Temporal Logics Meet Real-World Software Requirements: A Reality Check

Roman Bögli ✉
University of Bern
Bern, Switzerland

Atefeh Rohani
University of Bern
Bern, Switzerland

Thomas Studer
University of Bern
Bern, Switzerland

Christos Tsigkanos
University of Athens
Athens, Greece

Timo Kehrer
University of Bern
Bern, Switzerland

***Abstract*—Reasoning on the behavior of software systems is challenging, especially in critical domains such as aerospace. Transitioning from natural language to formal specifications enables long-pursued activities such as modeling, synthesis, and verification. Temporal logics are often used in this regard, each with different operators, expressiveness or associated implementations. However, a significant gap exists between the theoretical capabilities of the logics applied in formal methods and the practical needs for specifying real-world requirements. This paper addresses this gap through a case study of SpaceWire, a standard specification for a data-handling communication protocol often adopted on spacecraft and other on-board systems. We extract 89 software requirements exhibiting temporal behavior and transcribe them into logic-based formalizations using different established temporal logics, maximizing natural encoding. We analyze the suitability of the chosen logics for formalizing the selected software requirements to reason about potential implications for both researchers and practitioners.**

***Index Terms*—requirements, specification, temporal logics**

I. Introduction

Formal methods offer a systematic approach to specifying, verifying, and reasoning about software behaviors by transitioning from informal, natural language requirements to formal, logic-based representations [1]–[5]. In particular, temporal logics [1], [2], [6], [7] have been widely adopted for software-intensive systems where time and order of operations are critical, such as embedded systems [8], distributed systems [9], and real-time systems [10]–[12]. Since Pnueli's seminal work on *Linear Temporal Logic* (LTL) [13], a plethora of temporal logics have been proposed [14]–[16], addressing various limitations and challenges associated with different types of systems, properties, and expressiveness required.

Despite the expanding body of scientific literature on the one hand, and numerous success stories highlighting the successful adoption of formal methods in industrial practice [17], [18] on the other hand, transitioning from natural language requirements to logic-based representations remains challenging. To compensate, recent research efforts have increasingly focused on providing guidance to practitioners and engineers in mastering such transition, aiming to make temporal logic-based reasoning more accessible and practical. Proposals range from auxiliary utilities such as user-friendly notations [19] or specification patterns [20], [21] to generating logic formulae [3]–[5], [22].

However, all of these approaches eventually target a specific logical framework serving as basis of a dedicated method or tool. Even user-friendly notations are designed to map intuitively to operators and constructs of some underlying logics, representing syntactic sugar on top of the respective logical framework. More generally, the vast majority of the literature can be classified as *solution-oriented*, falling short in providing practical, *problem-oriented* guidance for selecting the most suitable temporal logic for specific tasks or domains in the first place. Both standard textbooks and research papers usually work with toy examples for the sake of illustration, where an example is artificially constructed such that it suits to explain a specific solution. While such examples are carefully chosen for didactic reasons, practitioners usually face a situation just the other way round: They start with a problem and seek a suitable solution.

The absence of problem-oriented guidance means engineers may lack the expertise to evaluate which logic is best suited for their requirements, increasing the risk of suboptimal or erroneous specifications. Conversely, researchers need problem-oriented evidence to align the expressive power of temporal logics, including supporting tools, and the practical needs of engineers in different domains. Specification is regarded as a critical bottleneck, particularly for aerospace, and other autonomous systems [23].

In this paper, we tackle the challenge of transitioning from natural language requirements to logic-based representations by taking a problem-oriented view through the lens of SpaceWire [24], a standard specification for a data-handling communication protocol often adopted on spacecraft and other on-board aerospace systems [25].

In a first phase, we systematically extract a comprehensive set of functional software requirements from the SpaceWire standard, each of which exhibits temporal behavior that is critical to the correct functioning of the system.

In a second phase, we seek to identify a natural logic-based formalization for each of these text-based requirement. This phase is conducted by a logician, with the aim of collecting and classifying arguments that justify the selection without being biased by practical constraints such as tool support, reasoning complexity etc., and seeking the most natural formalization.

The third and last phase is dedicated to quantitatively analyzing these natural formalizations obtained from the previous phase. Guided by two research questions, we analyze the distribution of logics employed to transcribe the SpaceWire requirements, and explore potential translations between the selected logical frameworks. Additionally, we shift our view towards a more practical perspective by assessing what we term the *engineering complexity* of our specifications in terms of the syntactical complexity of the respective temporal logic formulae, aiming to identify cases where a formalization becomes overly complex or intricate, raising concerns about its practical utility in industrial contexts. To automate the assessment of the obtained requirement formalizations, we introduce `tlparser`, a tool tailored for harvesting statistical data from logical formulae.

Through this analysis, we aim to assess the suitability of each chosen logic for formalizing real-world requirements, highlighting both strengths and limitations of these logics in a practical context. Together with the presented methodology and `tlparser`, we enable replicable "fingerprinting" of requirement documents. By reflecting on the challenges encountered during the formalization process, we provide insights that can benefit both researchers and software engineers, ultimately contributing to the improvement of formal methods for practical use. Findings from this investigation serve as a reality check as they offer a clearer understanding of how formal methods utilizing temporal logics can be applied effectively in the context of aerospace and other critical domains.

In summary, we present the following five contributions:

1) We extract all SpaceWire requirements having a temporal notion and provide a natural formalization, including a transparent justification for the selection of specific logics to best fit each requirement.
2) We conduct a quantitative analysis on these formalized requirements that examines (1a) the distribution of natural formalizations across different temporal logics, (1b) their mutual translation potential, and (2) the engineering complexity of the resulting logical formulae.
3) We interpret key insights derived from this quantitative analysis and discuss their implications for both practitioners and researchers.
4) We introduce `tlparser`[1], a tool designed for automated reproduction of our quantitative analysis, ensuring reproducibility and facilitating further research.
5) We provide an accompanying artifact[1] that includes all data generated during the SpaceWire case study, allowing others to validate and build upon our work.

The remainder of this paper is structured as follows. Sec. II introduces the SpaceWire requirement document and states the extraction process of requirements with temporal behavior. In Sec. III, we formalize the extracted requirements in the most natural manner, providing justifications and reasons on selecting candidate logical frameworks. Sec. IV presents quantitative analysis results of the gained formalizations, covering the logics distribution, mutual translatability, and engineering complexity. Sec. V discusses the obtained results and their implications, followed by related work in Sec. VI. Finally, Sec. VII concludes the paper alongside future work endeavors.

[1] Tool support and data package at doi.org/10.5281/zenodo.14764480.

II. Requirements in SpaceWire

SpaceWire [24] is a standard specification for a data-handling network often adopted on spacecraft and other on-board systems. The protocol is standardized by the European Space Agency (ESA) in collaboration with other international space agencies (e.g., NASA, JAXA, and RKA) and is based on the IEEE 1355 standard [26]. It is a full-duplex, bidirectional, serial, point-to-point data link that employs differential signaling to ensure high reliability. It contains precise requirements to facilitate connections between various components such as scientific instruments, memory, processors, downlink telemetry, and other sub-systems located on-board a spacecraft. SpaceWire is widely used in space missions [27] for its high-speed data transfer capabilities and robustness. While the standard is heavily oriented towards hardware specifications, it also delineates essential software requirements to ensure seamless and reliable communication between on-board software sub-systems. These requirements encompass protocol implementation, device drivers, network management, error handling, timing, synchronization, and interoperability.

For this work, we are interested in requirements about *behavior* and thus focus on functional [28] requirements that match the following two inclusion criteria:

- The requirement embodies a functional aspect concerning the software system.
- The requirement contains a notion of temporal behavior, as indicated by keywords such as *always*, *before*, *after*, *next*, *finally*, *eventually*, *until*, etc.

Consequently, we excluded non-functional SpaceWire requirements [24], such as *"The line receiver shall maintain correct operation for differential input voltages of up to 600 mV magnitude,"* as they focus on performance rather than defining software system behavior. Similarly, functional requirements lacking a temporal aspect, such as *"Zero or more data characters at the front of a packet shall form a destination address,"* were excluded.

Our inclusion and exclusion criteria result in the extraction of 89 requirements from the SpaceWire artifact investigated. The extracted requirements, specified using natural language (i.e., English), can be found in our replication package[1].

III. Formalizing SpaceWire

After requirements extraction, we proceed to formalize them in a *natural* manner by selecting suitable logical frameworks. This section defines naturalism in formulae and outlines the logical frameworks ultimately derived.

A. General Characteristics of Natural Formalization

Achieving a natural formalization of requirements is crucial for ensuring clarity and precision in system specification, while

minimizing unnecessary complexity. In general, the importance of balancing expressiveness and simplicity in temporal logics has been recognized as an active area of research. The concept of a natural process of formalization is investigated from different perspectives, such as philosophical discussions [29], existing approaches towards formalization [30], and the automated formalization of requirements [31], [32].

While there is no universally accepted definition of *naturalism* in formalizing requirements, building on insights from the literature, we adopt a pragmatic approach for the purpose of this paper. The process of translating natural language requirements into logical formulae begins by identifying temporal operators, which may be encoded explicitly or implicitly within the requirement. For instance, explicit encoding occurs when temporal behavior is directly addressed in the natural language, making it straightforward to translate into formulae. Implicit encoding, on the other hand, requires interpretation within a given context, domain knowledge, or resolving ambiguities caused by synonymous or vague wording.

Using this operators-centric approach, we declare a formalization of a requirement as natural when the following three characteristics apply:

1) The formalization is solely based on the temporal operators used in the requirement, regardless of having a previously selected logic in our mind.
2) The logic used is minimal, with just enough expressiveness to capture the requirement, prioritizing adequacy over strength.
3) Compact formalizations are favored over longer ones, given that they also support intuitive comprehension.

To further convey the meaning of natural formalizations, consider the converse – a formalization where, e.g., one of the characteristics above is not fulfilled. Firstly, a requirement may point to solely LTL temporal operators but a more expressive logic is employed to formalize it. Secondly, there may be the case where an unusual, verbose formula is obtained, but a more intuitive and succinct way exists. For example, the formula using a constrained temporal operator $\square_{(0,n)}A$ would be more natural than writing the following construct:

$$A \wedge \mathcal{X}A \wedge \mathcal{X}(\mathcal{X}A) \wedge \cdots \underbrace{\mathcal{X}(\mathcal{X}(\mathcal{X}(\cdots \mathcal{X}}_{n \text{ times}} A))).$$

Beyond these characteristics of naturalness in logical formulae, the granularity of atomic propositions (APs) is another key factor in the translation, as it determines the level of detail at which a requirement is formally represented. Again, we take a pragmatic approach, striving for the finest granularity achievable while maintaining the necessary level of coarseness. We will exemplify this pragmatism later in this section.

B. Application to the Extracted SpaceWire Requirements

Applying our operators-centric approach of natural formalization, we categorize the SpaceWire requirements according to the operators used in the logical structure of the requirement. We present an excerpt of the formalized SpareWire requirements in Table I, comprising a curated subset. These

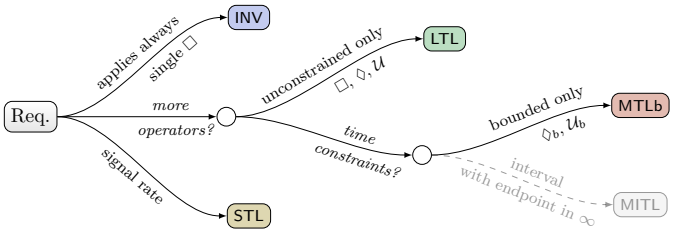

Fig. 1. Operator-centric approach to natural formalization: Logic framework decision tree derived from formalizing SpaceWire requirements.

requirements, labeled as R1 to R9, serve as reference points throughout the remainder of this paper. For the complete data set we refer to the replication package.

Figure 1 depicts the decision tree developed during the formalization of the extracted requirements. While not universally complete, it fully represents the process used to achieve natural formalizations for the 89 SpaceWire requirements. In the sequel, we explain the rationale and methodology behind the construction of the decision tree, referring to the examples shown in Table I for the sake of illustration.

Trivial global conditions. The simplest requirements we identified are conditionals that are always true, and therefore expressed by a single temporal operator $\square$, conveying the temporal notion of *always* (e.g., R1). We term those logically simple structured requirements *invariants* (INV). Although the invariants we found in SpaceWire may be further categorized in more detail (e.g. always-only requirements, action steps with no complex behavior, and event-condition-action requirements), we abstain from discussing these subcategories in this paper as they are not relevant to the discussion on the level of temporal logic.

Linear unconstrained behavior. Starting with R2 and R3, the expressiveness of INV no longer suffices as these requirements specify additional temporal behavior, necessitating more operators than just a single $\square$. To this end, we employ LTL, which has found diverse applications [13] and which is widely used for formulating statements with a temporal aspect [1]. For cases with additional temporal behavior, which is specified as unconstrained, we introduce the next leaf node in the decision tree, accounting for $\Diamond$ (*eventually*) and $\mathcal{U}$ (*until*) operators of LTL. Note that we intentionally did not express the $\mathcal{X}$ (*next*) in the decision tree path description leading to LTL as this operator is unconstrained by definition.

Recalling the applied pragmatism on AP granularity level outlined earlier, R4 serves as an underpinning example. We encoded *firstNullReceivedWithoutError* intentionally as a single AP as separating it in $(\textit{firstNullReceived}) \wedge (\neg \textit{error})$ introduces a problem: if the first 'Null' is received with an error, *firstNullReceived* would never hold again as subsequent nulls would no longer be the first one, rendering the formula unsatisfiable. This demonstrates the *'as coarse grained as necessary'* pragmatism applied here. Note that one could also propose to use LTL with past operators (e.g., *previously* [33])

TABLE I
EXAMPLE FORMALIZATIONS OF SPACEWIRE REQUIREMENTS

Ref	[Requirement ID] Requirement Text	Operators	Logic	Formalization
R1	[1006] Null detection shall be enabled **whenever** the receiver is enabled.	$\Box$	INV	$\Box((\textit{receiver enabled}) \rightarrow (\textit{Null detection enabled}))$
R2	[2008] The data link layer shall not send any N-Chars to the encoding layer **until** it has received one or more FCTs from the encoding layer [...].	$\Box$, $\mathcal{U}$	LTL	$\Box(\neg(\textit{send NChar})\ \mathcal{U}\ (\textit{FCT received}))$
R3	[2013] **When** the link is initialised or re-initialised, one FCT shall be sent for **every** eight N-Chars that can be held in the receive FIFO up to the maximum of seven FCTs.	$\Box$, $\mathcal{X}$, $\mathcal{U}$	LTL	$\Box((\textit{link state} : (\textit{initialised} \vee\ \textit{reinitialised})) \rightarrow (((8\ \textit{NChar held}) \rightarrow \mathcal{X}(\textit{one FCT sent}))\ \mathcal{U}\ (\textit{Num sent FCT} \leq 7)))$
R4	[2037] The gotNull.indication primitive shall be passed to the data link layer, **when the first** Null is received without any errors **after** the receiver has been enabled.	$\Box$, $\mathcal{X}$, $\Box$	LTL	$\Box(\textit{receiverEnabled} \rightarrow \mathcal{X}(\Box(\textit{firstNullReceivedWithoutError})) \rightarrow (\textit{gotNullPassed}))$
R5	[5002] If the host system tries to send an interrupt acknowledgement **too soon after** a corresponding interrupt code has been received [...] the result is indeterminate for that specific interrupt. The new interrupt acknowledgement code that the node sends can either be discarded by a router, or **repeatedly** propagated through the network [...].	$\Box$, $\mathcal{X}$, $\mathcal{U}$, $\Box$, $\Box$, $\Diamond$	LTL	$\Box(((\textit{corresponding interrupt code received}) \rightarrow \mathcal{X}\neg(\neg(\textit{send interrupt acknowledgement})\ \mathcal{U}(\textit{interrupt code propagated}))) \rightarrow (\Box(\textit{new interrupt code discard}) \vee \Box(\Diamond(\textit{new interrupt code propagated}))))$
R6	[3014] The **delay between** the interrupt code arriving and the interrupt acknowledgement being generated shall be less than the maximum time determined for a node to generate an interrupt acknowledgement code.	$\Box$, $\Diamond_I$, $I = \textit{interval}$	MTLb	$\Box((\textit{interrupt code arriving}) \rightarrow \Diamond_{(0,t)}(\textit{interrupt ack})\ \textit{generated}),\ t \leq \textit{max interrupt ack time}$
R7	[3001] The Link state machine shall leave the ErrorReset state [...] **When** the 6,4 µs timer is **elapsed** and LinkDisable is deasserted, move to the ErrorWait state.	$\Box$, $\Diamond_I$, $I = \textit{singleton}$	MTLb	$\Box((\Diamond_{6.4\mu s}(\textit{LinkDisable deasserted}) \rightarrow \mathcal{X}(\textit{ErrorWait state}))$
R8	[4001] **After** a reset or disconnect (see clause 5.4.8) an output port shall start operating at a data signaling rate of 10 ±1 Mb/s.	$\Box$, $\mathcal{X}$	STL	$\Box((\textit{reset} \vee \textit{disconnect}) \rightarrow \mathcal{X}(\ 9 \leq S_{data}(t) \leq 11))$
R9	[4002] The SpaceWire output port shall operate at 10 ±1 Mb/s **until** set to operate at a different data signaling rate.	$\Box$, $\mathcal{U}$	STL	$\Box((9 \leq S_{data}(t) \leq 11)\ \mathcal{U}\ (\textit{set different rate}))$

to formalize the notion of *first* in R4. Then, the proposition *firstNullReceivedWithoutError* could indeed be separated into two APs, *NullReceived* and *NoError* previously. However, the result would be overly complex and less natural.

The last requirement naturally formalized in LTL we selected for being discussed in the paper is requirement R5. It is noteworthy in our case study as it incorporates the notion of *"repeatedly"*, which we capture using the unbounded $\Diamond$ (*eventually*) within a $\Box$ operator.

Handling bounds and different models of time. In cases where a requirement includes constrained temporal operators, as in R6 and R7, LTL lacks the necessary expressiveness, pointing towards the family of *Metric Temporal Logic* (MTL [7], [12], [34]). Informally, MTL can be seen as a generalization of LTL in which temporal operators are replaced by time-constrained versions. There exists a range of metric temporal logics that account for expressing time boundaries, typically added as a number or interval to temporal operators.

In terms of our natural formalization of SpaceWire requirements, we initially considered two prominent fragments of MTL, known as *Metric Interval Temporal Logic* (MITL) and *Safety Metric Temporal Logic* [14], [35] (MTLb, read as MTL bounded) [14], [34]. In MTLb, temporal operators may be augmented by *bounded* intervals only, often abbreviated by using a notation with a single number. On the contrary, MITL supports *unbounded* time intervals, indicated by endpoints at infinity. For example, expressing a time interval such as (x, ∞) is possible in MITL, whereas in MTLb such intervals cannot be expressed for $\Diamond$ and $\mathcal{U}$ operators.

Interestingly, none of the SpaceWire requirements we encountered involves constrained operators with unbounded $\Diamond$ and $\mathcal{U}$ operators. In sum, MTLb turned out to be sufficient to naturally formalize all time-constrained SpaceWire requirements considered in our study, using bounded intervals (e.g., R6) and singleton numbers referring to exact time points (e.g., R7). For the sake of completeness, though never

chosen in terms of our study, we include the MITL path with unbounded intervals in our decision tree illustrated in Fig. 1.

Lastly, we note that MTL in general and MTLb in particular may be defined based on different models of time, which can be discrete or continuous [36] implying different semantics [14], [37] (i.e., pointwise vs. continuous semantics). While this did not influence the logics selection process in terms of our natural formalization, we found pointwise semantics sufficient for the SpaceWire case, as time constraints required only natural numbers, with no need for real numbers.

Coping with signal rates and quantities. Last but not least, certain requirements contain the concept of *signal*, predicating over a *signaling rate* or *quantity* in each moment of time. R8 and R9 exemplify this. Consequently, requirements containing propositions on resetting signals or external action on the signals are not contained in STL as they sufficiently formalized in LTL (following our characteristics on natural formalizations). To this end, we employ *Signal Temporal Logic* (STL) [38], [39], where at any time point a real number denoted by $S(t)$ is assigned, capturing the signal amount in t as the signal rate. Such requirements are deemed a special case, yielding a branch in the decision process.

IV. Quantitative Analysis of Formulae

This section presents the results of a quantitative analysis of the 89 formulae naturally specifying the 89 requirements extracted from the SpaceWire document. The analysis was guided by the following two research questions:

RQ1: What is the **distribution** of **natural** logics used for the transcribed SpaceWire requirements, and can they be mutually translated?

RQ2: What is the **engineering complexity** of the natural formulae for a transcribed SpaceWire requirement and does it differ among the logics?

The rationale behind RQ1 is twofold. First, by studying the distribution of the natural formalizations over the respective logical frameworks, we aim to spot general trends and outliers. Second, we account for the fact that natural formalizations are not necessarily the only possible ones. As noted at the end of Section III-A, a time-bounded requirement is most naturally formalized in MTLb but can also be translated into LTL. To that end, we analyze whether a given formalized requirement can also be translated into another logical framework. We deem a given formalization as conditionally translatable if certain assumptions are required due to varying underlying temporal models. As we are only considering translations within the set of logics obtained during the formalization process, we speak of *mutual translatability*. Note that the translatability of each formalized requirement must be assessed $c-1$ times, where c is the number of distinct logical frameworks used in the formalizations. Let further n be the number of formalized requirements, a total of $n(c-1)$ translation decisions – each classified as either *yes*, *no*, or *conditional* – must be made.

The second guiding research question RQ2 examines what we term the *engineering complexity* of all natural formalizations across the logics. This term is used to distinguish our focus from *computational complexity*, which is not the subject of this study. Specifically, we use the syntax structure of logical formulae as a proxy for engineering complexity. To that end, we analyze the *abstract syntax tree* (AST) of each formula, collecting quantitative metrics characterizing the complexity of the AST. Metrics include the level of nesting or *AST height* (ASTH), counts of *logical operators* (LOPs), *temporal operators* (TOPs), *atomic propositions* (APs), and *comparison operations* (COPs) within the APs if such where appearing in requirements treated. As we contemplate that engineering complexity positively correlates with the diversity of operators present in a given formula, we additionally determine the *Shannon entropy* [40] to measure the diversity of LOPs and TOPs across the different logics. COPs were intentionally excluded, as they are typically abstracted away using APs. For instance, in a 3-operator scenario, a formula that uses the first and third operators once and the second operator twice can be represented as the vector $v = [1, 2, 1]$. These binned counts are then used to calculate the base 2 Shannon entropy $H(v) = -\sum_i p_i \log_2(p_i)$ where p_i is the proportion of each bin's count relative to the total. The resulting entropy $H(v) = 1.5$ indicates almost maximal entropy, reflecting close to perfect operator diversity. Perfect diversity is reached in uniform vectors, with $v_i > 0$.

In order to address both research questions, we leverage `tlparser`, a command line tool developed as part of this work. It automates the parsing of temporal logic formulae, extraction of statistical metrics, and generation of reporting visualizations, ensuring reproducibility. It supports all operators relevant to the studied logics (detailed in the replication package) and is designed to be easily extensible to accommodate additional operators and logics in the future.

For demonstration purposes, consider the Formula 1 below as running example, being processed by the `tlparser` in two steps.

$$\square\,(y \wedge (u = 9) \rightarrow \lozenge\,(\neg y \vee i < 3)) \tag{1}$$

First, existing COPs ($=$, $<$) in each comparison instance are counted and then resolved with letter replacements to transform these instances into parsable APs (i.e. $i < 3$ becomes i_gt_3). In the second step, the formula with resolved comparison instances is parsed into an AST object which allows to extract APs (i.e. i_gt_3, u_eq_9, and y), TOPs (G and F or $\square$ and $\lozenge$ respectively), and LOPs ($\wedge$, $\vee$, $\rightarrow$, and $\neg$). Consequently, we harvest the statistical data of having ASTH $= 5$, APs $= 3$, COPs $= 2$, LOPs $= 4$, and TOPs $= 2$. The entropy of the LOPs and TOPs present in Formula 1 therefore yields ≈ 2.585.

A. Answer to RQ1

Fig. 2 shows the distribution of logics used to naturally formalize the SpaceWire requirements and their mutual translatability. In Subfigure (a), we observe that the majority of the total $n = 89$ requirements can naturally be formulated as INV (32) and LTL (39) expressions, followed by MTLb (15). Only 3 instances utilize the notion of signaling rates, resulting in

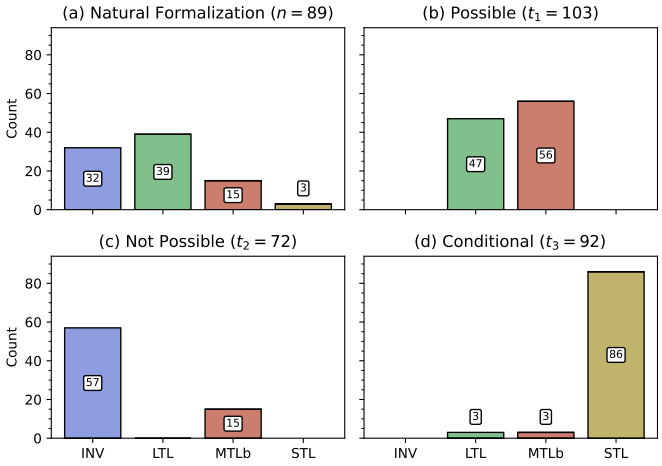

Fig. 2. Formalized requirements and their translation potential.

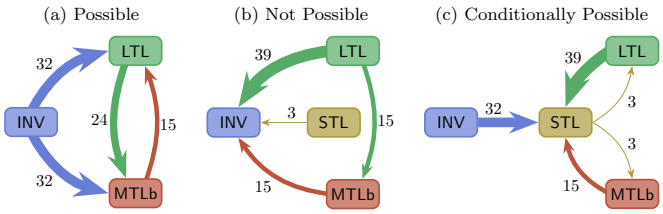

Fig. 3. Translation directions of the natural formalizations.

an STL specification. Subfigures (b), (c), and (d) illustrate the mutual translation potential among the set of employed logics (listed in x-axes). In the case of SpaceWire, this set – which we derived from the decision tree (cf. Fig. 1) – has a cardinality of $c = 4$ which results $n(c-1) = 89(4-1) = 267$ translation questions to be posed. Given the three possible outcomes of answers, we denote the answer classes as t_i, with $i \in \{1, 2, 3\}$ and $\sum_{i=1}^{3} t_i = 267$.

According to Subfigure (b), translations are possible to LTL in 47 cases and to MTLb in 56 cases, indicating that more than half of the considered requirements can be formalized – though not most naturally – using either of these two logical frameworks. Subfigure (c) shows that 57 formalizations cannot be translated to INV, and 15 cannot be translated to MTLb. Subfigure (d) illustrates that translation is conditionally possible for almost all (86) requirements in STL, and 3 times each in LTL and MTLb.

Fig. 3 apportions the aspect of mutual translatability illustrated in Fig. 2 (b-d) in greater detail using a directed acyclic graph. In Subfigure (a) of Fig. 3, for example, we observe that 15 naturally formalized requirements in MTLb can be translated into LTL. This is, for instance, possible through the application of successive $\mathcal{X}$ operators as showed earlier and could be applied on R7 from Table I. We note in Subfigure (b) that not all LTL formalizations can be translated to MTLb. An example of one of these recorded 15 instances is R2, which contains an unbounded $\mathcal{U}$ operator, or R5 given its unbounded $\Diamond$ operator. In contrast to this, R4 can be expressed in MTLb. Lastly, Subfigure (c) apportions translations that we deem possible under certain conditions. We deem translations to STL possible under circumstances due to the different semantics of STL. Firstly, dense time semantics pose challenges for translations between pointwise and dense time representations. Albeit there are some attempts in this direction [41], mostly investigated in scope of MTL rather than STL. Secondly, as APs in STL are propositions over *signals* (cf. R8, R9), they can, e.g., translated to MTLb with dense time, given the signal propositions would be reduced to APs.

Summarized Answer to RQ1

The analysis of 89 requirements reveals that most of them represent either trivial global conditions (32), classified as INV (i.e., invariants), or can be naturally formalized as LTL expressions (39). Only a few cases (15 MTLb and 3 STL) were naturally formalized using different logics. We record that 47 of the requirements not naturally formalized in LTL can be translated to LTL. Likewise, 56 requirements are translatable to MTLb. Under certain assumptions, all requirements (86) can be, though unnaturally, translated to STL.

B. Answer to RQ2

Figure 4 aggregates the metrics used to describe the engineering complexity of the SpaceWire requirement document. Each metric is computed over the set of naturally required logics (x-axes) and visualized using violin plots to illustrate the distribution of metrics across individual properties corresponding to requirements. The number of formalized requirements n per logic is stated below the x-axis categories and remains the same for all metrics. Additionally, we explicitly state the mean μ, median M, and standard deviation σ for each logic in each metric. The box plots within the violins indicate the interquartile range, while circles highlight outliers.

Analyzing the Subfigures of Fig. 4 from top left to bottom right, the following observations are worth noting. On average, all exhibit a similar amount of APs, with MTLb leading this ranking and STL having a more concentrated distribution with a smaller range. Requirements formulated in STL require on average a higher amount of COPs with greater scattering compared to the others. In the case of LOPs, MTLb exhibits a notably higher mean count and wider spread. In contrast, INV and STL show lower and more consistent counts, as indicated by their narrower distributions. Slightly more TOPs are present in LTL, MTLb, and STL formalization as in INV. LTL distinguishes itself from the others through its wider spread and 4 outliers. As for the ASTH incurred by each property, we observe that INV is shallower compared to the others, indicating generally simpler structural expressions. A similar pattern can be observed looking at the entropy.

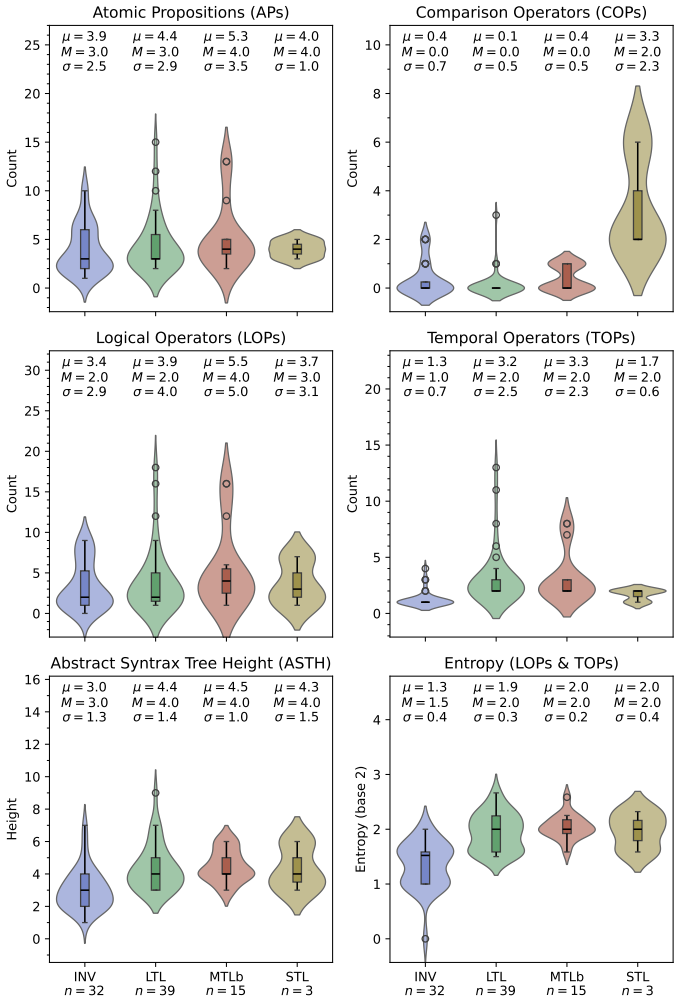

Fig. 4. Engineering complexity of the natural formalizations.

Summarized Answer to RQ2

We conclude the following moderate trends regarding engineering complexity. Variability and quantity are highest for atomic propositions (APs), logical operators (LOPs), and temporal operators (TOPs) in LTL and MTLb, while comparison operator (COP) usage is negligible across all logical frameworks except for STL. In terms of abstract syntax tree height (ASTH), INV expressions exhibit the lowest values, with other frameworks demonstrating higher values. A similar pattern emerges for entropy, though the differences are more pronounced.

V. Discussion

This section discusses our quantitative analysis results and sets them in relation to other theoretical and noteworthy aspects beyond translatability and engineering complexity. Subsequently, we discuss potential implications for both practitioners and researchers, before we finally discuss the threats to the validity of our results.

A. Reflection on RQs

Quantitative observations. The quantitative results presented in Section IV illustrate the inherent and multi-faceted facts about the natural translation of SpaceWire formalization that are unique to our approach. Figure 2, Subfigure (a) demonstrates that the majority of requirements can be naturally expressed in LTL, with a significant number of invariants (INV). This indicates that LTL, established as a foundational temporal logic, is sufficiently expressive to formalize a substantial portion of SpaceWire requirements. The second category is MTLb, followed by STL covering the smallest proportion of requirements.

Mutual translatability. Since the natural formalization strategy employed selects the logic with the least amount of temporal operators needed, we stick to the minimal logic which is expressive enough to formalize a requirement. However, there might be a stronger logic which shares the same operators as in the natural formalization, and therefore the formula is a mutual formula of these logics [37]. Furthermore, other translations might evolve using workarounds and relying on certain assumptions. For example, a formula in MTLb can be translated into LTL with respect to pointwise semantics, which we assume for all the MTLb formulae in our study.

In Fig. 2 (b-d) resp. Fig. 3, we clarify the possible mutual translations across the logical frameworks used in our natural formalizations. In fact, most of the formulae, though not naturally, can be translated to LTL and MTLb, as shown in Fig. 2, Subfigure (b). Note that these values add on top of the requirements which are naturally formalized using these logics, shown in Subfigure (a). In other words, a total of 86 out of the considered 89 SpaceWire requirements (39 naturally + 47 translations) could be formalized in LTL. This is all but the three requirements revolving around signals. Respectively, 71 of the considered 89 requirements ($15 + 56$) can be formalized in MTLb. Formulas with unconstrained $\Diamond$ and $\mathcal{U}$ operators are not expressible in MTLb.

Other translations, termed conditional translations, involve translations which are possible under circumstances, i.e., depending on the semantics used. This is particularly the case for STL since its underlying model differs from those of the other temporal logics. In STL, signal values (as real numbers) are assigned to each point in time (real time), and atomic propositions are defined as predicates over these signals. This approach is highly expressive, enabling all SpaceWire requirements to be represented in STL through an appropriate, albeit not natural, encoding. However, the reverse direction, i.e., translating STL specifications into logics with simpler models, is rarely feasible.

Engineering complexity. On the engineering complexity of formulas illustrated in Fig. 4, we observe that the number of APs, COPs, LOPs and ASTH logical formulations exhibited similar patterns. This result justifies a reasonable meaning of the *natural translation* concept. Some notable differences can be observed for STL formulae which tend to involve a more frequent use of comparison operators – expected due to

signals. However, we stress the fact that only three SpaceWire requirements were formalized naturally in STL, weakening its statistical significance. The number of temporal operators in INV is much less – an expected outcome. ASTH follows a similar pattern, a promising result indicating that greater expressiveness does not incur deeper nesting structures in the corresponding formula, which is desired from a practical engineering perspective. Finally, the Shannon entropy among LOPs and TOPs aligns with intuitive expectations. INV formalizations exhibit the least diversity in operator usage. The operator entropy in the other logical formulations is higher compared to INV but resides at similar levels and exhibiting a similar spread.

B. Beyond Natural Formalization and Engineering Complexity

The mutual translation of formulas in different logics raises the question of whether a "super" logic exists in which all the requirements could be formalized. From a theoretical point of view, there are more powerful temporal logics such as TPTL [42] and mu-calculus [43] which enjoy more expressiveness than the logics employed in this work. As we have already seen for the mutual translations among the logics used in our natural formalizations, the question of whether all requirements can be formalized in a single logic remains to be answered conditionally depending on the assumed semantics. In the case of SpaceWire, we would have to deviate from our natural formalizations and switch to more powerful models yielding dense time semantics.

While expressiveness and elegance of more powerful logics are worthwhile to study from a logician's perspective, from a practical point of view, we argue that these properties are not necessarily the most desirable ones. So far, we have already discussed the notion of naturalism, leading to logical formulae of acceptable engineering complexity for the SpaceWire case. Moreover, given that our study serves as a reality check for both practitioners and researchers, it is important to highlight that all the logics used in our natural formalizations are decidable. This ensures the practical applicability of reasoning algorithms employed by various tools, including theorem provers, model checkers and other tools promoted by formal methods. This observation is particularly noteworthy in the case of MTL, which is undecidable in general but decidable for MTLb [34], [44], the only fragment that occurs in the natural formalizations of SpaceWire requirements.

C. Implications for Practitioners

Language to property relationship. This paper deployed a problem-oriented approach to formalize requirements stated in natural language. In addition to outlining our methodology, we identify key properties that characterize naturalism in formal logical expressions and demonstrate how these properties inform the choice of an appropriate logical framework (cf. Fig. 1). Therewith we provide practitioners with insights into the systematic process of encoding temporal behavior in software requirements, emphasizing the impact of naturalism on the effectiveness of formalizations.

Tool support. In formal methods, it is essential to provide clear justifications for the formulation of specific requirements, especially when aiming to utilize tools such as model checkers, solvers, or runtime verification frameworks. These tools, while powerful, are typically constrained by the logical frameworks they support and offer interfaces to. Our findings on mutual translatability and the observation that most SpaceWire requirements can – albeit not always most naturally – be formalized within the same logical framework enhance adaptability and expand the applicability of tools across diverse verification contexts. Moreover, our quantitative analysis revealed that a substantial portion can be expressed as invariants. This observation is particularly relevant because invariants are relatively straightforward to handle in software systems, requiring less complexity in their verification compared to other property types. This enhances the feasibility of employing various tools and streamlines the implementation process, even more when other non-LTL formulae become translated into LTL.

Creating a comprehensive reference table of representative tools for each logical framework discussed in this paper proved challenging. The status of relevant tools is often prototypical and may often lack clarity about the specific frameworks or fragments they support. While based on anecdotal evidence, this highlights the critical need for improved visibility and production-level tooling within the formal methods domain, fostering industry adoption.

Fingerprinting. Practitioners prioritize implementation and the complexity involved in it. To address this, we introduced the concept of engineering complexity to evaluate formalized requirements. Using our method, which analyzes ASTs by examining aspects such as operators and entropy, we offer a systematic way to assess and comprehend the structural and logical intricacies of requirement documents. In essence, this approach *fingerprints* the requirement document. Such fingerprinting enables the clustering of requirements into equivalence classes, fostering consistency and coherence in requirement formulation and its formalization process. Consequently, it enables the detection of inconsistencies in how requirements are formulated. Such inconsistencies may arise from overly detailed or overly high-level requirements, structural outliers, or gradual drift in writing style over time. Building on this, fingerprint components could also serve as features for describing formulae. These features could then be used to train prediction models that estimate the complexity – and consequently, the feasibility – of implementing a given requirement against a particular setting.

D. Implications for Researchers

Observed Pareto principle. The distribution of SpaceWire requirements across logics used for their specification shows that most are expressible in LTL, while more expressive logics are used for a minority. This observation makes the case for 'simple' logics instead of more expressive logics which introduce complications in terms of automated reasoning. We noted such a case for the MTL family, where less expressive fragments of it were still expressive enough to formalize our

requirements. Though, we acknowledge the need for more expressive logics that can cover all requirements one may encounter beyond our SpaceWire case study.

Expressiveness and realizability. In search for a 'super logic' which is expressive enough to cover all the requirements, we may end up with a strong logic with significant power of expressiveness which might have drawbacks in the implementation level. We must strike a delicate balance between theoretical and practical considerations, ensuring that both sides' expectations are partially yet effectively met. In other words, once a super logic becomes more complex, tool support starts fading away.

A result of studying the SpaceWire protocol demonstrates that all the requirements with time constraints can be expressed in MTLb which has bounded $\mathcal{U}$ and $\Diamond$ operators. This means there is no requirement with an *eventually* or *until* operator which is unbounded, eliminating the need for more expressive logics. We note that practice points to adoption of linear time for specification (including within trace checking, GR(1) synthesis, etc) – but we acknowledge that consideration of branching logics such as TCTL (with respective application areas, such as quantitative analysis, e.g., within UPPAAL [45]) are left for future work. Likewise, we emphasize possible theoretical endeavors capable to account for uncertainty in requirements, such as fuzzy [46] or deontic logics [47], [48].

Shaping future research directions. Our findings confirmed the presence of a Pareto effect in the formalized SpaceWire requirements, where the majority of requirements are invariants or can be captured using LTL. This underscores the value of classifying basic temporal logic fragments or subsets and exploring their expressive power, computational complexity, and reasoning algorithms. While significant research has focused on more expressive (super) logics, our results highlight the importance of investigating existing fragments. In particular, combining carefully selected subsets and fragments of logical frameworks could yield unified frameworks that are more specialized and thus more effective in problem-oriented settings, such as those demonstrated in this paper.

Furthermore, we demonstrated the impact of mutual translatability, particularly when naturalism in formulae is no longer prioritized. The challenges related to tool support further highlight the importance of translatability, as many tools are restricted to specific logical frameworks or their fragments. This suggests that developing *'functions'* or *'recipes'* for (automatically) translating one logical formula into another with minimal information loss is a valuable direction for further research.

E. Threats to Validity

Internal validity. Subjectivity in natural formalization poses an internal validity threat, as there may exist different formulations for some requirements, which may affect the quantitative metrics employed in our investigation – (i) several equivalent properties may exist for a given requirement, and (ii) subjective interpretations may arise due to natural language ambiguity. To mitigate this, the specification process was performed according to the decision tree of Sec. III, ensuring that formalization was consistently guided by the temporal operators present in each requirement. Therefore, we believe that any residual ambiguity has minimal impact on the outcome of the quantitative analysis performed.

Additionally, our fine-to-coarse-grained pragmatism on AP granularity poses another potential threat to validity, as the chosen granularity directly impacts the perceived engineering complexity. To address this, we explicitly declared the pragmatism behind how granularity was handled, ensuring that reproducibility and comparability of the resulting fingerprinting process – both within and across requirement documents – remains feasible.

External validity. While the SpaceWire protocol is representative within its domain, relying on a single case limits the generalizability of our results towards other domains. Our investigation is tailored to SpaceWire use cases, and different contexts may yield varying quantitative outcomes and logical frameworks. However, we believe our overall methodology remains applicable across domains. In support of this, we provided a detailed specification of our methodology and a replication package, encouraging future empirical studies to explore larger and more diverse requirement sets over time.

Construct validity. The selected quantitative metrics (e.g., APs, COPs, LOPs, TOPs, and entropy – Sec. IV) are syntactic in nature as they could influence an implementation, ignoring factors like semantics or algorithm complexity (e.g., for model checking). This deliberate choice aims for (i) generality (since an overall objective for some system analysis such as synthesis or model checking is not assumed) and (ii) fingerprinting sets of specifications in an objective manner. By restricting the objective, logic employed and selected algorithms, engineering complexity could be assessed more accurately and confidently. On a similar note, we avoid a focus on semantic formula characteristics and do not address issues of simplification, uniform representation, normal or canonical forms.

Conclusion validity. Our approach to measuring engineering complexity is not exhaustive, as it excludes aspects such as interpretational difficulty and the behavior of formulas within a system's broader context. While the reliability of our method may decrease for relatively small formulas, we argue that our problem-oriented focus ensures practical value, as even complexity measurements for small formulas can provide insights for real-world applications.

VI. Related Work

This paper broadly addresses the problem of transitioning from natural language software requirements into precise specifications. Therefore, we approach related work from two perspectives. The first includes discussing other works by the wider community addressing the broader specification challenge, along with ones adopting case studies. Secondly, since choosing the suitable logic for a requirement is a non-trivial problem, we further discuss relevant studies addressing suitability and translation across different temporal languages, also justifying our selected logics.

Several approaches have pursued accessibility of temporal logic-enabled reasoning to practitioners and engineers alike. Specification patterns have become a popular solution. Dwyer et al. [20] introduced patterns for safety properties, which were later extended to address quantitative [49] real-time [10] and probabilistic [50] properties. Semiformal approaches use formal descriptions through an intermediate representation [51], natural language [30] or utilizing structured language such as the FRET approach [19]. In the latter case, a structured natural language can be used for incremental specification, visualization and tool interfacing [52]. Orthogonally, LTL synthesis has been approached from several directions, e.g., via classical semantic parsing, as translation or with evolutionary methods [5], [22]. Lately, large language models have also shown potential [3], [4], [53].

Model problems and benchmark cases such as the RPC memory specification or the ARM Advanced Microcontroller Bus Architecture [54] have shown to be highly useful to the community [55], justifying our choice of a well-known, flight-proven protocol. Formalization endeavors have been undertaken in multiple and diverse domains and systems, albeit typically targeting in each case specific logical frameworks and purposes with numerous examples which include on-board electric power systems [56], web services [57], hardware-software systems [58], railway control systems [59], or aerospace [60]. Our scope is different in that we build on such body-of-knowledge but extend towards (i) methodological aspects (*how* we arrived to a specification, considering a logics arsenal) instead of a specific formalism and (ii) we accompany our dataset of SpaceWire with tool support to aid reproduction and further utilization by the community. To the best of our knowledge, this is the first effort to take such a problem-oriented approach.

Temporal logics in general have proved a well-established foundation and have been extensively studied, ranging from logics as broadly applied in formal methods [2] to enable reasoning on systems behaviour [7], [61]–[63]. By extending LTL, studying real-time specifications started by introducing metric temporal logic by Alur and Henzinger [11], [12], [64], from the perspective of expressiveness and complexity. Real-time system behaviour specifications are further studied in [7], however, in many approaches the empirical study of requirements with respect to the theoretical aspects is not considered. In our investigation, we confront SpaceWire as a real-world case, and address its formalization process with respect to a set of provided and well-established logics. In order to analyze the possible translation of formulas, we point out that while some translations between logics are trivial, however, others require more rigorous methods as done in [65] and [37], employed in our study.

VII. Conclusion and Future Work

In this paper, we addressed the long-pursued challenge of transitioning from natural language to formal specification by tackling the SpaceWire data-handling protocol often adopted on spacecraft and other on-board systems. We systematically extracted software a total of 89 requirements and transcribed them into properties according to a decision process over different established logics with the aim of maximizing natural encoding. We assessed the suitability of respective logical languages, highlighting their practical traits from a systems engineering perspective. We believe that findings from this endeavor serve as a reality check as they offer a clearer understanding of how formal methods can be applied effectively in the context of aerospace and other critical domains, from the viewpoints of both practitioners and researchers.

Regarding theoretical aspects, we plan to investigate applicability and a supporting methodology towards use for synthesis and verification in particular, continuing with the SpaceWire protocol as a base case in tandem with the dataset and tool support accompanying this paper. For synthesis or planning, investigation of suitability and potential transformations to synthesizable logic fragments (e.g., GR(1) [66], [67]) would be a conceptual next step. Verification, particularly at runtime, requires investigating monitorability twofold: On one hand regarding appropriate models of time and on the other hand regarding formulation of properties e.g., for INV or LTL to appropriate three- or four-valued [62], [63] projections. Naturally, extending to other requirement documents is crucial to validate whether the presented results are generalizable and insights transferable to other domains.

Regarding engineering aspects, we aim to further develop our notion of fingerprinting requirements documents and respective analytics extraction, which we believe can play a larger role in adjacent requirements engineering activities. Integration with other tools such as FRET [19] would go a long way in supporting the early design cycle, as well as extending `tlparser` towards bridging to other domain-specific languages used by other tools. We envision extending our engineering complexity metrics to incorporate runtime performance of reasoning tasks enabled by these integrations, along with the semantic aspects of formulae. Orthogonally, our specifications' dataset may be leveraged to train GPT models for human-in-the-loop arrangements targeting a trade-off between manual specification and automation. Requirements generated from such models can offer significant value from a logical perspective, since these requirements could correspond to true formulas of the logical framework, within scope of automated theorem generation [68]. Finally and with a broader view, investigating the specification process with users in an exploratory manner and in an end-to-end setting [69] along with the tool and dataset presented appears promising and likely to draw further insights for the formal method community.

Acknowledgment

This work has been supported by the Swiss National Science Foundation (SNSF) project "`RUNVERSPACE`: Runtime Verification for Space Software Architectures", grant no. 220875. Furthermore, the authors would like to thank the Swiss Group for Original and Outside-the-box Software Engineering (CHOOSE) for sponsoring the trip to the conference.

References

[1] S. Demri, V. Goranko, and M. Lange, *Temporal Logics in Computer Science: Finite-State Systems*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

[2] M. Fisher, *Temporal Logic*. John Wiley & Sons, Ltd, 2011, ch. 2, pp. 9–48. [Online]. Available: https://doi.org/10.1002/9781119991472.ch2

[3] C. Hahn, F. Schmitt, J. J. Tillman, N. Metzger, J. Siber, and B. Finkbeiner, "Formal specifications from natural language," 2022. [Online]. Available: https://arxiv.org/abs/2206.01962

[4] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 383–396.

[5] J. X. Liu, Z. Yang, B. Schornstein, S. Liang, I. Idrees, S. Tellex, and A. Shah, "Lang2LTL: Translating Natural Language Commands to Temporal Specification with Large Language Models," in *Workshop on Language and Robotics at CoRL 2022*, 2022.

[6] V. S. Alagar and K. Periyasamy, *Temporal Logic*. London: Springer London, 2011, pp. 177–229. [Online]. Available: https://doi.org/10.1007/978-0-85729-277-3_11

[7] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," *ACM Comput. Surv.*, vol. 32, no. 1, p. 12–42, Mar. 2000. [Online]. Available: https://doi.org/10.1145/349194.349197

[8] M. Rebaiaia, "Embedded systems certification using temporal logic," in *Proceedings. 2004 International Conference on Information and Communication Technologies: From Theory to Applications, 2004.*, 2004, pp. 577–578.

[9] A. Alexander and W. Reisig, "Logic of Involved Variables - System Specification with Temporal Logic of Distributed Actions," in *Third International Conference on Application of Concurrency to System Design, 2003. Proceedings.*, 2003, pp. 167–176.

[10] S. Konrad and B. Cheng, "Real-time specification patterns," in *Proceedings. 27th International Conference on Software Engineering (ICSE)*, 2005, pp. 372–381.

[11] R. Alur and T. Henzinger, "Real-time logics: complexity and expressiveness," in *Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 390–401.

[12] R. Alur and T. A. Henzinger, "A really temporal logic," *J. ACM*, vol. 41, no. 1, p. 181–203, Jan. 1994. [Online]. Available: https://doi.org/10.1145/174644.174651

[13] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, Oct. 1977, pp. 46–57, iSSN: 0272-5428.

[14] J. Ouaknine and J. Worrell, "Some recent results in metric temporal logic," in *Formal Modeling and Analysis of Timed Systems*, F. Cassez and C. Jard, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–13.

[15] S. Feng, C. Carapelle, O. F. Gil, and K. Quaas, "MTL and TPTL for One-Counter Machines: Expressiveness, Model Checking, and Satisfiability," *ACM Trans. Comput. Logic*, vol. 21, no. 2, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3372789

[16] C. Madsen, P. Vaidyanathan, S. Sadraddini, C.-I. Vasile, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Metrics for signal temporal logic formulae," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 1542–1547.

[17] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic Temporal Logic Falsification of Cyber-Physical Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, May 2013. [Online]. Available: https://doi.org/10.1145/2465787.2465797

[18] R. Bögli, L. Lerena, C. Tsigkanos, and T. Kehrer, "A Systematic Literature Review on a Decade of Industrial TLA+ Practice," in *Integrated Formal Methods*, N. Kosmatov and L. Kovács, Eds. Cham: Springer Nature Switzerland, 2025, pp. 24–34.

[19] D. Giannakopoulou, A. Mavridou, J. Rhein, T. Pressburger, J. Schumann, and N. Shi, "Formal requirements elicitation with FRET," in *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020)*, no. ARC-E-DAA-TN77785, 2020.

[20] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *International Conference on Software Engineering (ICSE)*. IEEE, 1999.

[21] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," 2019. [Online]. Available: https://arxiv.org/abs/1901.02077

[22] A. Brunello, A. Montanari, and M. Reynolds, "Synthesis of LTL formulas from natural language texts: State of the art and research directions," in *26th International symposium on temporal representation and reasoning (TIME 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[23] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Verified Software. Theories, Tools, and Experiments: 8th International Conference, VSTTE 2016, Toronto, ON, Canada, July 17–18, 2016, Revised Selected Papers 8*. Springer, 2016, pp. 8–26.

[24] European Cooperation for Space Standardization (ECSS), "ECSS-E-ST-50-12C Rev.1 – SpaceWire – Links, nodes, routers and networks," May 2019. [Online]. Available: https://ecss.nl/standard/ecss-e-st-50-12c-rev-1-spacewire-links-nodes-routers-and-networks-15-may-2019/

[25] S. Parkes and P. Armbruster, "Spacewire: A spacecraft onboard network for real-time communications," in *14th IEEE-NPSS Real Time Conference, 2005.* IEEE, 2005, pp. 6–10.

[26] B. M. Cook and C. P. H. Walker, "SpaceWire and IEEE 1355 Revisited," in *Presented at the International Spacewire Conference*, vol. 17, 2007, p. 19.

[27] D. Roberts and S. Parkes, "Spacewire missions and applications," in *2010 SpaceWire International Conference*, vol. 13, 2010, pp. 431–436.

[28] K. Pohl, C. Rupp, and K. Pohl, *Requirements Engineering Fundamentals*, 2nd ed., ser. Rocky Nook computing. Santa Barbara, Calif: Rocky Nook, 2015.

[29] J. Peregrin and V. Svoboda, "Criteria for logical formalization," *Synthese*, vol. 190, no. 14, pp. 2897–2924, Sep. 2013.

[30] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: A survey," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. Helsinki, Finland: IEEE, Jul. 2019, pp. 400–406.

[31] V. Koscinski, C. Gambardella, E. Gerstner, M. Zappavigna, J. Cassetti, and M. Mirakhorli, "A natural language processing technique for formalization of systems requirement specifications," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 350–356.

[32] D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann, "Automated formalization of structured natural language requirements," *Information and Software Technology*, vol. 137, p. 106590, 2021. [Online]. Available: https://doi.org/10.1016/j.infsof.2021.106590

[33] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, "On the temporal analysis of fairness," in *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '80*. Las Vegas, Nevada: ACM Press, 1980, pp. 163–173.

[34] J. Ouaknine and J. Worrell, "Safety metric temporal logic is fully decidable," in *Tools and Algorithms for the Construction and Analysis of Systems*, H. Hermanns and J. Palsberg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 411–425.

[35] O. Maler, D. Nickovic, and A. Pnueli, "From MITL to Timed Automata," in *Formal Modeling and Analysis of Timed Systems*, E. Asarin and P. Bouyer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 274–289.

[36] I. M. Hodkinson and M. Reynolds, "Temporal logic," in *Handbook of Modal Logic*, 2007. [Online]. Available: https://doi.org/10.1016/S1570-2464(07)80014-0

[37] U. Hustadt, A. Ozaki, and C. Dixon, "Theorem proving for pointwise metric temporal logic over the naturals via translations," *J. Autom. Reason.*, vol. 64, no. 8, p. 1553–1610, Dec. 2020. [Online]. Available: https://doi.org/10.1007/s10817-020-09541-4

[38] A. Donzé, "On signal temporal logic," in *Runtime Verification*, A. Legay and S. Bensalem, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 382–383.

[39] E. Bartocci, C. Mateis, E. Nesterini, and D. Nickovic, "Survey on mining signal temporal logic specifications," *Information and Computation*, vol. 289, p. 104957, 2022. [Online]. Available: https://doi.org/10.1016/j.ic.2022.104957

[40] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[41] K. Baldor and J. Niu, "Monitoring dense-time, continuous-semantics, metric temporal logic," in *Runtime Verification*, S. Qadeer and S. Tasiran, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 245–259.

[42] P. Bouyer, F. Chevalier, and N. Markey, "On the Expressiveness of TPTL and MTL," in *FSTTCS 2005: Foundations of Software Technology and*

Theoretical Computer Science, S. Sarukkai and S. Sen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 432–443.

[43] N. Markey and P. Schnoebelen, "Mu-calculus path checking," *Information Processing Letters*, vol. 97, no. 6, pp. 225–230, 2006. [Online]. Available: https://doi.org/10.1016/j.ipl.2005.11.010

[44] J. Ouaknine and J. Worrell, "On the decidability and complexity of Metric Temporal Logic over finite words," *Logical Methods in Computer Science*, vol. Volume 3, Issue 1, Feb. 2007. [Online]. Available: https://doi.org/10.2168/LMCS-3(1:8)2007

[45] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, pp. 134–152, 1997.

[46] L. A. Zadeh, "Fuzzy logic," *Computer*, vol. 21, no. 4, pp. 83–93, 1988.

[47] L. Åqvist, "Deontic logic," in *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*. Springer, 1984, pp. 605–714.

[48] C. Shea-Blymyer and H. Abbas, "A deontic logic analysis of autonomous systems' safety," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3365365.3382203

[49] C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vazquez, R. Calinescu, and S. García, "Mission specification patterns for mobile robots: Providing support for quantitative properties," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2741–2760, 2022.

[50] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *Transactions on Software Engineering*, vol. 41, no. 7, pp. 620–638, 2015.

[51] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner, "Automatically extracting requirements specifications from natural language," *arXiv preprint arXiv:1403.3142*, 2014.

[52] A. Katis, A. Mavridou, D. Giannakopoulou, T. Pressburger, and J. Schumann, "Capture, Analyze, Diagnose: Realizability Checking Of Requirements in FRET," in *International Conference on Computer Aided Verification*. Springer, 2022, pp. 490–504.

[53] Z. Ma, C. Wen, J. Su, M. Zhao, B. Yu, X. Lu, and C. Tian, "Towards Practical Requirement Analysis and Verification: A Case Study on Software IP Components in Aerospace Embedded Systems," *arXiv preprint arXiv:2404.00795*, 2024.

[54] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *Verification, Model Checking, and Abstract Interpretation: 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006. Proceedings 7*. Springer, 2006, pp. 364–380.

[55] M. Broy, S. Merz, and K. Spies, "The rpc-memory case study: A synopsis," in *Formal Systems Specification: The RPC-Memory Specification Case Study*. Springer, 2005, pp. 5–20.

[56] H. Xu, U. Topcu, and R. M. Murray, "A case study on reactive protocols for aircraft electric power distribution," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 1124–1129.

[57] R. Kazhamiakin, M. Pistore, and M. Roveri, "Formal verification of requirements using spin: A case study on web services," in *Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004. SEFM 2004.* IEEE, 2004, pp. 406–415.

[58] H. Peng, S. Tahar, and F. Khendek, "Spin vs. vis: A case study on the formal verification of the atmr protocol," in *ICFEM 2000. Third IEEE International Conference on Formal Engineering Methods*. IEEE, 2000, pp. 79–87.

[59] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta, "Formalizing requirements with object models and temporal constraints," *Software & Systems Modeling*, vol. 10, no. 2, pp. 147–160, 2011.

[60] C. Boufaied, M. Jukss, D. Bianculli, L. C. Briand, and Y. I. Parache, "Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization," *Journal of Systems and Software*, vol. 174, p. 110881, 2021.

[61] A. Bauer, M. Leucker, and C. Schallhart, "Monitoring of Real-Time Properties," in *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, S. Arun-Kumar and N. Garg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 260–272.

[62] ——, "Comparing LTL Semantics for Runtime Verification," *J. Log. Comput.*, vol. 20, pp. 651–674, 06 2010.

[63] ——, "Runtime verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, p. 14, 09 2011.

[64] R. Alur and T. Henzinger, "Real-time logics: Complexity and expressiveness," *Information and Computation*, vol. 104, no. 1, pp. 35–77, 1993. [Online]. Available: https://doi.org/10.1006/inco.1993.1025

[65] U. Hustadt, A. Ozaki, and C. Dixon, "Theorem proving for metric temporal logic over the naturals," in *International Conference on Automated Deduction*, 2017.

[66] B. Aminof, G. De Giacomo, A. Murano, and S. Rubin, "Planning under LTL environment specifications," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 31–39.

[67] A. Camacho, J. Baier, C. Muise, and S. McIlraith, "Finite LTL Synthesis as Planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018, pp. 29–38.

[68] Y. Puzis, Y. Gao, and G. Sutcliffe, "Automated Generation of Interesting Theorems," in *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2006)*, vol. 2006, 01 2006, pp. 49–54.

[69] D. Ma'ayan and S. Maoz, "Using Reactive Synthesis: An End-to-End Exploratory Case Study," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 742–754.