Report from Dagstuhl Seminar 25392

Specification Engineering: Foundations for the Future of Software Development

Marsha Chechik*[1], Eunsuk Kang*[2], Shahar Maoz*[3], Jan Oliver Ringert*[4], and Allison Sullivan*[5]

1 **University of Toronto, CA.** chechik@cs.toronto.edu
2 **Carnegie Mellon University – Pittsburgh, US.** eunsukk@andrew.cmu.edu
3 **Tel Aviv University, IL.** maoz@cs.tau.ac.il
4 **Bauhaus-Universität Weimar, DE.** jan.ringert@uni-weimar.de
5 **University of Texas at Arlington, US.** allison.sullivan@uta.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 25392 "Specification Engineering: Foundations for the Future of Software Development". Specifications are an essential component in a variety of tasks in software engineering, including software verification, testing, modeling, requirements engineering, and program synthesis. While producing quality specifications has been a longstanding problem, recent advances in AI technologies, such as large-language models (LLMs), make it a timely problem to address from new perspectives. Automatically generating code from a high-level specification will likely emerge as a dominant paradigm for software development in the future. Thus, being able to write, maintain and evolve high quality specifications – the process of **specification engineering** – will become an essential skill for software engineers. This Dagstuhl Seminar brought together leading researchers in software engineering and formal methods to identify foundational problems and build a roadmap for specification engineering as a central activity in future development processes.

Seminar September 21–26, 2025 – https://www.dagstuhl.de/25392
2012 ACM Subject Classification Software and its engineering → Specification languages
Keywords and phrases formal methods, software assurance, software specification, specification engineering
Digital Object Identifier 10.4230/DagRep.15.9.160

1 Executive Summary

Marsha Chechik (University of Toronto, CA)
Eunsuk Kang (Carnegie Mellon University – Pittsburgh, US)
Shahar Maoz (Tel Aviv University, IL)
Jan Oliver Ringert (Bauhaus-Universität Weimar, DE)
Allison Sullivan (University of Texas at Arlington, US)

Formal specifications are mathematically precise descriptions of the behavior or properties of a system. Specifications are an essential component in a variety of tasks in software engineering, including software verification, testing, modeling, requirements engineering, and program synthesis. Despite a wealth of research on techniques and tools that take specifications as input, relatively less has been explored on addressing the challenges of

* Editor / Organizer

Specification Engineering: Foundations for the Future of Software Development, *Dagstuhl Reports*, Vol. 15, Issue 9, pp. 160–182
Editors: Marsha Chechik, Eunsuk Kang, Shahar Maoz, Jan Oliver Ringert, and Allison Sullivan
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

coming up with specifications in the first place, and maintaining them as system requirements evolve. Typically, specifications are assumed to have been created by software engineers, who may not have sufficient training or expertise in specification languages. Little is understood about what makes a specification "correct" or "high-quality", and how to validate specifications to ensure that they accurately reflect a user's intent. Specification tools are also notorious for their poor usability and high learning curve.

While producing quality specifications has been a longstanding problem, recent advances in AI technologies, such as large-language models (LLMs), make it a timely problem to address from new perspectives. Automatically generating code from a high-level specification will likely emerge as a dominant paradigm for software development in the future. Thus, being able to write, maintain and evolve high-quality specifications – the process of specification engineering – will become an essential skill for software engineers. LLMs are also being explored by researchers as a promising way of generating formal specifications from natural language requirements. However, since LLMs themselves do not provide guarantees about the correctness or quality of their output, new methods for validating and improving the quality of generated specifications will be crucial to make them reliable and useful.

This Dagstuhl Seminar "Specification Engineering: Foundations for the Future of Software Development" (25392) brought together leading researchers in software engineering and formal methods to identify foundational problems and build a roadmap for specification engineering as a central activity in future development processes. The seminar was organized around the following questions:

- Quality and Validation: What are key properties of a high-quality specification? How do we debug, validate, and repair specifications for these properties?
- Usability: How do we make it easier for engineers to express and validate their intent in a specification language? How do we make specifications readable and comprehensible?
- Scalable Specification Construction: How do we construct large, complex specifications out of smaller ones? How do we facilitate reuse of specifications? How do we support incremental, modular changes to a specification?
- Specification for/with AI: How do we use and tailor AI-based tools for specification-driven tasks such as code generation and verification? How do we make these models more effective at generating specifications from natural languages?

Activities

The seminar consisted of (1) several invited, "anchoring" talks around the four major topics listed above, (2) a series of shorter, "lightning" talks where participants shared new ideas, open problems, or ongoing projects on the topic of specification, and (3) two sets of breakout discussions. The first set of breakouts was assigned based on the four topics; after the initial discussions, the participants were encouraged to suggest or form different groups based on their topics of interest that emerged. The resulting second set of breakouts were centered around the topic of AI, covering LLMs for specification activities, specification of LLMs, and the use of AI for domain modeling. These activities were interleaved with ad-hoc discussions around the very concept of "specification" itself as well as planning for post-seminar activities and collaborations.

Outcome

Among many stimulating discussions around the topic of specification, two major themes emerged. First, the participants realized that the very idea of "specification" may not be as well-defined or agreed upon as many had previously thought before the seminar. For example, to some participants, a specification had a specific meaning as a type of artifact that describes the expected behavior of a program or a system (e.g., API contracts), while others thought that nearly every software artifact (e.g., code) could be considered a specification. After a seminar-wide discussion, the participants agreed that it would be more meaningful to talk about properties of a specification (e.g., whether it is formal or informal, readable, analyzable, modifiable, for what purpose it is used, etc.,) rather than attempting to define what a specification is (and is not).

Second, many participants agreed that specifications will have an essential role in the age of AI-driven development and provide new opportunities for research as well as engagement with practitioners. For example, natural language prompts are emerging as a common mechanism to specify developers' intent and system requirements, from which an implementation is automatically generated. However, it was also noted that informal, unstructured prompts are not an ideal specification mechanism for developing, debugging, and maintaining complex software systems, and that more structured specification methods are needed to support both developers and AI agents in these tasks. On the other hand, the participants also agreed that traditional specification methods and tools developed by the research community will likely need to be adapted or rethought to support the fuzzy, interactive, and informal ways in which developers collaborate with AI to develop software.

2 Table of Contents

3 Overview of Talks

3.1 Management of specifications in the large

Bernhard Rumpe (RWTH Aachen, DE)

The management of specifications at scale requires systematic coordination across multiple models written in multiple modeling languages and notations, each with its own semantics and interpretations. Our work focuses on the foundations of model-based software engineering, in particular the semantics of models and the construction of software tools that support precise, analyzable model representations. A central research question is how to formally relate heterogeneous models so that their structures, behavioral properties, and interdependencies remain coherent across levels of abstraction and domains. This includes investigating how symbols of various kinds defined in one model can be connected to in other models to lay the foundation for a robust and maintainable specification management. We want to apply these foundational principles for human-constructed models, but also derived models and potentially AI-generated artifacts, aiming to enable robustly shared syntactically connected models that support automated reasoning, collaborative design, and scalable model evolution. This direction to our belief is a core pillar for a unifying semantic basis for specification engineering in increasingly complex, heterogeneous system engineering environments, where even UML and SysML only cover a partial subset of viewpoints and are internally also not well integrated yet.

3.2 Formal Specification Generation as Design Transformations and the Limits of Automation

Mauricio Castillo-Effen (Lockheed Systems – Arlington, US)

Formal specifications play an increasingly important role in enabling the adoption of formal methods and automated reasoning in industrial settings, yet the process of deriving them from natural-language requirements remains poorly understood. Current approaches to specification generation often treat it as a translation problem, assuming that requirements encode the information needed for formalization. We question this assumption, particularly in the context of rapid, iterative Systems Engineering and the growing interest in applying generative AI to support these processes.

In this talk, we presented preliminary results from studying the generation of formal specifications as a design transformation process characterized by a gradual reduction of epistemic uncertainty. We took two complementary perspectives. First, we studied the cognitive processes applied by humans to transform informal requirements into increasingly formal design artifacts. Second, we evaluated whether agentic generative AI systems could perform similar transformations with comparable epistemic rigor.

To structure this analysis, we used Gero's Function–Behavior–Structure (F-S-B) ontology as a model of the design process and its successive refinements. Using this concept, we carried out an empirical study on a design challenge that involved exploring a large design

space with the goal of generating adaptable search-and-rescue drones. We tasked human participants and different configurations of agentic generative AI systems with the same objectives and constraints.

Our findings indicated that while agentic LLM systems could generate syntactically well-formed artifacts, they struggle with context preservation, controlled abstraction, and uncertainty management. Common failure modes included premature concretization, cascading semantic errors, and ontology drift, resulting in artifacts that resemble specifications without supporting their intended epistemic role ("specifictions"). In contrast, human designers rely on implicit contextual knowledge and iterative reformulation to progressively stabilize meaning before formalization.

We conclude that generating formal specifications cannot be reliably delegated to autonomous agents without well-defined mechanisms for transparent uncertainty management, contextual grounding, and human oversight. These results suggest that future AI-assisted approaches to specification engineering should emphasize not only automation but also support the human users' ability to steer the formalization process.

3.3 Abstraction Engineering

Benoît Combemale (INRIA – Rennes, FR)

Joint work of Nelly Bencomo, Jordi Cabot, Marsha Chechik, Betty H.C. Cheng, Benoît Combemale, Andrzej Wąsowski, Steffen Zschaler
Main reference Nelly Bencomo, Jordi Cabot, Marsha Chechik, Betty H. C. Cheng, Benoît Combemale, Andrzej Wąsowski, Steffen Zschaler: "Abstraction Engineering", CoRR, Vol. abs/2408.14074, 2024.
URL http://dx.doi.org/10.48550/ARXIV.2408.14074

Modern software-based systems operate under rapidly changing conditions and face ever-increasing uncertainty. In response, systems are increasingly adaptive and reliant on artificial-intelligence methods. In addition to the ubiquity of software with respect to users and application areas (e.g., transportation, smart grids, medicine, etc.), these high-impact software systems necessarily draw from many disciplines for foundational principles, domain expertise, and workflows. Recent progress with lowering the barrier to entry for coding has led to a broader community of developers, who are not necessarily software engineers. As such, the field of software engineering needs to adapt accordingly and offer new methods to systematically develop high-quality software systems by a broad range of experts and non-experts. In [1], we look at these new challenges and propose to address them through the lens of Abstraction. Abstraction is already used across many disciplines involved in software development – from the time-honored classical deductive reasoning and formal modeling to the inductive reasoning employed by modern data science. The software engineering of the future requires Abstraction Engineering – a systematic approach to abstraction across the inductive and deductive spaces. We discuss the foundations of Abstraction Engineering, identify key challenges, highlight the research questions that help address these challenges, and create a roadmap for future research.

References

1 N Bencomo, J Cabot, M Chechik, BHC Cheng, B Combemale, S Zschaler. *Abstraction engineering*. arXiv (`https://arxiv.org/abs/2408.14074`), 2024.

3.4 Helping students learn formal specification

Alcino Cunha (University of Minho, PT)

Joint work of Alcino Cunha, Nuno Macedo, José Creissac Campos, Iara Margolis, Emanuel Sousa
Main reference Alcino Cunha, Nuno Macedo, José Creissac Campos, Iara Margolis, Emanuel Sousa: "Assessing the impact of hints in learning formal specification", in Proc. of the 46th International Conference on Software Engineering: Software Engineering Education and Training, SEET@ICSE 2024, Lisbon, Portugal, April 14-20, 2024, pp. 151–161, ACM, 2024.
URL http://dx.doi.org/10.1145/3639474.3640050

Alloy4Fun is a web tool for helping students self study the Alloy formal specification language. In particular, it allows the creation of specification challenges where students are asked to formally specify natural language requirements. If the specification is incorrect, a counter-example is depicted graphically, as usual in Alloy. This counter-example can be seen as a hint that helps the student progress towards the correct specification. Unfortunately, we had some anecdotal evidence that students sometimes struggle to understand such hints. Recently, we conducted a large user study to assess the impact of this and other kinds of hints in learning formal specification. In this talk I presented the design of this study and briefly discussed its results. The main conclusion of the study is that none of the studied hints had an impact on learning retention, and only giving the student precise error locations had an impact on immediate performance. Finally, I discussed some potential uses of LLM in this context, namely using LLMs to help students understand counter-examples or wrong specifications.

3.5 ML for Specifications for ML

Matthew Dwyer (University of Virginia – Charlottesville, US)

Formalizing functional requirements as specifications can enable a variety of powerful validation and verification (V&V) approaches to be applied. Such specifications formulate a precondition, which describes a set of system inputs, and an associated postcondition, which constraints system output for those inputs. However, precisely formulating requirements for ML-enabled systems that process raw sensor data, e.g., image, lidar, can be very challenging.

In this talk we will describe why it can be challenging to directly formulate specifications for such systems and propose an alternative approach that develops black-box models for specification preconditions. Inputs generated from such models conform to preconditions with high-probability and are both realistic and diverse – desirable properties for V&V. Consequently, observed system behavior for those inputs can then be checked against formalizations of postconditions to enable a form of spec-based V&V.

The approach to developing these precondition models begins with the standard approach of formulating natural language statements of functional requirements over a glossary of domain-specific terms that define semantic features of inputs. It then combines several different ML techniques to construct a generative model for the precondition which can be leveraged for V&V. In this way, ML supports the development of specification models for V&V of ML-enabled systems.

As the saying goes "All models are wrong, but some are useful" and the first part applies to the models we generate, but preliminary results suggest that the second part may also apply. We will end with a series of questions related to how and when such techniques might be usefully applied that we hope initiates fruitful discussion.

3.6 Context-Aware Trace Contracts

Reiner Hähnle (TU Darmstadt, DE)

Joint work of Reiner Hähnle, Eduard Kamburjan, Marco Saletta

We illustrate the usage of Context-Aware Trace Contracts (for short: CATs) by way of an example. CATs are a systematic approach to specify non-procedure local behavior. Technically, they consist of symbolic expressions specifying the assumed behavior of the callers before a procedure enters its contract, the behavior a procedure guarantees, and the behavior expected to happen in the continuation after termination. This generalizes state-based, Hoare-style specification triples.

References

1 Hähnle, R., Kamburjan, E., Scaletta, M.: Context-aware trace contracts. In: De Boer, F., Damiani, F., Hähnle, R., Johnsen, E.B., Kamburjan, E. (eds.) Active Object Languages: Current Research Trends. LNCS, vol. 14360, pp. 292–325. Springer, Cham (2024).

3.7 Role of Specification Languages in Verification of Neuro-Symbolic Systems and Complex Systems with Machine Learning Components

Ekaterina Komendantskaya (Heriot-Watt University – Edinburgh, GB)

Machine Learning (ML) is increasingly used to implement components of Cyber-Physical Systems – i.e. systems that interact with the physical (continuous) world and at the same time have (digital) programmed components. Usually, ML components are used on the intersection of these two, i.e. a neural network usually processes a sensor input and gives class predictions or discrete commands. For example, in a car, a neural controller may measure how close the car is to an obstacle, and give a command to brake. In a medical application, a neural network may measure the patient's temperature, blood pressure or blood composition and decide on a dose of administered drugs. Both cases are examples of safety-critical systems, i.e. systems whose failure potentially endangers health and life of the users. Formally verifying that such systems "do not go wrong" is one of the biggest challenges of the day.

One of the most known problems in the domain is the problem of "a missing spec" that refers to the fact that ML components are obtained via data-driven optimisation procedures, and come without any clear formal specification. However, often specifications are available, thanks to the knowledge of the safety requirements concerning the symbolic components of the system in question. E.g. we may know the critical distance or critical dose, after which

the system goes into an unsafe state; and these can result in meaningful safety specifications. Such cases are a sweet spot for formal specification and verification. In my talk, I discussed the benefits of deploying a domain-specific language Vehicle for writing specifications of properties of ML components of neuro-symbolic systems. Vehicle allows users to specify the properties of the neural components of neuro-symbolic programs once, and then safely compile the specification to other interfaces (ML solvers, interactive theorem provers, ML backends) using a tailored typing and compilation procedure. I gave a high-level overview of Vehicle's overall design, its interfaces and compilation & type-checking procedures, and then demonstrated its utility by formally verifying the safety of a simple autonomous car controlled by a neural network, operating in a stochastic environment with imperfect information.

Vehicle is available at: `https://github.com/vehicle-lang/vehicle`.

3.8 Specification Reverse Engineering for Decentralized Applications

Yi Li (Nanyang TU – Singapore, SG)

Joint work of Yi Li, Ye Liu,, Yixuan Liu, Zhiyang Chen, Cyrille Artho, Chengxuan Zhang
Main reference Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, Yang Liu: "PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation", in Proc. of the 32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025, The Internet Society, 2025.
URL https://www.ndss-symposium.org/ndss-paper/propertygpt-llm-driven-formal-verification-of-smart-contracts-through-retrieval-augmented-property-generation/

Smart contracts are computer programs running on blockchains to implement Decentralized Applications. The absence of contract specifications hinders routine tasks, such as contract verification, security auditing, and effective test generation, leading to vulnerabilities and increased development costs. In this talk, we introduce the concept of Specification Reverse Engineering for smart contracts and presented a summary of our past works in this field. The two main approaches are: (1) "learning from the past", where benign scenarios from smart contract transaction histories are used to generate function-level invariants, e.g., InvCon [1] and InvCon+ [2], and contract-level behavior models, e.g., SMCon [3] and SPCon [4]; and (2) "learning from each other", where specifications of other similar smart contracts can be reused in constructing new specifications, e.g., Trace2Inv [5] and PropertyGPT [6].

References

1 Ye Liu and Yi Li. Oct. 2022. InvCon: a dynamic invariant detector for Ethereum smart contracts. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 1–4.
2 Ye Liu, Chengxuan Zhang, and Yi Li. 2025. Automated invariant generation for Solidity smart contracts. IEEE Transactions on Dependable and Secure Computing.
3 Ye Liu, Yixuan Liu, Yi Li, and Cyrille Artho. Mar. 2025. Specification mining for smart contracts with trace slicing and predicate abstraction. In Proceedings of the 32nd IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).
4 Ye Liu, Yi Li, Shang-Wei Lin, and Cyrille Artho. July 2022. Finding permission bugs in smart contracts with role mining. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pages 716–727, New York, NY, USA. ACM.
5 Zhiyang Chen, Ye Liu, Sidi Mohamed Beillahi, Yi Li, and Fan Long. July 2024. Demystifying invariant effectiveness for securing smart contracts. In Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE), volume 1 of number FSE, pages 1772–1795. ACM New York, NY, USA.

6 Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. Feb. 2025. PropertyGPT: LLM-driven formal verification of smart contracts through retrieval-augmented property generation. In Proceedings of 32nd Annual Network and Distributed System Security Symposium (NDSS).

3.9 Exploring Development Methods for Reactive Synthesis Specifications

Shahar Maoz (Tel Aviv University, IL), and Jan Oliver Ringert (Bauhaus-Universität Weimar, DE)

Joint work of Shahar Maoz, Dor Ma'ayan, Jan Oliver Ringert
Main reference Dor Ma'ayan, Shahar Maoz, Jan Oliver Ringert: "Exploring Development Methods for Reactive Synthesis Specifications", ACM Trans. Softw. Eng. Methodol., Association for Computing Machinery, 2025.
URL http://dx.doi.org/10.1145/3767159

Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification. Despite significant research progress in the past decades, reactive synthesis is still in an early stage of use. Previous studies found that the lack of development methods for reactive synthesis specifications is one barrier to its wider adoption. In this paper, we adapt two development methods, an incremental method and a modular method, to the context of reactive synthesis specifications. The methods are based on existing software development methods on the one hand and studies about reactive synthesis on the other hand. Then, we report on an exploratory case study in which participants developed specifications using the two methods. We evaluated the methods using a mixed-method analysis that combines grounded theory analysis of Slack communication with participants, quantitative exploratory data analysis of the synthesis IDE usage logs, and qualitative independent expert review of the final specifications. Our findings show clear benefits of modular specification development in terms of ease of planning, synthesis time, fewer unrealizability issues, and faster debugging. However, the incremental development method was more natural and easy to use, and specifications developed incrementally were also easier to validate during the development process.

3.10 Specification engineering: Notes on usability

Shahar Maoz (Tel Aviv University, IL)

I discuss the challenge of usability in the context of specifications. Usability consists of learnability, effectiveness, efficiency, memorability, error prevention and recovery, and user satisfaction. In the context of specification, these are manifested in terms of language – including syntax, semantics, and abstractions, and in terms of analysis tools – their input, output, and methodology. I will briefly discuss two examples for the challenge. First, the case of temporal logics and specification patterns. While LTL is known for being difficult to read and write correctly, and specification patterns were suggested as a means to describe

common properties, very little research has been done on their usability, i.e., on patterns learnability, effectiveness, efficiency, etc. Second, the case of an unrealizable core. While specifications for reactive synthesis are often unrealizable, and computing an unrealizable core (minimal unrealizable subset) was suggested as a means to localize the fault, very little research has been done on unrealizable core's usability, again, on its learnability, effectiveness, etc. Finally, I will present two recent example projects that deal with the comprehension of specifications and with the process of developing them.

3.11 What Properties Affect Boolean Formula Comprehension in Formal Specifications?

Shahar Maoz (Tel Aviv University, IL)

Joint work of Shahar Maoz, Ilia Shevrin
Main reference Ilia Shevrin, Shahar Maoz: "What Properties Affect Boolean Formula Comprehension in Formal Specifications?", ACM Trans. Softw. Eng. Methodol., Association for Computing Machinery, 2025.
URL http://dx.doi.org/10.1145/3744557

Writing formal specifications is an important yet challenging aspect of software engineering. Correct specifications facilitate verification efforts and reduce bugs. However, the declarative nature of specifications differs from the imperative approach of most common programming languages, and software engineers often perceive formal methods as difficult. Arguably, guidelines and tools for writing readable specifications should lower the barrier to formal methods adoption. In this work, we focus on Boolean formulas, a fundamental building block of specifications. Analogous to research on code comprehension, we conducted an experiment that attempts to identify what properties affect Boolean formula comprehension by software engineers. To this end, we collected 59 representative Boolean formulas and tested how various syntactic properties, such as negation symbol count and nesting level, affect comprehension task response times and correctness. Our experiment with 181 participants shows that eliminating negation symbols and decreasing operator count are among the most significant factors that improve comprehension. We use these empirical results to derive a reading complexity score and develop a fast regression-based refactoring algorithm for Boolean formulas. Finally, we conducted a follow-up experiment with 57 participants, which provided strong evidence for the algorithm's effectiveness in improving comprehension.

3.12 Learning LTL Specifications from Demonstrations with Uncertainty

Rômulo Meira-Góes (Pennsylvania State University – University Park, US)

Joint work of Rômulo Meira-Góes, Constantino Lagoa, Parastou Fahim
Main reference Parastou Fahim, Constantino Lagoa, Rômulo Meira-Góes: "Learning Linear Temporal Specifications from Demonstrations with Uncertainty", submitted to The 2026 American Control Conference.

Learning temporal logic specifications from system demonstrations is essential for tasks such as formal verification and controller synthesis, especially in safety-critical domains. Existing approaches typically assume demonstrations are correct or only affected by misclassification errors. In practice, however, system traces are often uncertain or incomplete due to sensor

faults, measurement errors, or data loss. We present a framework for learning minimal Linear Temporal Logic (LTL) formulas from demonstrations with uncertainty. Our approach models uncertainty via Hamming distance to generate possible estimates around each observed trace, which are grouped with constraints requiring that at least one trace per group is consistent with the learned formula. Our problem is then reduced to an equivalent Pseudo-Boolean Optimization. We evaluate our method against state-of-the-art LTL learning approaches and show that it recovers specifications that more closely align with ground-truth formulas under uncertainty.

3.13 Specification Engineering for Neuro-symbolic Programming and Vice Versa

Federico Mora (University of Waterloo, CA)

Neuro-symbolic programming systems often use a machine learning (neuro) component to generate candidate programs and a program analysis (symbolic) component to check candidate programs. When the symbolic component finds an issue, the neuro component tries again. Eudoxus [1] is one example neuro-symbolic system that follows this scheme. This talk discusses the challenges of generating programs that use APIs through similar neuro-symbolic approaches. For such neuro-symbolic systems to work as a whole, we need good specifications for all relevant APIs. When a specification is too restrictive, the symbolic component will block valid candidate programs. When the specification is too forgiving, the symbolic component will allow incorrect candidate programs. Specifically, this talk discusses the challenges of and opportunities for engineering good specifications at scale in the neuro-symbolic programming context.

References

1 Federico Mora, Justin Wong, Haley Lepe, Sahil Bhatia, Karim Elmaaroufi, George Varghese, Joseph E. Gonzalez, Elizabeth Polgreen, Sanjit A. Seshia: Synthetic Programming Elicitation for Text-to-Code in Very Low-Resource Programming and Formal Languages. NeurIPS 2024

3.14 Temporal Logic Sketching

Daniel Neider (TU Dortmund, DE)

Joint work of Simon Lutz, Daniel Neider, Rajarshi Roy
Main reference Simon Lutz, Daniel Neider, Rajarshi Roy: "Specification Sketching for Linear Temporal Logic", in Proc. of the Automated Technology for Verification and Analysis – 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II, Lecture Notes in Computer Science, Vol. 14216, pp. 26–48, Springer, 2023.
URL http://dx.doi.org/10.1007/978-3-031-45332-8_2

Temporal Logic Sketching is a novel approach designed to simplify the process of writing formal specifications. The central idea is that an engineer can provide a partial formula, called a sketch, in which components that are difficult to formalize may be left unspecified. Given a set of examples describing desired and undesired system behaviors, the goal of a sketching algorithm is to complete the sketch so that the resulting specification is consistent with the provided examples.

This talk presents recent advances in specification sketching and surveys existing approaches for various temporal logics, including Linear Temporal Logic (LTL), Signal Temporal Logic (STL), Metric Temporal Logic (MTL), Property Specification Language (PSL), Computation Tree Logic (CTL), and Alternating-time Temporal Logic (ATL). It highlights both the challenges inherent in this paradigm and the opportunities it offers for specification engineering. In addition, the talk outlines key theoretical contributions, such as a comprehensive complexity analysis of learning logical formulas from data.

3.15 Task Models as a Mean to Identify and Justify Automations in Software Programming Tasks

Phillippe Palanque (Toulouse University, FR)

Programming is usually considered as a difficult task [1] and even some metrics about the cognitive aspect of this difficulty have been proposed [4]. Associated with this difficulty is the diversity of tasks such as structuring code, writing code, testing code or even reading and understanding error messages [2]. One key contributing factor to addressing this difficulty is training and learning [4] and another key one is the programming environment used for programming [3].

This presentation has argued that building and exploiting both high-level and detailed descriptions of programming tasks would provide multiple benefits in terms of training and learning in general [5] (but also specifically in the area of safety critical systems such as aeronautics).

Identifying and describing tasks in such a way might appear as a niche and cumbersome work, however, with the multiplicity of software assistants [6] and with virtually any modern IDE proposing such tools while also allowing for custom add-ons, listing, describing and modeling developers' tasks is needed if we want to be able to compare the usability and performance of such tools.

As argued in [7] the complexity of tasks is not an intrinsic value but a combined value of the tasks themselves and the system that is used to perform the tasks. This is known as the task-artefact cycle, which requires, at design time of an interactive system, the identification of the tasks and the evolution of their complexity when modifications are made on the system.

In this presentation we propose the exploitation of the HAMSTERS task modeling notation to represent programmers' tasks and to assess their complexity on a concrete IDE. We also argue that automating some of these tasks with software assistants (which is a very common activity in the area of software engineering (see a recent survey in [6]) should be assessed in terms of how they demonstrate a reduction of the complexity of these tasks and how learning how to perform those tasks is improved.

References

1 B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023). ACM, 500–506.

2 B. A. Becker, P. Denny, R. Pettit, D. Bouchard, D. J. Bouvier, B. Harrington, A. Kamil, A. Karkare, C. McDonald, P.-M. Osera, J. L. Pearce, and J. Prather. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In Proc. of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '19). ACM, 177–210.

3 C. M. Lewis. How programming environment shapes perception, learning and goals: logo vs. scratch. In Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10). ACM, 346–350.
4 B. Alwis, G. C. Murphy, and S. Minto. Creating a cognitive metric of programming task difficulty. In Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering (CHASE '08). ACM, 29–32.
5 C. Martinie, D. Navarre, P. Palanque, and C. Fayollas. A generic tool-supported framework for coupling task models and interactive applications. ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '15). ACM, 244-253.
6 S. Leblanc, M. Burgueño, L. Cabot, J. Le Pallec, X, Gérard, Software assistants in software engineering: A systematic mapping study. Softw: Pract Exper. 2023; 53(3): 856–892.
7 C. Martinie, P. Palanque, E. Bouzekri, A. Cockburn, A. Canny, and Eric Barboni. Analysing and Demonstrating Tool-Supported Customizable Task Notations. Proc. ACM Hum.-Comput. Interact. 3, EICS, Article 12 (jun 2019), 26 pages.

3.16 Live Programming for Specs (and Beyond)

Allison Sullivan (University of Texas at Arlington, US)

In a quest to explore ways we could broaden adoption of specification languages, this talk highlights some new research directions Dr. Sullivan is exploring. The topics center around the concept of "live programming" which is a body of work dedicated to building development environments that interweave writing and executing programs. There are interesting challenges to do this for traditional imperative languages (e.g. Java) but for a specification language, highlighting output changes is as "simple" as demonstrating why two formulas differ. So with that in mind, where can we take live programming for specs? What specification languages could this apply to?

3.17 Specification in the Large at Amazon Web Services

Michael W. Whalen (Amazon Inc. – Minneapolis, USA & The University of Minnesota – Minneapolis, USA)

AWS is perhaps the world's largest user of automated reasoning, which is driven by specifications. In this talk, I will first give an overview of different projects using specifications and difficulties we have, then focus on the grand challenge of creating specifications for reasoning across multiple microservices to prove customer-relevant properties.

3.18 No more garbage in: Validating formal models

Pamela Zave (Princeton University, US)

This talk presents an overview of validation, as I see it and practice it. It offers and justifies three complementary definitions of validity, and how a formal model can be validated for each. It also presents two opinions, formed during my experience of validating many formal models, on how validation should be done.

3.19 Precision in formal modeling: Why we need it, and how to get it

Pamela Zave (Princeton University, US)

We used to say that a formal model should be "complete, consistent, and unambiguous." And I always wondered about "unambiguous," because a well-formed formal model cannot be ambiguous. This talk shows how precision–the traditional partner of accuracy–applies to formal models, and describes what unambiguity is supposed to describe. The talk also applies the thinking of Michael Jackson to birthday cakes.

3.20 The CNA model of network architecture

Pamela Zave (Princeton University, US)

Compositional Network Architecture is a general, reusable formal model of network architecture. It is the basis for a recent book on the subject–the first networking book to be based on a formal model!

This talk introduces the Alloy model, which is included in the seminar materials, with emphasis on the seminar topics: (1) It has been thoroughly validated (62 percentage of the code is strictly for VALIDATION). (2) The model is big and complex, but would be less so if it could be separated into views. I explain the desired decomposition, which I wish were supported by tools for SCALABLE CONSTRUCTION. (3) The model is also being translated into Isabelle so we can prove theorems about it. The translation, and the motivations for it, is related to many questions about USABILITY.

4 Working groups

4.1 What is a Specification?

Marsha Chechik (University of Toronto, CA) and others

This breakout tackled fundamental definitional questions about specifications in the age of AI. They built on a list of essential properties of specifications that the entire seminar group came up with during an earlier discussion: adequacy to the problem, including: formality, readability, modifiability, precision, incrementality, comprehensibility, decomposability, and fitness to use case. The breakout group compared three development paradigms across multiple dimensions of properties: vibe coding (where the specification is the history of prompts and feedback iterated until user acceptance), specifications for correctness of AI components (assuming we cannot fully specify behavior, using lists of examples and accepting black-boxes as valid components), and traditional code contracts (formal, complete specifications like sorting algorithm contracts with precise logical properties). An outcome of this exercise was that specifications remain fundamentally about abstraction, communication, and intent – but the forms they take and the guarantees they provide may need to evolve in the AI era.

4.2 Specification for and with AI

Ekaterina Komendantskaya (Heriot-Watt University – Edinburgh, GB), Thorsten Berger (Ruhr-Universität Bochum, DE), Mauricio Castillo-Effen (Lockheed Systems – Arlington, US), Marsha Chechik (University of Toronto, CA), Jyotirmoy Deshmukh (USC – Los Angeles, US), Matthew Dwyer (University of Virginia – Charlottesville, US), Taylor T. Johnson (Vanderbilt University – Nashville, US), Federico Mora (University of Waterloo, CA), Daniel Neider (TU Dortmund, DE)

This group addressed the dual challenge of specifying AI systems themselves and using AI to assist with specification tasks. For specifying AI, key problems included handling the inherent uncertainty in task definitions, understanding what constitutes a "bug" in AI systems, managing the distance between problem space (context, users) and embedding space, and dealing with AI's fundamentally different characteristics – built for vague problems, subject to novel attacks, operating in large state spaces that are hard to explore systematically. The group distinguished between approaches for small language models (built for single tasks, easier to specify and potentially verify) versus large language models, and discussed the gap between task-specific verification and the multi-task, multi-modal nature of modern AI systems.

For AI-assisted specification, the group explored how LLMs could help translate high-level intent to lower-level behavioral specifications, convert natural language to temporal logic, and translate between specification languages. Promising approaches included neuro-symbolic solutions, constraint-guided generation, tools like Amazon's Kiro for conversational specification elicitation, and the concept of LLMs as judges for tasks without conventional

oracles. However, fundamental questions remain about the role of humans in AI-assisted specification engineering, establishing notions of correctness (full versus partial, with or without certificates), handling the challenges of "vibe coding" where huge pull requests lead to knowledge loss, and managing the lifecycle from specification through manual code customization. The group emphasized the need for systematic frameworks, better traceability between high-level features and code artifacts, and principled approaches to collaboration between humans and AI in the specification process.

4.3 Usability of Specifications

Shahar Maoz (Tel Aviv University, IL), José Creissac Campos (University of Minho, PT), Reiner Hähnle (TU Darmstadt, DE), Yi Li (Nanyang TU – Singapore, SG), Alexandra Mendes (University of Porto, PT), Phillippe Palanque (Toulouse University, FR), Bernhard Rumpe (RWTH Aachen, DE), Kathryn T. Stolee (North Carolina State University – Raleigh, US), Harold Thimbleby (Swansea University, GB)

This breakout explored usability in the context of formal specification languages, tools, and methods through the lenses of established HCI frameworks such as ISO 9241 and Nielsen's usability criteria. Participants emphasized that usability spans learnability, efficiency, effectiveness, and user satisfaction, while user experience further encompasses emotional, aesthetic, and value-driven dimensions. A recurring theme was that specification work is a creative, cognitively demanding activity involving diverse users – domain experts, programmers, students, formal methods specialists – each with different needs and expectations. As a result, understanding usability requires identifying the tasks users perform (e.g., expressing properties, validating interpretations, refactoring expressions, checking consistency) and studying how tools can support these tasks.

The group identified research gaps in generalizing usability studies, understanding trade-offs inherent in designing notation, and supporting mental-model alignment through visualization, navigation, and incremental interaction. Existing frameworks – such as the cognitive dimensions of notations, the "physics of notations," and studies on programmer usability – offer valuable foundations but do not yet address the full complexity of specifying, debugging, or evolving formal models. Promising directions include supporting transformation and refactoring workflows, leveraging domain-specific or embedded DSLs to reduce cognitive distance for domain experts, and building persuasive interfaces that gently guide users toward sound practices. Participants also highlighted the need for benchmarks and representative tasks to systematically evaluate usability in specification contexts.

4.4 Specification Quality and Validation

Rômulo Meira-Góes (Pennsylvania State University – University Park, US), Alcino Cunha (University of Minho, PT), Eunsuk Kang (Carnegie Mellon University – Pittsburgh, US), Pamela Zave (Princeton University, US)

The breakout on Quality and Validation examined what it means for a specification to be "high-quality" and how one can systematically validate it. Participants noted that the field lacks shared definitions for quality attributes of specifications – attributes that range from those expressible purely in formal terms to those that also depend on informal understanding of the domain. Validation was characterized as ensuring correspondence between formal models and the real world or stakeholders' mental models, encompassing accuracy, completeness, and generality. The group emphasized that, although examples exist (e.g., testing formal models, scenario or predicate generation, Zave and Jackson's "turnstile" work), these methods have not sufficiently raised awareness of validation's importance or made validation accessible across domains.

The group discussed several barriers to adoption: validation is under-taught, often domain-specific, and many engineers undervalue domain modeling. Physics-based disciplines offer instructive analogies where domain constraints assist validation within error bounds, but formal specification practice rarely incorporates similar reusable frameworks. Looking ahead, the group identified machine learning and large language models (LLMs) as both a challenge and an opportunity. As tools increasingly use AI to generate, analyze, or validate specifications, the community must develop ways to validate the outputs of such tools – and potentially use LLMs themselves as aids for scenario generation, explanation, and model exploration. This, they argued, presents a strategic opening to reassert the importance of validation in the broader software engineering ecosystem.

4.5 LLMs for Specifications

Michael W. Whalen (Amazon Inc. – Minneapolis, USA & The University of Minnesota – Minneapolis, USA), Matthew Dwyer (University of Virginia – Charlottesville, US), Lars Grunske (HU Berlin, DE), Yi Li (Nanyang TU – Singapore, SG), Shahar Maoz (Tel Aviv University, IL), Rômulo Meira-Góes (Pennsylvania State University – University Park, US), Bernhard Rumpe (RWTH Aachen, DE)

This breakout explored how large language models can support the creation, transformation, and understanding of specifications. The group developed a taxonomy of LLM uses – ranging from generating and translating specification artifacts, to refining and repairing them, to assisting with semantic comparison and cross-validation. LLM strengths lie especially in bridging informal and formal representations, offering multiple interpretations of ambiguous requirements, and generating explanations that help humans understand complex artifacts. The group noted that LLMs may accelerate tasks such as formalizing requirements, migrating between specification languages, producing test cases, and supporting exploratory design-space analysis.

At the same time, participants emphasized that LLMs must be used within a carefully designed human–AI workflow. Humans remain essential for comprehension, validation, and selection among alternative candidates. Concerns included dependence on input quality (with code often yielding better results than natural language), sensitivity to bias when both code and specifications are generated by the same model, and the need for diversity across LLM sources to enable cross-checking. The group strongly rejected the idea of using LLMs for verification itself, instead positioning them as generators, explainers, and collaborators rather than judges. Open questions remain about whether traditional specifications are still needed in an era of "vibe-coding," how to effectively validate and select between LLM-generated alternatives, and optimal approaches for ensuring specification diversity while avoiding algorithmic bias.

4.6 Scalable Construction of Specifications

Michael W. Whalen (Amazon Inc. – Minneapolis, USA & The University of Minnesota – Minneapolis, USA), Alcino Cunha (University of Minho, PT), Eunsuk Kang (Carnegie Mellon University – Pittsburgh, US), Rômulo Meira-Góes (Pennsylvania State University – University Park, US), Jan Oliver Ringert (Bauhaus-Universität Weimar, DE), Allison Sullivan (University of Texas at Arlington, US), Pamela Zave (Princeton University, US)

This group tackled the fundamental challenge of constructing large, complex specifications from smaller components, examining issues of composition both within single formalisms and across heterogeneous languages and logics. Key problems included how to assemble proofs about different system elements into coherent whole-system arguments, how to facilitate specification reuse across systems and abstraction levels, and how to support incremental, modular changes while managing proof maintenance effort. The group also addressed methodological questions around distributing specification construction across engineering teams and developing specification product lines.

Critical scalability concerns extended beyond mere size to include verification time, incremental analysis capabilities, and the challenge of maintaining formal arguments as specifications evolve. The group discussed several technical approaches including contract-based decomposition, component-based architectures with information hiding (such as AADL with local component proofs), and the use of equivalence relations between abstraction levels. A recurring theme was the tension between code-level proofs of individual components and higher-level API abstractions, particularly when implementations may not perfectly refine their specifications, raising questions about how to compose partial correctness guarantees into system-level assurance arguments.

4.7 AI for Domain Modeling

Pamela Zave (Princeton University, US), Alcino Cunha (University of Minho, PT), Eunsuk Kang (Carnegie Mellon University – Pittsburgh, US)

This focused group examined the critical but often neglected practice of domain modeling, distinguishing sharply between productive and counterproductive uses of LLMs in this space. The group identified a "bad idea" – encouraging non-experts to do domain modeling with LLM assistance, since non-experts cannot properly evaluate LLM outputs – and a "good idea" – domain experts using LLMs as collaborative tools to help create quality domain models. For experts, LLMs can write domain models in natural or formal languages, answer validation questions, help get started, encourage continuation, and stimulate thinking. The group illustrated their approach with a practical example of validating assumptions in safety-critical systems, showing how LLMs can identify edge cases and failure scenarios that experts might overlook.

Three key research directions emerged: training data (determining whether to use large public models or small specialized ones, whether to restrict training to authoritative or proprietary data, and whether including diverse sources like science fiction might be valuable), prompt engineering (teaching experts to write well-structured domain models and validate them successfully, recognizing that prompt size and quality are crucial, and devising methods that guide experts through appropriate steps while encouraging independent critical thinking), and "meta-prompt-engineering" (asking the LLM how to write good prompts for domain modeling tasks). The emphasis throughout was on LLMs as tools to augment expert judgment rather than replace it.

5 Open problems

5.1 Let's Verify ChatGPT: What Would We Verify and How Could We Get There? (or Is Neural Network Verification Useful and What's Next?)

Taylor T. Johnson (Vanderbilt University – Nashville, US)

Main reference Taylor T. Johnson: "Is Neural Network Verification Useful and What Is Next?." Proceedings of the 61st Allerton Conference on Communication, Control, and Computing, 2025.
URL https://hdl.handle.net/2142/130315

Due to the advent of small language models (SLMs) especially for agentic artificial intelligence (AI), we suggest a challenge to verify a realistic foundation model (or large language model [LLM]) like ChatGPT. Could we verify ChatGPT, what would we verify, and how could we do it? We review recent results in neural network verification and recent scalability as demonstrated in the Verification of Neural Networks Competition (VNN-COMP), along with the limitations and usefulness of these approaches, detailed more in [1]. The neural network verification problem is given a trained neural network and a specification often formalized as preconditions and postconditions represented by sets in the input and output spaces of the neural network – prove the network satisfies the specification, and amounts to

showing the neural network maps any element of the precondition into the postcondition. More realistically, we suggest trying to verify a fully open SLM like Ai2's OLMo2 series (open code, data, weights, etc.) like OLMo2 1B, and this guiding grand challenge to verify an SLM (or LLM) will illustrate areas of need in formal methods for specification and verification of transformer architecture models.

References

1 Taylor T. Johnson, "Is Neural Network Verification Useful and What Is Next?", Proceedings of the 61st Allerton Conference on Communication, Control, and Computing, 2025.

5.2 Specification coherence

Harold Thimbleby (Swansea University, GB)

We take it for granted that formal methods and specifications are a good idea, but we don't always seem to have any good terminology to support clear and productive arguments. This paper introduces and defines the terms "coherence" and "views" as contributing to terminology to help focus and make such arguments clearer.

This paper motivates the term coherence with the example of thinking about helping developers achieve (more) dependable programs (the normal goal of formal methods and specifications), as well as its use in courts of law, where litigants want to achieve more dependable evidence.

Participants

- Thorsten Berger
Ruhr-Universität Bochum, DE
- José Creissac Campos
University of Minho, PT
- Mauricio Castillo-Effen
Lockheed Systems – Arlington, US
- Marsha Chechik
University of Toronto, CA
- Benoît Combemale
INRIA – Rennes, FR
- Alcino Cunha
University of Minho, PT
- Jyotirmoy Deshmukh
USC – Los Angeles, US
- Matthew Dwyer
University of Virginia – Charlottesville, US
- Lars Grunske
HU Berlin, DE
- Reiner Hähnle
TU Darmstadt, DE
- Taylor T. Johnson
Vanderbilt University – Nashville, US
- Eunsuk Kang
Carnegie Mellon University – Pittsburgh, US
- Ekaterina Komendantskaya
Heriot-Watt University – Edinburgh, GB
- Yi Li
Nanyang TU – Singapore, SG
- Shahar Maoz
Tel Aviv University, IL
- Rômulo Meira-Góes
Pennsylvania State University – University Park, US
- Alexandra Mendes
University of Porto, PT
- Federico Mora
University of Waterloo, CA
- Daniel Neider
TU Dortmund, DE
- Phillippe Palanque
Toulouse University, FR
- Jan Oliver Ringert
Bauhaus-Universität Weimar, DE
- Bernhard Rumpe
RWTH Aachen, DE
- Kathryn T. Stolee
North Carolina State University – Raleigh, US
- Allison Sullivan
University of Texas at Arlington, US
- Harold Thimbleby
Swansea University, GB
- Michael W. Whalen
Amazon Inc. – Minneapolis, USA & The University of Minnesota – Minneapolis, USA
- Pamela Zave
Princeton University, US