D. Neider and R. Roy, "Expanding the horizon of linear temporal logic inference for explainability," in *REW*. IEEE, 2022, pp. 103–107.

[15] M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, and S. Woltran, "Debugging asp programs by means of asp," in *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 2007, pp. 31–43.

[16] 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.

[17] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning SAT solvers," in *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2021, vol. 336, pp. 133–182. [Online]. Available: https://doi.org/10.3233/FAIA200987

[18] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, and T. B. Hashimoto, "Benchmarking large language models for news summarization," *arXiv preprint arXiv:2301.13848*, 2023.

[19] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations research*, vol. 26, no. 1, pp. 127–140, 1978.

[20] L. Sha, T. Abdelzaher, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok *et al.*, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2, pp. 101–155, 2004.

[21] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, pp. 1–44, 2011.