Formal Methods in Requirements Engineering: Survey and Future Directions

Robert Lorch, Baoluo Meng, Kit Siu
Abha Moitra, Michael Durling, Saswata Paul
Sarat Chandra Varanasi
GE Aerospace Research
Niskayuna, NY, United States
{robert.lorch,baoluo.meng,siu,abha.moitra}@ge.com
{durling,saswata.paul,saratchandra.varanasi}@ge.com

Craig McMillan
GE Aerospace
Evendale, OH, United States
craig.mcmillan@ge.com

ABSTRACT

Requirements engineering plays a pivotal role in the development of safety-critical systems. However, the process is usually a manual one and can lead to errors and inconsistencies in the requirements that are not easily detectable. Formal methods are mathematically rigorous techniques that can aid engineers to detect errors and produce consistent and correct requirements. We survey a variety of requirements capture and analysis tools presented in the literature. Specifically, we focus on tools that incorporate formal methods techniques into their analyses. We discuss the various tools' strengths and weaknesses, identify current trends in requirements engineering research, and highlight open research questions.

CCS CONCEPTS

• **Software and its engineering** → **Requirements analysis**; **Specification languages**.

KEYWORDS

Requirements Engineering, Requirements Analysis, Formal Methods

ACM Reference Format:
Robert Lorch, Baoluo Meng, Kit Siu, Abha Moitra, Michael Durling, Saswata Paul, Sarat Chandra Varanasi, and Craig McMillan. 2024. Formal Methods in Requirements Engineering: Survey and Future Directions. In *Formal Methods in Software Engineering (FormaliSE) (FormaliSE '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3644033.3644373

1 INTRODUCTION

Numerous studies have shown that for safety-critical systems, verification and validation costs account for a significant fraction of the overall design costs. Most errors are introduced early in the system development lifecycle, but often go undiscovered until late in the development process, costing organizations time and money [13, 32, 82]. To help address this issue, formal methods can be incorporated into the system design phase through formal requirements capture and analysis tools. These tools provide a means for comprehensive, automated analysis of system requirements for defects (i.e., underspecification, contradictions, etc.), allowing the discovery of errors earlier in the development process. There are many such tools in the literature, all fulfilling the same general task but with slight strategic and theoretical variations. Thus, a systematic overview of the current tools, as well as a discussion of green areas for further improvement, is desired.

Requirements engineering is indispensable in the development of safety-critical systems. Safety-critical software development standards, like DO-178C [86], outline the guidelines and requirements for the certification of software used in civil aviation systems. It provides guidance on the requirements engineering process and characterizes objectives related to various types of requirements[1]. DO-178C also includes objective tables related to the development and analysis of these requirements. For example, one objective states "*High-level requirements are accurate and consistent.*" Further sections detail the meaning of accuracy and consistency for high-level requirements: "*The objective is to ensure that each high-level requirement is accurate, unambiguous, and sufficiently detailed, and that the requirements do not conflict with each other,*" and similarly for low-level requirements. The NASA Systems Engineering Processes and Requirements document [77] and the FAA Requirements Engineering Management Handbook [57] also provide guidelines and best practices for the requirements engineering process. They emphasize the importance of systematically eliciting and capturing requirements and ensuring that requirements are free of conflict. The documents highlight the need for a thorough analysis of requirements to ensure they are complete and unambiguous. However, none of them provide clear definitions of the properties requirements should satisfy.

In this paper, we contribute:

(*i*) a scientific, systematic overview of formal requirements capture and analysis tools in the literature;
(*ii*) a summary of the various tools' properties and capabilities; and
(*iii*) identification of current trends and open research questions in formal requirements engineering.

[1]Types of requirements include system requirements, High Level Requirements (HLRs), Low Level Requirements (LLRs), derived requirements, software requirements, safety-related requirements, airworthiness requirements, inter-version compatibility requirements, functional requirements, operational requirements, timing requirements, partitioning requirements, failure detection requirements, and safety monitoring requirements.

FormaliSE '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0589-2/24/04
https://doi.org/10.1145/3644033.3644373

2 PRELIMINARIES

We introduce some relevant concepts.

Definition 1. *A software or system **requirement** describes an intended behavior or property of the software or system. Requirements are expressed in natural language but can be formalized into an unambiguous statement (called a **formal requirement** or **specification**).*

Traditionally, formal requirements are stated in (potentially a fragment of) a formal logic such as *first order logic* or *linear temporal logic* (LTL). More details about LTL can be found elsewhere [49]. The process of gathering requirements, formalizing them, and performing analyses is called *requirements capture and analysis*. A set of requirements can be analyzed for various desired properties. These properties are described below—this discussion is purposefully kept informal, as precise definitions vary across tools.

- A requirement is *atomic* if it describes a single, indivisible aspect of the system.
- *Consistency* means that the set of requirements does not contain logical contradictions (i.e., the set of requirements is satisfiable).
- *Realizability* means that the set of requirements describes a system that is actually implementable. More concretely, this is true if and only if for all possible sets of input values, there exists some set of output values that satisfies all requirements. (Both realizability and consistency are types of *conflict analysis*.)
- *Completeness* means that the requirements specify all relevant system behaviors.
- *Correctness* is the slippery notion that a requirement captures its intended meaning, often involving interactions with stakeholders.
- *Independence* means that no requirement is implied by other requirements (i.e., the requirements are free of *redundancy*).
- *Contingency* means that each requirement is satisfiable and falsifiable.
- A requirement demonstrates *vacuity* if part of the requirement definition has no bearing on whether or not the requirement is satisfied.

To illustrate the types of requirements defects that formal methods tools seek to detect, we give examples of how the above properties intuitively apply to a set of natural-language requirements. We consider the use case of a door to an initially empty room with a counter that increments when a person enters the room and decrements when a person leaves the room. Entrance and exit sensors detect the actions and update the counter. The stakeholders need the counter to correctly reflect the number of people in the room at any given time. For this use case, we present example requirements to show how each of the aforementioned properties can be violated:

- Consider the requirement:
  - **R0** The entry sensor shall increase the counter by one for each detected entry and decrease the counter by one for each detected exit.

  The requirement is *not atomic* because it can be divided into two functionalities: incrementing for entrances and decrementing for exits.
- Consider the assumption and two requirements:
  - **A0** At least one person enters the room.
  - **R1** The entry sensor shall increase the counter by one for each detected entry.
  - **R2** The entry sensor shall decrease the counter by one for each detected entry.

  The set is *inconsistent* because the entry sensor cannot both increase and decrease the counter at the same time for each detected entry.
- Consider the two requirements:
  - **R2** The entry sensor shall decrease the counter by one for each detected entry.
  - **R3** At any time, the counter shall correctly reflect the total number of people who have entered the room.

  The set is *consistent* because it possible to simultaneously satisfy both requirements–specifically, both are satisfied if nobody ever enters the room. However, the set is *not realizable* because *if* at least one person enters the room, then **R2** and **R3** cannot both be satisfied.
- Consider the two requirements:
  - **R3** At any time, the counter shall correctly reflect the total number of people who have entered the room.
  - **R4** The entry sensor shall increase the counter by one for each detected entry.

  The set is *incomplete* because important behavior such as recording the exit of people from the room have not been specified.
- Consider the requirement:
  - **R3** At any time, the counter shall correctly reflect the total number of people who have entered the room.

  R3 is *incorrect* because it does not capture the intended behavior of the stakeholders where the stakeholders want the counter to always show the exact number of people inside the room.
- Consider the two requirements:
  - **R1** The entry sensor shall increase the counter by one for each detected entry.
  - **R5** The entry sensor shall increase the counter by 2^0 for each detected entry.

  The set is *not independent* because **R1** implies **R5**.
- Consider the requirement:
  - **R6** The exit sensor shall decrease the counter by one for each detected exit.

  R6 is *contingent* because there exist implementations that satisfy and falsify the requirement. For example, an implementation that erroneously *increases* the counter at each detected exit falsifies the requirement. Additionally, an implementation that correctly decreases the counter by one for each detected exit satisfies the requirement.
- Consider the requirements:
  - **R7** At any time, if the number of people who have exited the room is greater than the number of people who have entered the room, then the counter shall correctly reflect the total number of people who are in the room.
  - **R8** The room shall be initially empty.

  R7 is *vacuously true* because the room is initially empty (from **R8**)—it can never be the case that the number of people who exited is greater than the number of people who

entered. Therefore, the consequent of the implication ("the counter shall correctly...") has no impact on whether or not the requirement is satisfied.

3 METHODOLOGY

We performed a systematic literature review through a combination of (*i*) a manual search and (*ii*) an automated search of digital libraries.

The manual search began with manual identification of relevant papers through prior domain knowledge. Using Google Scholar, we iteratively expanded the set of papers we surveyed by manually identifying relevant citations of the current set (as well as relevant papers who cite papers in the current set) until reaching a fixpoint. This yielded 25 papers (representing 18 tools) as a benchmark set.

Based on an analysis of the benchmark set, we constructed the following search string:

```
"((“requirements engineering”
      OR “requirements analysis”)
AND (“formal analysis”
      OR “formal verification”
      OR “formal validation”
      OR “formally verify”
      OR “formal method*”)
AND (“tool” OR “tools” OR “framework*”)
AND ((“consisten*” AND (“complete*” OR “vacuity”))
      OR “realizability”))".
```

Additionally, the searches were filtered for freely available conference papers and journal articles in English that were published from 2005 on. IEEE Xplore, ACM Digital Library, and Springer Link returned 961, 435, and 900 papers (respectively) for a total of 2,296 papers. We achieved a sensitivity of $\geq$ 80% with respect to the benchmark set, surpassing the threshold prescribed by [108][2].

After manual review of the papers generated from the automated search, we identified 44 relevant tools. Combining with the tools from the manual search, we found a total of 59 distinct tools.

In order to be classified as "relevant," papers must demonstrate an *implemented tool* that performs *formal analysis* of requirements. More specifically, we omitted tools that were purely dedicated to requirements capture without any analysis (e.g., [74]), tools that were clearly designed to be synthesis tools rather than requirements analysis tools (e.g., [66]), tools that only performed non-formal analyses (e.g., using natural language processing (NLP) for inconsistency detection), and conceptions of algorithms or approaches that were not yet implemented.

4 RESULTS

We provide an overview of the requirements capture and analysis tools from our literature search. First, we present general discussions of all the tools by categorization, where we highlight a variety of general technical approaches. We then present a table summarizing each tool's properties, followed by a more detailed discussion of a select few interesting tools, and finally an overview of research trends and future directions.

4.1 Tool Descriptions

The largest group of tools are *instrumentations of general-purpose model checkers* [1, 4, 6–8, 11, 15, 18, 29, 31, 34, 38, 39, 43, 44, 53, 68, 72, 73, 80, 83, 93, 94, 99, 106]. These tools translate a set of input requirements to the backend model checker's specification language and construct various model checking queries to check the desired properties. For example, consider a set of n specifications $\{\varphi_i \mid i \in \mathbb{N}\}$. A simple consistency check would invoke the model checking query $\neg(\varphi_1 \wedge \cdots \wedge \varphi_n)$ against a *universal model* that satisfies all traces. A counterexample demonstrates that the specifications are consistent, and they are inconsistent otherwise.

Another large group of tools are *synthesis tools with debugging support for requirements analysis* [27, 42, 62, 65, 76, 104]. They are based on the synthesis of a reactive system given requirements formulated in LTL, often involving fragments with tractable algorithms such as GR(1) [81] and GXW [19], as well as translations to Büchi automata. The success (resp., failure) of the synthesis algorithm directly corresponds to the realizability (resp., unrealizability) of the specification. Realizability analysis is useful because it can reveal hidden environmental assumptions. Note that realizability is a stronger property than consistency, but the two only differ when the system being modeled takes input from an environment that controls some of the variables. The tools have a variety of ways to debug unrealizable specifications— for example, a *counterstrategy* describes the environment's strategy to create situations where at least one requirement must be falsified.

Another popular approach to capturing requirements involves the class of *ontology-based tools* [2, 5, 23, 24, 47, 70, 71, 75, 85, 91, 92]. These tools fundamentally differ from the previously discussed tools as they are based on a *description logic* rather than a *temporal logic*. Description logics focus on types of entities and relationships between them, rather than on an evolution of state over time. Description logics have some drawbacks, mainly that (*i*) they lack temporal richness, and (*ii*) they rely on manually predefined ontologies which represent knowledge domains. But, description logics also have benefits. First, they allow analyses of language-related concepts like ambiguity, noise, and latent domain knowledge. Second, the analyses are relatively scalable and do not encounter state space explosion as is possible in model checking. Third, there often exists a natural translation from boilerplate natural language input to description logic.

Another group is comprised of *logic programming tools* which rely on formalisms like ASP and languages like Datalog [41, 46, 59, 72, 98]. An example tool was developed by Hall et al. [46], which uses goal-directed s(CASP) as a reasoning engine. They use the event calculus formalism [89] to perform commonsense reasoning of failure scenarios in dynamic sytems by assuming dense real-valued domains for time and other continuous physical properties. They utilize capabilities of constraint-solving over reals CLP($\mathcal{R}$) [50] available in s(CASP) to model a dense time domain. A unique aspect of this tool is that it can test *adequacy* of requirements by encoding known potential failure scenarios such as *single event upsets* to the logic program and querying the system for consistency using

[2]To calculate sensitivity, we divided the number of tools present in the benchmark set that were found by the automated search by the the total number of tools in the benchmark set *that were indexed in at least one of the libraries*. Two tools were not indexed, and incorporating them into the sensitivity calculation still yields $\geq$ 70%.

abductive reasoning. They utilize domain knowledge for a more comprehensive consistency check. However, the event calculus formalism does not use temporal operators from LTL.

Some tools take a *refinement-based approach* with the *Event-B* formalism [90, 96, 101]. A benefit of this approach is that requirements can be decomposed and details can be added as the development process matures. With refinement, the user can check if requirement decompositions are consistent and complete (both within a requirement decomposition and across requirement decompositions).

A few tools are characterized by *unrestricted natural language input* [35–37, 80]. In addition to formal reasoning, these tools involve sophisticated NLP in the front end to produce the underlying formalism. This involves clear trade-offs– unrestricted natural language increases the scope of formalizable requirements and maximizes user-friendliness (especially in a setting without formal methods experts), but may lead to an increase in analysis errors due to incorrect generated formalisms.

Another strategy for user-friendliness involves *UML input,* benefiting from developers' previous experience with UML and allowing developers to easily apply the tool to existing projects [7, 15, 34, 97, 100, 106]. While some tools use features already present in UML, some tools introduce an external constraint language to formally express requirements.

We encountered several tools that utilized formal representations for test generation [51, 61, 84, 103]. While these tools are somewhat formal, they do not perform formal analysis of requirements, so we will not discuss them in depth.

Finally, a wide range of tools do not cleanly fit into any of the above categories, as they employ *unique or custom backend reasoning engines* [3, 14, 26, 28, 48, 55, 56, 63, 78, 87, 88, 95, 100, 102]. These tools reduce requirements analysis to, for example, graph algorithms [55], abstract state machines (ASMs) [87], custom tableau algorithms [26, 35], new formalisms [14], custom heuristics [100], or database queries [71, 78]. Addionally, some tools [10, 12, 54] involve *multiple* reasoning engines to analyze various properties.

4.2 Notable Tools

Many of the tools found in the literature review are research prototypes that are no longer actively developed or used. In this section, we highlight a few tools that are actively maintained and regularly applied to real-world problems.

4.2.1 ARSENAL. The ARSENAL (Automatic Requirements Specification Extraction from Natural Language) framework [36, 37], developed at SRI International and TTTech, uses a combination of natural language processing (NLP) and formal methods (FM) to map natural language requirements to formal representations. This framework consists of two main stages, the NLP stage and the FM stage. Core steps in the NLP stage involve using the Standard Typed Dependency Parser [25] to detect semantic dependencies in the text and convert the dependencies into a tabular, symbolic intermediate representation (IR) that outlines properties of terms and relationships between terms. The IR table is used to generate a formal model that can be used by formal methods tools. First, the IR table is translated into a SAL [9] model, which is a Kripke structure-based model that can be passed as input to the SAL model checker. Model checking techniques are used to check the satisfiability of the model, which corresponds to a requirement consistency check. Second, ARSENAL uses the IR table to generate logical specifications in a subclass of LTL called Generalized Reactivity (1), or GR(1), which is known to be expressive but still amenable to synthesis [81]. *Synthesis* techniques are algorithms to automatically generate an implementation satisfying a given specification, if possible. The success (resp., failure) of the synthesis algorithm directly corresponds to the requirements being realizable (resp., unrealizable). If the specification is unrealizable, then ARSENAL suggests candidate environmental assumptions that would make the specification realizable.

4.2.2 ASSERT. The ASSERT (Analysis of Semantic Specifications and Efficient generation of Requirements-based Tests) tool [23, 24, 58, 67, 70, 91] was developed by General Electric to assist with formal requirements capture and analysis for aviation software. The input language, called the SADL requirements language (SRL) [23], is an extension of the Semantic Application Design Language (SADL) [22] and was designed specifically for requirements capture and analysis in ASSERT. SADL an ontology language based on set theory and first order logic that unambiguously encodes information about the model in question in terms of classes (sets of entities) and properties (relationships between classes). The analysis engine, called Requirements Analysis Engine (RAE) [64], is built on top of the ACL2s theorem prover [16] and can reason about contingency, independence, conflict, and completeness of requirements. Additionally, ASSERT performs requirements-based test case generation by systematically generating tests that satisfy each requirement, following DO-178C guidelines.

4.2.3 FRET. FRET (Formal Requirements Elicitation Tool) [38, 39, 53] is a requirements elicitation tool developed at NASA that focuses on the realizability of specifications of reactive systems. The tool's inputs are requirements formed in a structured natural language called FRETish [40]. First, each FRETish requirement is translated into two representations, which are past- and future-time metric temporal logic (MTL) formulae (MTL is similar to LTL, but uses *bounded* temporal operators). The set of past-time MTL requirements are translated into the synchronous dataflow language Lustre [45], and then the Lustre model is then fed into the backend reasoning engines JKind [30] and Kind 2 [17] for realizability analysis. On the other hand, the future-time MTL requirements are formulated in a NuSMV [21] model, which is analyzed by the NuSMV model checker. A distinguishing feature of FRET is its front-end GUI, which helps the user debug realizability issues by interacting with counterexample traces and visualizing relationships between requirements.

4.2.4 SpeAR. SpeAR (Specification and Analysis of Requirements) [29, 44] was developed by Collins Aerospace and AFRL to assist in developing and analyzing system requirements. The input language is a structured natural language with the semantics of past-time LTL (LTL with only past-time temporal operators). The input language has a natural translation to Lustre, which is fed into the JKind model checker for backend analysis. Lustre models are hierarchically structured, where a top-level component contains system-level inputs, outputs, and properties, and the behavior of the system is

defined in lower-level components. JKind checks completeness by performing compositional verification, which determines whether the conjunction of all component-level requirements logically entails each system-level requirement. JKind also checks consistency in terms of a satisfiability query on component-level requirements.

4.2.5 Spectra. Spectra [65] is a specification language developed by academic researchers at Tel Aviv University and the University of Leicester that includes analysis tools (called Spectra tools) for requirements analysis and controller synthesis. The Spectra language is similar to Lustre [45], as it unambiguously defines the behavior of reactive systems. Due to the input language's syntax, there is a straightforward translation to the GR(1) fragment of LTL, which is amenable to synthesis [81]. Spectra tools applies the synthesis algorithm, along with some custom heuristics, to determine the realizability of the specification. If the specification is unrealizable, Spectra computes an *unrealizable core* that contains a minimal set of conflicting requirements. Additionally, Spectra presents a *counterstrategy*, which is a description of how the environment can force requirement violations, as well as a list of candidate assumptions to make the specification realizable. Further, Spectra detects when a specification can be trivially satisfied (i.e., when environmental assumptions can be forcibly violated) as well as when requirements contain vacuous expressions.

4.3 Tool Comparison

In Table 1, we present some details about each tool discovered in our search. In particular, we outline each tool's name and references (column 1), information about the tools' front- and back-end strategies (columns 2, 3, 4, 6, 8), whether each tool is open source (column 5), and whether or not each tool has been used in industry or in some large case study (column 7). Information about industrial use and/or case studies can be found from the references listed in column 1.

4.4 Current Trends and Future Directions

Notably, very few tools (only ~5% from Table 1) use an unrestricted natural language input, while the rest require a structured input. Many papers express the concern that natural language input can lead to incorrect formalizations due to the wide range of potential unrestricted natural language inputs (e.g., [11, 23, 38]). While this is a legitimate concern, there is an overlooked dual concern that either (i) a structured input will be formulated incorrectly by the user, (ii) only a subset of requirements will be fit into the defined structure, or (iii) the constrained structure will dissuade system engineers from using the tool altogether. In NLP, there is a trend of using large language models (LLMs) to assist with translation from NL to a formal representation (e.g., [60, 105]). LLMs are a good match for such translations because they are strong at deconstructing underlying semantic structure, and they have vast general-purpose knowledge that does not require the user to supply a large training data set (which is often not available in many domains, especially if the formalism is niche). This is because LLMs can perform various tasks using purely in-context learning, which only requires the user to supply a handful of examples. Additionally, LLMs' chatbot functionality can assist with the formalization process through various interactive steps. Using modern NLP strategies is thus a promising area for future development of formal requirements capture and analysis tools. On a related note, none of the papers included a scientific study of usability outside of intuitive, informal claims. In industry, tools requiring front-end formalization can get push-back from non-formal methods experts, even if they seem "usable" by formal methods experts and have been validated by a case study. For tools with structured inputs, rigorous study about the proportion of requirements that can be naturally formulated, as well as comprehensive feedback from requirements engineers without knowledge of formal methods, is desired.

Similarly, very few tools give a detailed discussion of scalability. (A notable exception is Eddy [92], where an in-depth scalability analysis demonstrated acceptable performance, even as the number of requirements grew beyond the number tested in their main case study.) In an industrial setting, projects often have hundreds or thousands of requirements (that are often difficult to formalize), and therefore scalability is of supreme importance to practitioners. Establishing a set of large-scale, industry-relevant benchmarks could help users compare different tools.

Model checking and synthesis algorithms based on variants of FOL and LTL dominate back-end reasoning techniques (over 60% of tools from Table 1). Description logics are less popular but still notable due to their natural mapping to various problem domains. However, none of these formalisms can analyze requirements of systems that involve (i) bounded-time temporal operators over arbitrary points in time and (ii) span both discrete and continuous-time domains—a concrete obstacle is the computational complexity associated with such formalisms. Relevant formalisms include *metric temporal logic* (MTL, explored in [7, 38, 39, 53]), *signal temporal logic*, and *hybrid timed automata* . More work could be done to identify (potentially domain-specific) fragments and algorithms to expand the usage of these formalisms.

Lastly, although we gave an informal overview of analysis properties (e.g. consistency) in Section 2, formal definitions of such terms differ greatly between tools. For the purposes of this study, we take claims about the analysis capabilities of the various tools at face value (e.g., if a paper claims that the implemented tool supports consistency analysis, we have reported as such in Table 1 regardless of their specific definition of "consistency.") Moreover, many papers confuse "verification" and "validation", or use "consistency" in informal, confusing ways. The community should be careful to consult multiple sources for terminology and stick to standard usages.

Threats to Validity. Despite our best efforts to be comprehensive, our search realistically missed some tools. But, we argue that our automated search of multiple digital libraries led to a relatively large and representative set.

Additionally, it is possible that there exist tool capabilities that were not reflected in our analysis due to a lack of adequate description or publicly available information.

5 RELATED WORK

We briefly mention some related surveys of formal methods and requirements engineering. Studies by Jones et al. [52] and Pandey and Batra [79] are similar in spirit. However, [52] is now over 25 years old, and a fresh look is useful. Similarly, [79] is over 10 years old, and they also focus on general approaches rather than specific

Tool	Input language	Reasoning Engine	Analysis properties	Open source?	Formalism	Industry use or large case study	Notable features
AGREE [31]	AADL + Lustre	JKind	Consistency, realizability	Yes	Past LTL	Yes	Compositional reasoning
Alrajeh et al. [1]	FOLTL	Model checker, inductive logic programming (ILP) learner	Completeness	No	Fluent Linear Temporal Logic (FLTL)	Yes	Obstacle analysis
Arcaini et al. [4]	CTL	NuSMV	Consistency, completeness, minimality	Yes	CTL	Yes	NA
ARSENAL [36, 37]	Natural language (NL)	SAL	Consistency, realizability	No	GR(1)	Yes	Candidate assumption generation
AsmetaRE [87]	Structured NL	ASMETA	Consistency	Yes	ASMs	Yes	Simulation
ASSERT [23, 24, 70, 91]	Structured NL	ACL2s	Contingency, independence, conflict, completeness	No	Set theory, FOL	Yes	Test generation
Avdeenko & Pustovalova [5]; Garanina & Borovikova [33]	Structured NL	Hermit	Completeness, consistency	Yes	OWL, LTL	No	NA
Barnat et al. [8]	LTL	DiVinE	Consistency, redundancy, completeness, vacuity	No	LTL	Yes	Suggest missing requirements
BTC Embedded-Platform [11]	Structured NL	Model checker	Consistency, vacuity, completeness	No	Simplified universal pattern [11]	Yes	Test generation, GUI requirement visualization
$\mathcal{B}$-Tropos [72]	Graph-based (graphical)	Model checkers, $\mathcal{S}$CIFF abductive proof procedure	Consistency	No	LTL, $\mathcal{S}$CIFF (logic programming)	Yes	Property checking
CAMEmb-Modeler [97]	UML	VDMTools	NA	No	VDM++	No	Context boundary analysis
CARL [35]	NL	Custom tableau-based algorithm	Consistency	No	Nonmonotonic propositional logic (PL) with predicates	Yes	Scenario consistency, implied facts of requirements
CDET [88]	FOL	Custom DAG-based algorithms	Consistency	No	FOL	No	Domain-specific repair suggestions
CHASE [76]	Structured NL, NL	TuLiP	Correctness, completeness, consistency, realizability	Yes	GR(1)	Yes	Reasons over time intervals, simulation

Table 1: Comparison of tools discovered in our search.

Tool	Input language	Reasoning Engine	Analysis properties	Open source?	Formalism	Industry use or large case study	Notable features
Chen et al. [18]	SafeNL (structured NL)	MyCCSL	Consistency	No	Clock constraint specification language	Yes	NA
CLEAR [10]	Structured NL	SMT solvers, model checkers, Acacia	Consistency, completeness, redundancy, correctness, realizability	No	FOL, LTL	Yes	Test generation
Concern Model Verifier [47]	OWL	Jena	Consistency, completeness	No	OWL	Yes	NA
DECIMAL [78]	Tables + FOL	SQL database	Completeness, consistency	No	FOL	Yes	Support for product line specifications
Degiovanni et al. [26]	LTL	Custom algorithm	Consistency, realizability	Yes	LTL Tableau, string counting	Yes	Conflict likelihood estimation
EARS-CTRL [62]	Structured NL	autoCode4 [20]	Correctness, realizability	Yes	GXW	No	NA
Eddy [92]	Pseudo SQL	HermiT	Consistency	No	OWL	Yes	Demonstrated scalability, privacy policy analysis
EuRailCheck [15]	UML + Structured NL	NuSMV	Consistency, property checking	No	Fragment of FOLTL with RE	Yes	Simulation
Traichaiyaporn & Aoki, Wakrime et al. [96, 101]	Event-B	Rodin	Completeness, consistency	Yes	Event-B	Yes	Refinement analysis
FRET [38, 39, 53]	Structured NL	Kind 2 [17], JKind[30], NuSMV[21]	Consistency, realizability	Yes	past or future MTL	Yes	Conflict visualization, simulation
Garis et al. [34]	UML	Alloy	Consistency	Yes	FOL	No	Alloy analyzer graphical editing
Greenyer et al. [42]	Modal sequence diagrams	Custom synthesis algorithm	Consistency, realizability	No	Büchi automata	Yes	Simulation, incremental synthesis
Greenyer, Sharifloo et al. [43]	MSDs	NuSMV	Consistency	No	CTL	Yes	Support for product line specifications
Hall et al. [46]	Structured NL	s(CASP) solver	Consistency, adequacy	No	Event Calculus, ASP	Yes	NA
K.M. et al. [2]	NL	HermiT	NA	No	SWRL	Yes	NL requirement quality analysis, latent domain knowledge discovery

Table 1: Comparison of tools discovered in our search (continued).

Tool	Input language	Reasoning Engine	Analysis properties	Open source?	Formalism	Industry use or large case study	Notable features
Lauenroth & Pohl [56]	OVM + DRS	Custom model checking algorithms	Consistency	No	VOM, DRS, PL, FSMs	Yes	Product line analysis
Li et al. [59]	Structured NL	clingcon	Consistency	No	ASP	Yes	NA
MADES [7]	UML + MARTE	Zot	Consistency, correctness	Yes	MTL	Yes	Simulation, property checking
MBIPV [106]	UML	NuSMV	Privacy violations	Yes	Kripke structures + CTL	Yes	Privacy violation analysis
Miller et al. [68]	RSML^{-e}, CTL, PVS	NuSMV, PVS	Consistency, completeness	No	RSML^{-e}, CTL, PVS	Yes	Simulation, property checking
MIRA [94]	Structured NL	SMT solvers, model checkers	Consistency, completeness	Yes	MSCs	Yes	Simulation, test generation
Mokos et al. [71]	Structured NL	SPARQL	Completeness, consistency	No	OWL	Yes	Measure noise & ambiguity, domain knowledge, extract tacit knowledge, property checking
NAT2TEST [14]	Structured NL	Custom algorithms	Consistency, completeness, reachability, time lock	No	DFRS, CSP	Yes	Test generation
Pi et al. [80]	NL	Model checker	Consistency	No	LTL	Yes	Hierarchical heuristic checking
Prema [48]	Pseudocode	Z3	Consistency	No	State machines	Yes	Simulation, Test generation, Requirement visualization
RATSY [12, 54]	Büchi automata	NuSMV, CUDD, Anzu	Consistency, realizability, correctness	Yes	PSL	Yes	Trace manipulation, counterstrategies, countertraces, simulation, interactive debugging game
REInDetector [75]	Structured NL	Pellet	Consistency	No	OWL	Yes	Inconsistency explanation
ReqV [99]	Structured NL	Aalta, NuSMV	Consistency	No	LTL	Yes	Minimal subset of conflicting requirements
ReSA [63]	Structured NL	Z3	Consistency	Yes	PL	Yes	NA
Runde & Fay [85]	Not specified	Jess	Consistency, redundancy	No	OWL	Yes	NA
Sikora et al. [90]	Message Sequence Charts (MSCs)	Custom algorithms	Consistency	No	Interface automata	No	Refinement checking

Table 1: Comparison of tools discovered in our search (continued).

Tool	Input language	Reasoning Engine	Analysis properties	Open source?	Formalism	Industry use or large case study	Notable features
Slugs [27]	LTL	Custom GR(1) synthesis tool	Consistency, realizability, non-well-separation	Yes	GR(1)	No	Counterstrategy, Assumption weakening, simulation
SpeAR v2.0 [29, 44]	Structured NL	JKind	Completeness, consistency	Yes	Past LTL	Yes	Compositional reasoning
SpecCC [104]	Structured NL	G4LTL	Consistency, realizability	Yes	LTL	Yes	NA
SpecPro [73]	Structured NL	POLSAT	Consistency	Yes	LTL	Yes	Fault injection modeling, minimal inconsistent subsets
SpecScribe [83]	Structured NL	SAL	Consistency, completeness	No	LTL	No	Property checking
Spectra [65]	Reactive systems language	Custom GR(1) synthesis tool	Consistency, realizability, vacuity, non-well-separation	Yes	GR(1)	Yes	Unrealizable cores, counterstrategies, candidate assumptions
Sukumaran et al. [93]	UML	SAL	Correctness, completeness, consistency	No	SAL language	Yes	Scenario generation, simulation
Thyssen & Hummel [95]	RE tables	Custom DFA algorithms	Completeness, consistency	No	DFA	Yes	Conflict resolution tables
TURTLE-P [3]	MSCs	Not specified	Consistency	No	Not specified	Yes	Property checking
Ultimate Req-Analyzer [55]	Structured NL	Ultimate Automizer	Consistency, rt-consistency[55], vacuity	Yes	PEA, graphs	Yes	Small subsets of troubled requirements
VARED [6]	NL	SMV, InVeriant	Consistency, vacuity	No	LTL, linear hybrid automata	No	Tool support for building state model
Wahler et al. [100]	UML/OCL	Custom heuristics	Consistency	No	FOL	Yes	Scalable, polynomial-time algorithms
Walter et al. [102]	Structured NL	Custom algorithm	Redundancy	No	SPS, LTL, FOL	Yes	NA
ST-Tool [41]	Graphical	ASSAT, Cmodels, DLV, Smodels	Consistency, correctness	No	Datalog	Yes	Security property checking, GUI

Table 1: Comparison of tools discovered in our search (continued).

requirements analysis tools. We note that [79] is a nice complement to this paper, as it discusses approaches (e.g. Z-method, B-method) that are not as emphasized in this paper. Additionally, [69] and [107] are more recent, but are domain-specific. For example, [69] focuses on formal methods in requirements engineering specifically for security, which has its own additional set of challenges, while [107] centers on industrial cyber-physical systems. Moreover, [107] focuses more on semi-formal methods (e.g., requirements management and adoption of industrial standards), which are outside the scope of this paper.

6 CONCLUSION

We have presented a variety of formal methods tools from the literature for requirements capture and analysis. They exhibit considerable variation in terms of input language and analysis capabilities,

and many tools have been applied in large-scale case studies. The use of unrestricted NL with new AI-based front-ends is an active frontier of research and is a green area for new progress in terms of tool usability. Additionally, the use of richer formalisms (e.g., MTL and STL) that allow the specification and analysis of richer properties is an active area of work. Finally, there is still much room to rigorously and scientifically study tool usability and scalability such that greater benefits can be reaped in an industrial setting.

Acknowledgements & Disclaimer. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] Dalal Alrajeh, Jeff Kramer, Axel Van Lamsweerde, Alessandra Russo, and Sebastián Uchitel. 2012. Generating obstacle conditions for requirements completeness. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 705–715.

[2] KM Annervaz, Vikrant Kaulgud, Shubhashis Sengupta, and Milind Savagaonkar. 2013. Natural language requirements quality analysis based on business domain models. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 676–681.

[3] Ludovic Apvrille, Pierre de Saqui-Sannes, and Ferhat Khendek. 2006. TURTLE-P: a UML profile for the formal validation of critical and distributed systems. *Software & Systems Modeling* 5 (2006), 449–466.

[4] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. 2011. A model advisor for NuSMV specifications. *Innovations in systems and software engineering* 7 (2011), 97–107.

[5] Tatiana Avdeenko and Natalia Pustovalova. 2015. The ontology-based approach to support the completeness and consistency of the requirements specification. In *2015 International Siberian Conference on Control and Communications (SIBCON)*. IEEE, 1–4.

[6] Julia Badger, David Throop, and Charles Claunch. 2014. Vared: Verification and Analysis of Requirements and Early Designs. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 325–326.

[7] Luciano Baresi, Gundula Blohm, Dimitrios S Kolovos, Nicholas Matragkas, Alfredo Motta, Richard F Paige, Alek Radjenovic, and Matteo Rossi. 2015. Formal verification and validation of embedded systems: the UML-based MADES approach. *Software & Systems Modeling* 14 (2015), 343–363.

[8] Jiri Barnat, Jan Beran, Lubos Brim, Tomas Kratochvíla, and Petr Ročkai. 2012. Tool chain to support automated formal verification of avionics simulink designs. In *Formal Methods for Industrial Critical Systems: 17th International Workshop, FMICS 2012, Paris, France, August 27-28, 2012. Proceedings 17*. Springer, 78–92.

[9] Saddek Bensalem, Vijay Ganesh, Yassine Lakhnech, César Munoz, Sam Owre, Harald Rueß, John Rushby, Vlad Rusu, Hassen Saiedi, and Natarajan Shankar. 2000. An overview of SAL. In *Lfm2000: Fifth NASA Langley Formal Methods Workshop*.

[10] Devesh Bhatt, Anitha Murugesan, Brendan Hall, Hao Ren, and Yogananda Jeppu. 2018. The CLEAR way to transparent formal methods. In *4th Workshop on Formal Integrated Development Environment*.

[11] Tom Bienmüller, Tino Teige, Andreas Eggers, and Matthias Stasch. 2016. Modeling requirements for quantitative consistency analysis and automatic test case generation. In *Workshop on Formal and Model-Driven Techniques for Developing Trustworthy Systems at 18th International Conference on Formal Engineering Methods*.

[12] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. 2010. RATSY–a new requirements analysis tool with synthesis. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 425–429.

[13] Barry W Boehm. 1984. Software engineering economics. *IEEE transactions on Software Engineering* 1 (1984), 4–21.

[14] Gustavo Carvalho, Ana Cavalcanti, and Augusto Sampaio. 2016. Modelling timed reactive systems from natural-language requirements. *Formal Aspects of Computing* 28 (2016), 725–765.

[15] Roberto Cavada, Alessandro Cimatti, Alessandro Mariotti, Cristian Mattarei, Andrea Micheli, Sergio Mover, Marco Pensallorto, Marco Roveri, Angelo Susi, and Stefano Tonetta. 2009. Supporting requirements validation: The EuRailCheck tool. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 665–667.

[16] Harsh Raju Chamarthi, Peter Dillinger, Panagiotis Manolios, and Daron Vroon. 2011. The ACL2 Sedan theorem proving system. In *Tools and Algorithms for the Construction and Analysis of Systems: 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26–April 3, 2011. Proceedings 17*. Springer, 291–295.

[17] Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. 2016. The Kind 2 model checker. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*. Springer, 510–517.

[18] Xiaohong Chen, Zhiwei Zhong, Zhi Jin, Min Zhang, Tong Li, Xiang Chen, and Tingliang Zhou. 2019. Automating consistency verification of safety requirements for railway interlocking systems. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 308–318.

[19] Chih-Hong Cheng, Yassine Hamza, and Harald Ruess. 2016. Structural synthesis for GXW specifications. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*. Springer, 95–117.

[20] Chih-Hong Cheng, Edward A Lee, and Harald Ruess. 2017. autoCode4: structural controller synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I 23*. Springer, 398–404.

[21] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer, 359–364.

[22] Andrew Crapo and Abha Moitra. 2013. Toward a unified English-like representation of semantic models, data, and graph patterns for subject matter experts. *International Journal of Semantic Computing* 7, 03 (2013), 215–236.

[23] Andrew Crapo, Abha Moitra, Craig McMillan, and Daniel Russell. 2017. Requirements capture and analysis in ASSERT (TM). In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 283–291.

[24] Andrew W Crapo and Abha Moitra. 2019. Using OWL ontologies as a domain-specific language for capturing requirements for formal analysis and test case generation. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. IEEE, 361–366.

[25] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses.. In *Lrec*, Vol. 6. 449–454.

[26] Renzo Degiovanni, Pablo Castro, Marcelo Arroyo, Marcelo Ruiz, Nazareno Aguirre, and Marcelo Frias. 2018. Goal-conflict likelihood assessment based on model counting. In *Proceedings of the 40th International Conference on Software Engineering*. 1125–1135.

[27] Rüdiger Ehlers and Vasumathi Raman. 2016. Slugs: Extensible gr (1) synthesis. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28*. Springer, 333–339.

[28] Peter H Feiler, Bruce A Lewis, and Steve Vestal. 2006. The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. In *2006 IEEE conference on computer aided control system design, 2006 IEEE international conference on control applications, 2006 IEEE international symposium on intelligent control*. IEEE, 1206–1211.

[29] Aaron W Fifarek, Lucas G Wagner, Jonathan A Hoffman, Benjamin D Rodes, M Anthony Aiello, and Jennifer A Davis. 2017. SpeAR v2. 0: Formalized past LTL specification and analysis of requirements. In *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings 9*. Springer, 420–426.

[30] Andrew Gacek, John Backes, Mike Whalen, Lucas Wagner, and Elaheh Ghassabani. 2018. The JKind model checker. In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II 30*. Springer, 20–27.

[31] Andrew Gacek, Andreas Katis, Michael W Whalen, John Backes, and Darren Cofer. 2015. Towards realizability checking of contracts using theories. In *NASA Formal Methods: 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings 7*. Springer, 173–187.

[32] Daniel Galin. 2004. *Software quality assurance: from theory to implementation*. Pearson education.

[33] Natalia Garanina and Olesya Borovikova. 2019. Ontological approach to checking event consistency for a set of temporal requirements. In *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*. IEEE, 0922–0927.

[34] Ana Garis, Ana CR Paiva, Alcino Cunha, and Daniel Riesco. 2012. Specifying UML protocol state machines in alloy. In *Integrated Formal Methods: 9th International Conference, IFM 2012, Pisa, Italy, June 18-21, 2012. Proceedings 9*. Springer, 312–326.

[35] Vincenzo Gervasi and Didar Zowghi. 2005. Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 14, 3 (2005), 277–330.

[36] Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. 2014. Automatically extracting requirements specifications from natural language. *arXiv preprint arXiv:1403.3142* (2014).

[37] Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. 2016. ARSENAL: automatic requirements specification extraction from natural language. In *NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings 8*. Springer, 41–46.

[38] Dimitra Giannakopoulou, Anastasia Mavridou, Julian Rhein, Thomas Pressburger, Johann Schumann, and Nija Shi. 2020. Formal requirements elicitation with FRET. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020)*.

[39] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. 2021. Automated formalization of structured natural language requirements. *Information and Software Technology* 137 (2021), 106590.

[40] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. 2021. Automated formalization of structured natural language requirements. *Information and Software Technology* 137 (2021), 106590.

[41] Paolo Giorgini, Fabio Massacci, John Mylopoulos, Alberto Siena, and Nicola Zannone. 2005. ST-Tool: A CASE tool for modeling and analyzing trust requirements. In *Trust Management: Third International Conference, iTrust 2005, Paris, France, May 23-26, 2005. Proceedings 3*. Springer, 415–419.

[42] Joel Greenyer, Christian Brenner, Maxime Cordy, Patrick Heymans, and Erika Gressi. 2013. Incrementally synthesizing controllers from scenario-based product line specifications. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 433–443.

[43] Joel Greenyer, Amir Molzam Sharifloo, Maxime Cordy, and Patrick Heymans. 2013. Features meet scenarios: modeling and consistency-checking scenario-based product line specifications. *Requirements Engineering* 18 (2013), 175–198.

[44] Kerianne H Gross, Aaron W Fifarek, and Jonathan A Hoffman. 2016. Incremental Formal Methods Based Design Approach Demonstrated on a Coupled Tanks Control System. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 181–188.

[45] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. 1991. The synchronous data flow programming language LUSTRE. *Proc. IEEE* 79, 9 (1991), 1305–1320.

[46] Brendan Hall, Sarat Chandra Varanasi, Jan Fiedor, Joaquín Arias, Kinjal Basu, Fang Li, Devesh Bhatt, Kevin Driscoll, Elmer Salazar, and Gopal Gupta. 2021. Knowledge-assisted reasoning of model-augmented system requirements with event calculus and goal-directed answer set programming. *arXiv preprint arXiv:2109.04634* (2021).

[47] Liu Hua-Xiao, Wang Shou-Yan, and Jin Ying. 2013. A Tool to Verify the Consistency of Requirements Concern Model. In *2013 International Conference on Computational and Information Sciences*. IEEE, 1955–1958.

[48] Yihao Huang, Jincao Feng, Hanyue Zheng, Jiayi Zhu, Shang Wang, Siyuan Jiang, Weikai Miao, and Geguang Pu. 2019. Prema: a tool for precise requirements editing, modeling and analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1166–1169.

[49] Michael Huth and Mark Ryan. 2004. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press.

[50] Joxan Jaffar, Spiro Michaylov, Peter J Stuckey, and Roland HC Yap. 1992. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 14, 3 (1992), 339–395.

[51] Erwan Jahier, Nicolas Halbwachs, and Pascal Raymond. 2013. Engineering functional requirements of reactive systems using synchronous languages. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 140–149.

[52] Sara Jones, David Till, and Ann M Wrightson. 1998. Formal Methods and Requirements Engineering: Challenges and Synergies. *Journal of Systems and Software* 40, 3 (1998), 263–273.

[53] Andreas Katis, Anastasia Mavridou, Dimitra Giannakopoulou, Thomas Pressburger, and Johann Schumann. 2022. Capture, Analyze, Diagnose: Realizability Checking Of Requirements in FRET. In *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II*. Springer, 490–504.

[54] Robert Könighofer, Georg Hofferek, and Roderick Bloem. 2013. Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies. *International journal on software tools for technology transfer* 15, 5-6 (2013), 563–583.

[55] Vincent Langenfeld, Daniel Dietsch, Bernd Westphal, Jochen Hoenicke, and Amalinda Post. 2019. Scalable analysis of real-time requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 234–244.

[56] Kim Lauenroth and Klaus Pohl. 2008. Dynamic consistency checking of domain requirements in product line engineering. In *2008 16th IEEE International Requirements Engineering Conference*. IEEE, 193–202.

[57] David L Lempia and Steven P Miller. 2009. *Requirements Engineering Management Handbook*. Technical Report.

[58] Meng Li, Baoluo Meng, Han Yu, Kit Siu, Michael Durling, Daniel Russell, Craig McMillan, Matthew Smith, Mark Stephens, and Scott Thomson. 2019. Requirements-based automated test generation for safety critical software. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.

[59] Wenbin Li, David Brown, Jane Huffman Hayes, and Miroslaw Truszczynski. 2014. Answer-set programming in requirements engineering. In *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings 20*. Springer, 168–183.

[60] Jason Xinyu Liu, Ziyi Yang, Benjamin Schornstein, Sam Liang, Ifrah Idrees, Stefanie Tellex, and Ankit Shah. 2022. Lang2ltl: Translating natural language commands to temporal specification with large language models. In *Workshop on Language and Robotics at CoRL 2022*.

[61] Shaoying Liu. 2016. Validating formal specifications using testing-based specification animation. In *Proceedings of the 4th FME Workshop on Formal Methods in Software Engineering*. 29–35.

[62] Levi Lúcio, Salman Rahman, Chih-Hong Cheng, and Alistair Mavin. 2017. Just formal enough? automated analysis of EARS requirements. In *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings 9*. Springer, 427–434.

[63] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. 2016. Resa tool: Structured requirements specification and sat-based consistency-checking. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 1737–1746.

[64] Panagiotis Manolios. 2017. Scalable methods for analyzing formalized requirements and localizing errors. US Patent 9,639,450.

[65] Shahar Maoz and Jan Oliver Ringert. 2021. Spectra: a specification language for reactive systems. *Software and Systems Modeling* 20, 5 (2021), 1553–1586.

[66] Shahar Maoz and Yaniv Sa'ar. 2011. AspectLTL: an aspect language for LTL specifications. In *Proceedings of the tenth international conference on Aspect-oriented software development*. 19–30.

[67] Craig McMillan, Andy Crapo, Michael Durling, Meng Li, Abha Moitra, Panagiotis Manolios, Mark Stephens, and Daniel Russell. 2019. *Increasing development assurance for system and software development with validation and verification using ASSERT*. Technical Report. SAE Technical Paper.

[68] Steven P Miller, Alan C Tribble, Michael W Whalen, and Mats PE Heimdahl. 2006. Proving the shalls: Early validation of requirements through formal methods. *International Journal on Software Tools for Technology Transfer* 8 (2006), 303–319.

[69] Aditya Dev Mishra and Khurram Mustafa. 2022. A review on security requirements specification by formal methods. *Concurrency and Computation: Practice and Experience* 34, 5 (2022), e6702.

[70] Abha Moitra, Kit Siu, Andrew W Crapo, Michael Durling, Meng Li, Panagiotis Manolios, Michael Meiners, and Craig McMillan. 2019. Automating requirements analysis and test case generation. *Requirements Engineering* 24 (2019), 341–364.

[71] Konstantinos Mokos, Theodoros Nestoridis, Panagiotis Katsaros, and Nick Bassiliades. 2022. Semantic Modeling and Analysis of Natural Language System Requirements. *IEEE Access* 10 (2022), 84094–84119.

[72] Marco Montali, Paolo Torroni, Nicola Zannone, Paola Mello, and Volha Bryl. 2011. Engineering and verifying agent-oriented requirements augmented by business constraints with-Tropos. *Autonomous Agents and Multi-Agent Systems* 23, 2 (2011), 193–223.

[73] Massimo Narizzano, Luca Pulina, Armando Tacchella, and Simone Vuotto. 2019. Property specification patterns at work: verification and inconsistency explanation. *Innovations in Systems and Software Engineering* 15 (2019), 307–323.

[74] Anmol Nayak, Hari Prasad Timmapathini, Vidhya Murali, Karthikeyan Ponnalagu, Vijendran Gopalan Venkoparao, and Amalinda Post. 2022. Req2Spec: Transforming software requirements into formal specifications using natural language processing. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 87–95.

[75] Tuong Huan Nguyen, Bao Quoc Vo, Markus Lumpe, and John Grundy. 2014. KBRE: a framework for knowledge-based requirements engineering. *Software Quality Journal* 22 (2014), 87–119.

[76] Pierluigi Nuzzo, Michele Lora, Yishai A Feldman, and Alberto L Sangiovanni-Vincentelli. 2018. CHASE: Contract-based requirement engineering for cyber-physical system design. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 839–844.

[77] Office of the Chief Engineer. 2013. *NASA systems engineering processes and requirement*. Technical Report.

[78] Prasanna Padmanabhan and Robyn R Lutz. 2005. Tool-supported verification of product line requirements. *Automated Software Engineering* 12, 4 (2005), 447–465.

[79] SK Pandey and Mona Batra. 2013. Formal Methods in Requirements Phase of SDLC. *International Journal of Computer Applications* 70, 13 (2013), 7–14.

[80] Xingxing Pi, Jianqi Shi, Yanhong Huang, and Hansheng Wei. 2019. Automated Mining and Checking of Formal Properties in Natural Language Requirements.

In *Knowledge Science, Engineering and Management: 12th International Conference, KSEM 2019, Athens, Greece, August 28–30, 2019, Proceedings, Part II 12*. Springer, 75–87.

[81] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2006. 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, 364–380.

[82] Strategic Planning. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology* 1 (2002).

[83] Uwe Proß, Erik Markert, Jan Langer, Andreas Richter, Chris Drechsler, and Ulrich Heinkel. 2008. A platform for requirement based formal specification. In *2008 Forum on Specification, Verification and Design Languages*. IEEE, 237–238.

[84] Christopher L Robinson-Mallett and Robert M Hierons. 2017. Integrating graphical and natural language specifications to support analysis and testing. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 331–338.

[85] Stefan Runde and Alexander Fay. 2011. Software support for building automation requirements engineering—An application of semantic web technologies in automation. *IEEE Transactions on Industrial Informatics* 7, 4 (2011), 723–730.

[86] RTCA (Firm). SC-205. 2011. *DO-178C: Software considerations in airborne systems and equipment certification*. Technical Report.

[87] Patrizia Scandurra, Andrea Arnoldi, Tao Yue, and Marco Dolci. 2012. Functional requirements validation by transforming use case models into Abstract State Machines. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 1063–1068.

[88] Jan Scheffczyk, Uwe M Borghoff, Andreas Birk, and Johannes Siedersleben. 2005. Pragmatic consistency management in industrial requirements specifications. In *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*. IEEE, 272–281.

[89] Murray Shanahan. 2001. The Event Calculus Explained. In *Artificial intelligence today: Recent trends and developments*. Springer, 409–430.

[90] Ernst Sikora, Marian Daun, and Klaus Pohl. 2010. Supporting the consistent specification of scenarios across multiple abstraction levels. In *Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, REFSQ 2010, Essen, Germany, June 30–July 2, 2010. Proceedings 16*. Springer, 45–59.

[91] Kit Siu, Abha Moitra, Michael Durling, Andy Crapo, Meng Li, Han Yu, Heber Herencia-Zapana, Mauricio Castillo-Effen, Shiraj Sen, Craig McMillan, et al. 2017. Flight critical software and systems development using ASSERT. In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.

[92] Daniel Smullen and Travis D Breaux. 2016. Modeling, analyzing, and consistency checking privacy requirements using eddy. In *Proceedings of the Symposium and Bootcamp on the Science of Security*. 118–120.

[93] Srihari Sukumaran, Ashok Sreenivas, and R Venkatesh. 2006. A rigorous approach to requirements validation. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*. IEEE, 236–245.

[94] Sabine Teufl, Dongyue Mou, and Daniel Ratiu. 2013. MIRA: A Tooling-Framework to Experiment with Model-Based Requirements Engineering. In *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 330–331.

[95] Judith Thyssen and Benjamin Hummel. 2013. Behavioral specification of reactive systems using stream-based I/O tables. *Software & Systems Modeling* 12 (2013), 265–283.

[96] Kriangkrai Traichaiyaporn and Toshiaki Aoki. 2013. Preserving correctness of requirements evolution through refinement in event-b. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 1. IEEE, 315–322.

[97] Naoyasu Ubayashi, Yasutaka Kamei, Masayuki Hirayama, and Tetsuo Tamai. 2011. A context analysis method for embedded systems—Exploring a requirement boundary between a system and its context. In *2011 IEEE 19th International Requirements Engineering Conference*. IEEE, 143–152.

[98] Sarat Chandra Varanasi, Joaquin Arias, Elmer Salazar, Fang Li, Kinjal Basu, and Gopal Gupta. 2022. Modeling and Verification of Real-Time Systems with the Event Calculus and s (CASP). In *International Symposium on Practical Aspects of Declarative Languages*. Springer, 181–190.

[99] Simone Vuotto, Massimo Narizzano, Luca Pulina, and Armando Tacchella. 2019. Poster: Automatic consistency checking of requirements with ReqV. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 363–366.

[100] Michael Wahler, David Basin, Achim D Brucker, and Jana Koehler. 2010. Efficient analysis of pattern-based constraint specifications. *Software & Systems Modeling* 9 (2010), 225–255.

[101] Abderrahim Ait Wakrime, J Paul Gibson, and Jean-Luc Raffy. 2018. Formalising the requirements of an E-voting software product line using Event-B. In *2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 78–84.

[102] Benedikt Walter, Jakob Hammes, Marco Piechotta, and Stephan Rudolph. 2017. A formalization method to process structured natural language to logic expressions to detect redundant specification and test statements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 263–272.

[103] WenXuan Wang, Jun Hu, JianChen Hu, JieXiang Kang, Hui Wang, and ZhongJie Gao. 2020. Automatic test case generation from formal requirement model for avionics software. In *2020 6th International Symposium on System and Software Reliability (ISSSR)*. IEEE, 12–20.

[104] Rongjie Yan, Chih-Hong Cheng, and Yesheng Chai. 2015. Formal consistency checking over specifications in natural languages. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1677–1682.

[105] Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2023. Harnessing the Power of Large Language Models for Natural Language to First-Order Logic Translation. *arXiv preprint arXiv:2305.15541* (2023).

[106] Tong Ye, Yi Zhuang, and Gongzhe Qiao. 2022. MBIPV: a model-based approach for identifying privacy violations from software requirements. *Software and Systems Modeling* (2022), 1–30.

[107] Farzana Zahid, Awais Tanveer, Matthew MY Kuo, and Roopak Sinha. 2022. A systematic mapping of semi-formal and formal methods in requirements engineering of industrial cyber-physical systems. *Journal of Intelligent Manufacturing* 33, 6 (2022), 1603–1638.

[108] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Information and Software Technology* 53, 6 (2011), 625–637.

Received 15 December 2023; accepted 12 January 2024