



On Teaching Applied Formal Methods in Aerospace Engineering

Kristin Yvonne Rozier^(✉) 

Iowa State University, Ames, IA, USA
kyrozier@iastate.edu

Abstract. As formal methods come into broad industrial use for verification of safety-critical hardware, software, and cyber-physical systems, there is an increasing need to teach practical skills in applying formal methods at both the undergraduate and graduate levels. In the aerospace industry, flight certification requirements like the FAA’s DO-178B, DO-178C, DO-333, and DO-254, along with a series of high-profile accidents, have helped turn knowledge of formal methods into a desirable job skill for a wide range of engineering positions. We approach the question of verification from a safety-case perspective: the primary teaching goal is to impart students with the ability to look at a verification question and identify what formal methods are applicable, which tools are available, what the outputs from those tools will say about the system, and what they will not, e.g., what parts of the safety case need to be provided by other means. We overview the lectures, exercises, exams, and student projects in a mixed-level (undergraduate/graduate) Applied Formal Methods course (Additional materials are available on the course website: <http://temporallogic.org/courses/AppliedFormalMethods/>) taught in an Aerospace Engineering department. We highlight the approach, tools, and techniques aimed at imparting a good sense of both the state of the art and the state of the practice of formal methods in an effort to effectively prepare students headed for jobs in an increasingly formal world.

1 Introduction

Verification is a fundamental engineering skill; the current surge toward autonomy and increasingly intelligent operation of hardware, software, and cyber-physical systems has changed how we need to apply, and teach, verification at the university level. Industrial aerospace systems, including avionics, commercial aircraft, Unmanned Aerial Systems (UAS), satellites, and spacecraft, are being pushed toward design-for-verification, e.g., by Model-Driven Engineering [16, 18, 37–40], Fault Detection, Isolation and Recovery (FDIR) [5, 14], and Runtime Verification [15, 23, 24, 33]. “Nowadays, it is well-accepted that the development of critical [aerospace] systems involves the use of formal methods,” [1].

Thanks to NSF CAREER Award CNS-1552934 for supporting this work.

In addition to the obvious need to train verification engineers and researchers developing new formal methods, we are faced with the need to train a wide range of engineers with basic skills like understanding the outcome of a formal methods analysis.

Through a mixed-level (undergraduate/graduate) course, we introduce students to the fundamentals of formal methods, which we define as a set of mathematically rigorous techniques for the formal specification, design, validation, and verification of safety-critical systems, of which aircraft and spacecraft are the prime example. The course explores the tools, techniques, and applications of formal methods with an emphasis on real-world use-cases such as enabling autonomous operation. Students build experience in writing mathematically analyzable specifications from English operational concepts for real systems, such as aircraft and spacecraft. Together, the class examines the latest research to gain an understanding of the current state of the art, including the capabilities and limitations of formal methods in the design, verification, and system health management of today's complex systems. Students leave with a better understanding of real-world system specification, design, validation, and verification, including why the FAA specifically calls out formal methods in certification requirements such as DO-178B [21], DO-178C [20], DO-333 [19], and DO-254 [22].¹

This course is intended to be a fun, interactive introduction to applying formal analysis in the context of real-world systems. We emphasize hands-on learning, through the use of software tools in homeworks and projects. Students learn the real tools used at NASA, Boeing, Collins Aerospace, Honeywell, Airbus, the Air Force, and others. Students from all areas of aerospace engineering, electrical and computer engineering, computer science, and other engineering disciplines, are encouraged to enroll. The course is cross-listed at the senior undergraduate/entry graduate levels and cross-listed in the Aerospace Engineering (AERE) and Computer Science (COMS) departments at Iowa State University and advertised in the Electrical and Computer Engineering and Mathematics departments; students from Industrial Engineering and Mechanical Engineering have also enrolled in this elective. Aiming for broad appeal, all concepts in the class are motivated chiefly through aerospace engineering applications; this shows direct applications to those students in the Aerospace Engineering Department and provides interesting use-cases for other majors. Example applications in homeworks include many different aspects of automated air traffic management and designing for autonomous operations of UAS.

Applied Formal Methods takes a *safety case* perspective [2, 7, 10, 16, 32]; in Aerospace Engineering, a safety case enables flight certification by providing an explicit statement of safety claims, a body of evidence concerning the system,

¹ Note that the railway industry has comparable standards CENELEC EN 50126 [8], EN 50128 [9], and EN 50129 [11]; these govern applications of formal methods in industrial rail systems, such as the success in verifying Paris' fully automatic, driverless Métro Line 14 (aka *Météor-Métro est-ouest rapide*) [3]. The course highlights railway, motor vehicle, medical, and other applications of industrial formal verification.

and an argument, based on the evidence, that the system satisfies its claims [31]. The major learning objective is for students to be able to read and understand, contribute to, and design an engineering system for being flight certified by a safety case, as this capability is now a general engineering skill. Students have the opportunity to construct a safety case as a half-semester final project for the course.

Learning Outcomes. Our central focus is to enable students to look at a problem, identify what we can verify, what information is needed to perform that analysis, how to validate the verification setup, and how to place the results in the field, e.g., by identifying what is now known, to what extent, and what is not known. Students learn to read research papers in formal methods, identify the current state of the practice, critically analyze current capabilities and limitations of the available tools and techniques, and effectively identify the inputs and outputs to verification, including what they really mean with respect to industrial safety standards. We specifically emphasize learning techniques for specification debugging and validation of mathematical models of systems. By the end of the course, students can identify what we can verify, and how; what can't we verify and why not; and what do we not have enough information to verify (and what additional information would we need). To construct an effective safety case, students must be able to recognize incomplete verification problems, identify ways to complete them, and identify assumptions and risks to validation.

Specific Learner Objectives. Through hands-on experience with formal methods tools and techniques, classroom discussions, homeworks, and projects students have the opportunity to learn to:

- Specify system requirements formally in Linear Temporal Logic (LTL) and Computational Tree Logic (CTL).
- Specify systems as formal models, i.e., models in a formal semantics.
- Apply model checking to system models and LTL specifications to determine if the models satisfy the specifications.
- Use tools popular in industrial verification labs, including explicit and symbolic model checkers, and theorem provers.
- Evaluate real-world systems to determine appropriate formal methods to use in their analysis.
- Evaluate system requirements, including determining if they are safety or liveness, and performing basic specification debugging.
- Analyze and draw conclusions about real-world systems regarding formal properties, understanding their significance and the inherent assumptions and limitations.
- Explain the principles underlying formal methods for different types of system analysis (e.g. design time versus runtime), the capabilities, and the limitations.
- Develop an understanding of the current state of the art and how to find formal methods tools for real-world use cases.

Prerequisites. The course requires the mathematical maturity and experience with proof structures covered in *Calculus II* (ISU MATH 166). Due to the cross-listing, the prerequisite is a disjunction of the Aerospace Engineering course *Computational Techniques for Aerospace Design* (ISU AERE 361) or the Computer Science *Algorithms* course (ISU COMS 311); both have MATH 166 as a prerequisite. Students should be familiar with first-order logic quantifiers and inductive proof techniques in order to understand Theorem Proving; professor permission enables registration for students who learned these skills in another 300-level course, e.g., from other engineering majors.

Organization. The remainder of this paper is organized as follows. Section 2 overviews the high-level approach to teaching Applied Formal Methods, including course assignments and examinations, highlights from the syllabus, and a general course schedule. We specifically pull out the tools and techniques covered in class in Sect. 3. Further details about the student research presentations and half-semester projects, including group projects, appear in Sects. 4 and 5 respectively. Section 6 concludes with an outlook toward continuous improvement of the course.

2 Approach

The first half (55–60%) of the course is a survey of the formal methods using modern tools, exemplified by case studies on industrial applications of formal verification. Class sessions are largely interactive and include discussions of the readings, guest speakers from industry, small group activities, and lecture. Students are encouraged to participate actively in class sessions. Lectures commence with “Formal Methods

Explained: what are formal methods, why do we need formal methods, and why don’t we formally verify everything?” The course proceeds to briefly review propositional logic and proofs. Class sessions cover in detail temporal logics, strategies for formal specification, specification debugging [27, 28], system modeling, explicit model checking [29], theorem proving [6], and symbolic model checking based on [25]. These are the topics covered by the midterm examination, given during normal class hours, covering the material from readings and homeworks from the first half of the course. The second half of the course requires only two assignments: an in-class presentation of a research paper of the student’s choosing, and a final project spanning the second half of the course, which serves in place of a final exam. All students are required to present the results of their final project mid-term and final results to the class and turn

Grade Component	Weight
Homeworks and Projects	30%
Midterm	25%
Research Paper Presentation	15%
Evaluation of Other Presentations	5%
Final Project	25%

Fig. 1. The weight assigned to each component: grades are assigned based on performance on homeworks, projects, presentations, a midterm exams, and a final project.

in a report including all artifacts required for reproducibility of their results at the end of the semester during the final exam period. Figure 1 summarizes the course assignments.

Other formal methods are discussed in class, included in in-class activities, demos or videos, and readings. Classes trace the relationships shown in Fig. 2.

Homeworks and Projects. All homeworks are distributed and collected via github classroom. Homeworks are required to be typed and formatted in L^AT_EX; some require submitting input files, e.g., for Spin, NUXMV, or PVS, solving the verification exercises. Students may talk about the problems with fellow students and the professor, but must submit individually-drafted write-ups. We occasionally discuss and work through parts of homework problems or variations thereof in class. When discussing with fellow students they must strictly follow the “empty hands policy:” one cannot leave a discussion meeting with any record of the discussion (hard copy or electronic). All scratch paper must be torn and thrown away and all boards erased. Homeworks are encouraged to include B^IB^TE^X references sections, including credit to collaborators and outside sources consulted. Students are encouraged to consult research papers, books, or other published materials in accordance with the University Honor Code (which prohibits searching for answers online, posting questions to internet forums, or discussing any assignments with others on the internet). All solutions should be written in each student’s own words, even if the solutions exist in a publication referenced in the homework bibliography. While we adjust the course schedule every semester, depending on the students’ backgrounds and the availability of guest speakers, a common schedule for the 16-week semester appears in Table 1.²

Reading Materials. Reading materials are included in the homeworks or otherwise distributed in class, e.g., research papers. There is no required textbook for this class. Two optional textbooks provide supplemental materials for students who desire additional reading, with the following caveats.

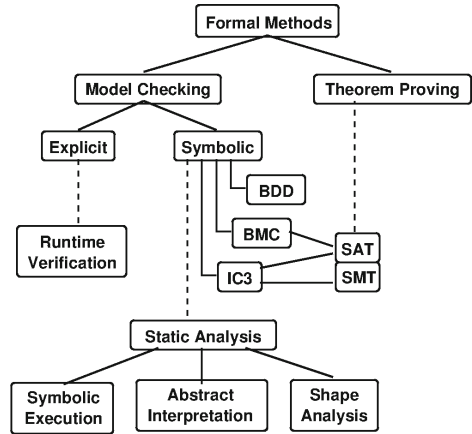


Fig. 2. The tree of Formal Methods, as presented in lectures; solid lines represent direct variations whereas dashed lines represent related derivations.

² In the U.S., there is usually a one-week break in the second half of the semester, after the mid-term project report presentations (Thanksgiving Break or Spring Break).

Table 1. A typical schedule for the homework assignments/small verification projects (top) comprising the survey of formal methods tools and techniques, along with the independent-research-based course assignments (bottom) across a 15-week semester with a following final exam period.

Week:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0: git	■																
1: PL	■	■															
2: TL		■	■	■													
3: Spec				■	■	■	■										
4: E-MC						■	■	■									
5: TP							■	■	■								
6: S-MC								■	■	■							
Midterm									■	■	■	■	■	■	■	■	■
Pres									■	■	■	■	■	■	■	■	■
P: Prop										■	■						
P: MP												■	■				
P: FP																■	■
P: Fin																	■

HW 0	github classroom and L ^A T _E X fundamentals; due 2 nd class period
HW 1	Propositional Logic: review of logic and proof structure; ~1 week
HW 2	Temporal Logic: LTL and CTL; ~1.5 weeks
HW 3	Classifying Specifications & Explicit-State Modeling in Spin; ~2.5 weeks
HW 4	Explicit-State LTL Model Checking in Spin; ~1 week
HW 5	Theorem Proving: exercises in PVS (or Isabelle); ~1 week
HW 6	Symbolic Model Checking with NUXMV; ~1 week
Midterm	Comprehensive exam covering all homework topics in the 9 th or 10 th week
Presentation	Choice of paper due concurrently with HW 6; research paper presentation and peer evaluations during class periods after midterm
Project (P)	Initial project proposal due immediately following midterm; mid-way presentation (MP) in front of the class 2-3 weeks later; final presentation (FP) during the last week of classes; final paper/verification artifacts (Fin) due during final exam period

Optional Textbook: *An Introduction to Practical Formal Methods Using Temporal Logic* [13]. Use this for:

- good background on LTL: well-formed formulas, semantics, encoding English sentences, expressivity, normal forms, relationship to automata
- reactive system properties: safety, liveness, fairness
- specification and modeling of real systems
- deciding the truth of a temporal formula; related proof techniques including explicit model checking
- thorough chapter on Spin, including how to run it from the command line and a good Promela tutorial

- review of classical and propositional logic
- extensions including synthesizing software from specifications

Be cautious that:

- LTL is instead called PTL in this book; that is non-standard
- LTL2BA is not the best tool; SPOT is superior now: <https://spot.lrde.epita.fr/>
- URLs provided are outdated (no longer active or superseded by the state of the art)
- Spin chapter refers to outdated `xspin` (though only briefly)

Optional Textbook: *Systems and Software Verification: Model-Checking Techniques and Tools* [4]. Use this for:

- supplemental material on temporal logics (LTL, CTL, CTL*)
- background on automata as system models
- review of explicit and symbolic model checking
- reachability, safety, liveness, deadlock-freeness, fairness
- overview of modeling abstraction methods
- out-of-date chapters on SPIN and SMV still have useful reviews of basic tool usage
- ideas for related formal methods, including timed automata models, additional tools

Be cautious that:

- *This book is extremely out of date!*
- LTL is the proper name for Linear Temporal Logic (book calls it PLTL)
- comparisons of LTL vs CTL/CTL* have been changed/been disproved
- SMV version described is no longer available; current tool is NUXMV
- Spin version described has been updated (`xspin` vs `ispin`)

3 Tools and Techniques

While homeworks include hands-on projects in Spin, NUXMV, and PVS (or Isabelle), several other tools and techniques are covered in lectures, demos, or in-class activities. These tools, plus the most popular selections from student-devised projects, are collected in Table 2.







The first half of the semester (before the midterm) lectures are predominantly taught with a combination of slides and in-class exercises, frequently involving the class breaking into two or three groups, each with their own whiteboard, and solving problems in competition, usually in the form of a game. Groups must convince the rest of the class of the correctness of their answers to receive game points. The winning team is often awarded a prize like NASA stickers or similar swag from a guest speaker. Sometimes the same problem is posed to all groups, and sometimes each group is assigned a different strategy to employ then discuss with the class. For example, lessons on temporal logic encodings

Table 2. Tools featured in different areas of the Applied Formal Methods course.




	<p>Spin Model Checker http://spinroot.com/ ✈️ ☞</p>
	<p>SPOT Produces Our Traces https://spot.lrde.epita.fr/ ♻️ ☞ (Optional for use in Spin-related homeworks)</p>
	<p>NUXMV Model Checker https://es-static.fbk.eu/tools/nuxmv/ ✈️ ☞</p>
	<p>PVS Theorem Prover http://pvs.csl.sri.com/ ✈️ (OR Isabelle) ☞</p>
	<p>Isabelle Theorem Prover https://isabelle.in.tum.de/ ✈️ (OR PVS)</p>
	<p>PRISM Model Checker http://www.prismmodelchecker.org/ ♻️ ☞</p>
	<p>Z3 SMT Solver https://github.com/Z3Prover/z3 ♻️ ☞</p>
	<p>R2U2 Runtime Verifier http://temporallogic.org/research/R2U2/ ♻️</p>
	<p>Dafny Language and Program Verifier http://rise4fun.com/dafny/ ♻️</p>

(continued)

Table 2. (continued)

	<p>CBMC (Bounded Model Checker for C and C++ programs) http://www.cprover.org/cbmc/</p> <p> </p>
	<p>Coq Proof Assistant https://coq.inria.fr/ [Book: <i>Formal Reasoning About Programs</i> http://adam.chlipala.net/frap/]</p> <p> </p>

Legend:

-  Required in homework assignments & covered thoroughly in class
-  Featured in in-class instruction or presentation by guest lecturer(s)
-  Utilized in student-selected final project(s)

involve dividing the class by their personal preferences into an LTL group, a CTL group, and an optional CTL* group (should anyone in the class feel most strongly about that logic). During this *Temporal Logic Showdown* (based in part on [35]), requirements in the form of English and/or figures (timelines, drawings, flowcharts, etc.) are posed to the class simultaneously. The first group to correctly encode the requirement in their logic and buzz in wins the points for the round. After that, encoding in the other logic (between LTL and CTL) earns half-points and the first team to buzz during that round in has the chance to steal those points by completing the correct encoding in the other team's logic and buzzing in before that team.

4 Research Paper Presentations

Each member of the class presents a research paper in applied formal methods to the class during the second half of the semester. A presentation consists of a slide presentation to the class covering the paper, and a discussion including the student's own analysis of its results. Students sign up for presentation times. The professor must approve all papers selected. Students can choose their papers from a provided list of papers or from a list of relevant publication venues. Alternatively, students may feel free to propose a paper on applying formal methods from any source for approval. Students evaluate the presentations of others for credit; anonymized summaries of the feedback of classmates are included in each student's evaluation. While the professor reads these evaluations, presentations are graded by the professor alone.

4.1 Professor's Presentation Evaluation Form

Students design their in-class research paper presentations according to the following evaluation criteria. Point values are listed in \llbracket s.

1. Did the presentation address the following aspects of the paper?
 - (a) $\llbracket 5$ What was the motivation given for the work? What problem was being solved or question was being answered?
 - (b) $\llbracket 5$ What was the product of the paper? How was the paper novel and what did it contribute to the field? What tools were used, problems were solved, and artifacts were created?
 - (c) $\llbracket 5$ Is the work in the paper *reproducible*,³ i.e. are all of the necessary artifacts available to redo the study, including any models, specifications, theorems, code, data, benchmarks, or other instruments used to complete the study described in the paper.
 - (d) $\llbracket 5$ Is the work in the paper *correct*, i.e. did the authors specifically address how that they know their work is correct or provide any evidence of correctness such as a proof or a comparison to known results?
 - (e) $\llbracket 5$ Is the work in the paper *buildable*, i.e. is the foundation laid in such a way that others in the future would be able to build on it, extend it, and utilize the results in a meaningful way to accomplish a different project?
 - (f) $\llbracket 5$ Is there future work? This can include both future work listed in the paper and ideas the student has for extending the work.
2. $\llbracket 10$ Did the presentation accurately overview the paper and the work presented therein, given the time limit? Did the student make an effort to fully understand the material and explain, if some piece is missing or not understandable, why that is the case?
3. $\llbracket 20$ Was the presentation clear? Did the student make an effort to present the materials clearly and instructively, not necessarily in the order of the paper? Did the student draw on additional sources to fill out the information and background knowledge required to understand the paper? Did the student draw figures or create ways of presenting the material clearly and fully aside from simply pasting in artifacts from the paper?
4. $\llbracket 15$ Did the student adequately cover background information and related work in an effort to enable him/herself as well as the class to understand the material being presented? Examples of doing this well might include the student reading and including material from some of the paper's citations or manuals for the tools used or otherwise including related background information to aid understanding of the material presented in the paper. These papers are short (usually about 15 pages) snapshots of single projects in formal methods and are meant to be read by practitioners familiar with the field and so usually do not include sufficient background information in the main text.

³ For further reference on how exactly to define reproducibility, correctness, and buildability, please refer to: Rozier, Kristin Yvonne, and Rozier, Eric. "Reproducibility, Correctness, and Buildability: the Three Principles for Ethical Public Dissemination of Computer Science and Engineering Research," In IEEE International Symposium on Ethics in Engineering, Science, and Technology, Ethics'2014, May 23–24, 2014 [26].

4.2 Student's Presentation Evaluation Form

Peer evaluations earn students participation credit and provide good feedback that is summarized, anonymized, and returned to their peers. Point values are listed in [1]s.

1. [2] What did you learn today? List at least three things you took away from today's class material.
2. [1] Was the presentation clear? What did you like about the way your classmate explained the materials to you? What constructive suggestions do you have to offer this classmate about how to present the material more clearly? (Your response will not be passed on to your classmate, however, an anonymized summary of all suggestions may be presented in class at the professor's discretion.)
3. [1] Was the content of the paper useful? Do you think the authors have contributed something that you or others might use or build upon in a future foray into formal verification? Why or why not?
4. [1] Is today's paper/formal method/topic something you think would be useful to examine in more depth in this class? Why or why not? Some paper topics may be covered in more depth following student presentations in the upcoming weeks; some may be earmarked for updating this class the next time it is taught.

5 Student Projects

In lieu of a final exam, students complete half-semester projects demonstrating their knowledge of applying formal methods. The high-level concept is simple: pick a system, pick a formal method, and successfully apply that method to that system. Students may work in groups of size one, two, or three. They are encouraged to discuss their proposal with the professor early and often; a formal project proposal is due mid-semester. Weekly progress reports, and a mid-term presentation to the class ensure steady progress while encouraging them to name their verification challenges and bring them up for discussion in class.

5.1 Initial Project Plan: Statement of Work

For the initial project plan, each person/group submits a statement of work that specifically addresses the following questions:

1. Define your group. Who are the members of your group? What is your group name?
2. Define the parameters of your project. What formal method are you using? What specifications will you verify? What system will you analyze?
3. What does a success look like for your project? For example, a successful model checking project will be able to demonstrate a system model, validation of that model, a set of temporal logic specifications, a set of model checking

runs checking the specifications against the model, and an analysis of the results. A successful theorem proving project will be able to demonstrate a set of (validated) theorems that automatically prove in an automated theorem prover and an analysis of the results of the proofs. A successful project in runtime monitoring will be able to demonstrate a set of specifications, a set of runtime monitors constructed from them, experimental results over many system runs demonstrating correct operation of the runtime monitors, and analysis of the results.

4. How will you demonstrate your analysis? In other words, answer all of the following questions that relate to your project:
 - What benchmarks will you use? Where will you get them from?
 - How will you demo your analysis (in the class?) (in your final report?)
 - How will you measure your results?
5. Remember to think about important logistics and organization elements. Each person/group will collaborate via a git repository shared with the professor. What will be the structure of your repo? How often should members check point models/specifications/documentation elements? If the project is a group project, how will the group coordinate? For a group, when will group meeting be? For a single-person project, what time have you scheduled each week to work on the project?
6. Provide a project timeline: for each week, list what you plan to accomplish that week. Be realistic and make backup plans! Your group will email the professor a (short) report at the end of each week with a project update according to your weekly plan. This email can be as simple as a statement that all tasks were accomplished that week, or as complicated as a detailed explanation why something did not work and how you have replanned to do an equivalent task. Weekly reports are due at 5pm on Fridays. This is your chance to get feedback on your progress and questions every week!

5.2 Progress Report and Preliminary Results

Provide a preliminary report from your group in the form of an in-class presentation of your results-so-far, making sure to **explicitly** answer the following questions:

- What parts of your project have you completed? Provide a bulleted list of work outputs to date.
- Provide an outline of your final report. What will the format be? What sections will you include? How do you plan to present any data and your analysis?
- What challenges have you encountered so far and how do you plan to overcome them? Provide a bulleted list of pairs {Challenge, Plan for action} to answer this section.
- Do you think you will need to change/modify/add to your project in any way? If so, make your case here. For example, if you have discovered that all of your specifications fail when analyzed against your system, what is your plan to modify the system and/or specifications?

5.3 Final Report and Presentation to the Class

Each person/group presents their project and results to the class during the last class periods. The time slots vary according to the size of the group. The final report from each person/group is due during the scheduled final exam period. The final report follows the outline and format described in the preliminary progress report. It includes the deliverables listed in the initial project plan/statement of work. Specifically, students should make sure to include the following:

- An abstract: succinctly summarize the final project setup and results.
- **All** models, specifications, code, or other artifacts needed to reproduce the work and re-run the verification tasks you completed for the project. **The professor must be able to re-run the verification procedure(s) followed.**
- Overview of the project including introduction, motivation, problem setup, and other information needed to understand the problem domain.
- Related work and background information, citing any resources used in the design and completion of this project.
- How was validation performed?
- What precisely was verified? What does it mean? How are the results significant?
- A complete verification analysis: results, performance of the tool(s) used, etc.
- A bibliography; Chicago Manual of Style (CMS) format is preferred.

The final report is cumulative; it needs to include *all* work done for the project in a complete report. Failure to include any of the required sections listed above results in losing points, even if the work was mentioned in class or in a presentation.

5.4 Example Student Projects

Students are encouraged to design final projects involving real-life systems of personal interest. Many students choose to form a project from the verification component of their graduate or undergraduate thesis research, or of a senior design or club project, such as creating a safety case for the launch of a student-designed CubeSat. Other popular categories of projects include designing tools to create instances of a game the student enjoys or to play such a game. Verification of autonomous driving or security scenarios from popular media, and “classic” projects (like verification of an elevator or traffic light protocol) have been proposed every semester. A competitive project category has emerged where two or three students all verify the same system from the same initial specification using a different favorite verification tool akin to an extended version of the VerifyThis⁴ competition, with additional creative judging criteria.

⁴ <https://www.pm.inf.ethz.ch/research/verifythis.html>.

Table 3 collects brief descriptions of student-designed final projects; in all cases, the size of the expected final deliverables scaled linearly with the number of students in the group and was adjusted for undergraduate vs graduate status. Several of the projects changed from the initial project proposal as the students ran into unexpected road blocks or discovered new tangents worth pursuing. Changes often stemmed from negative validation results, and ranged from minor adjustments in scope to major changes in the tools used (e.g., after being able to prove a construct could not be expressed in one tool), or problem objective. Accordingly, many of the final reports include thoroughly-explored negative results.

Table 3. A representative selection of student-devised final projects, 2015–2018.

Project description	#	U/G	Tool(s) used
Verify a lane-keeping module for autonomous cars. Starting with a road line detection algorithm, design a correct control algorithm, verify safety requirements using KeymaeraX and software implementation via CBMC, and validate including with real-world testing via augmenting the student’s own car	1	U	KeymaeraX, CBMC
Utilize explicit model checking to generate 3×3 magic square puzzles with unique solutions, and to solve a given 3×3 puzzle	1	U	Spin
Analyze a real system (the CySat Make to Innovate (M:2:I) undergraduate research project) under active development spanning multiple abstraction layers on a demonstration mission toward surveying near-Earth objects under NASA’s CubeSat Launch Initiative. Software and hardware verification that the ISU-designed flight computer meets mission reliability requirements	1	U	Spin, nuXmv
Verify the control of a tilt-wing medevac UAS designed by an ISU senior design team meets safety specifications	1	U	Spin
Generate attack graphs (structures representing all attack scenarios that an attacker can launch on a system) via a model-based approach with components/behaviors/defences/vulnerabilities and specification of security/resiliency properties. Iteratively model-check, disjuncting the previous counterexample to the current security property to generate acyclic attack graphs	1	G	AADL, Lustre, Jkind, AGREE
Model the ZigBee wireless protocol along with a collection of possible faults using OCRA for component based modeling, contract-based design and refinement, nuXmv for model checking of resulting transition systems, and xSAP for safety assessment and analysis	1	G	nuXmv, OCRA, xSAP
Use Spin to generate winning strategies for the Kartenspiele card game after a failed attempt with PVS	1	G	Spin, PVS

(continued)

Table 3. (*continued*)

Project description	#	U/G	Tool(s) used
Create a python library to parse mission-time linear temporal logic (MLTL), create an explicit state-space graph of a formula, display this with graphviz, and find a satisfying path through the graph, comparing two different search algorithms	1	G	N/A
Model a set of self-driving car intersection navigation scenarios and driving paths; use symbolic model checking to verify that the car always chooses a safe path. Generalize this to a maze solver, replicating previously-published experiments with TuLiP. Solve two small mazes using the GR(1)Py toolkit	1	G	NUXMV, TuLiP, GR(1)Py
Model, validate, and verify a set of traffic signaling algorithms using symbolic model checking. Scale the number of traffic lights to four per intersection and the number of successive intersections, varying properties like the timing of lights, max cumulative wait time, and max allowable queue length at a light. Compare performance for BDD, BMC, and IC3 back-ends	1	G	NUXMV
Define the formal operational semantics for a Simply Typed Message-passing Calculus (STMC) for software concurrency. Machine-checked proofs demonstrate the correctness of the message passing model including broadcasting, multicasting and guarded receive, and show the utility of the calculus by proving the properties guaranteed delivery of messages, the happens-before relation between the various actions, and the mover properties of the possible actions	1	G	Coq
Verify a vehicle-to-vehicle communication subsystem of an autonomous vehicle platooning system	1	G	Spin
Evaluate security of a Software Defined Network (SDN) model, including firewalls, a switch-level security feature to prevent malicious attacks, and a controller-level security feature to prevent DOS attacks by verifying invariants including reachability, isolation, loop freedom, no dead-ends	1	G	NUXMV
Formally analyze three security protocols: Needham-Schroder Public Key Protocol, Otway-Rees Protocol and Kerberos Protocol. Analysis of a protocol is targeted towards detection of attacks in the protocol and suggestive modifications to the protocol that can eradicate the attack detected	1	G	NUXMV
Two students compete to verify the same Traffic Alert and Collision Avoidance System (TCAS) [34]: will explicit model checking or symbolic model checking be the better formal method for this task? One employs Holzmann's suggestions for optimizing the Spin model, the other takes advantage of NUXMV's newer back-end search algorithms. The competition includes performance, ease-of-use, modeling language expressibility, and usefulness of counterexamples	2	G	Spin, NUXMV

(continued)

Table 3. (*continued*)

Project description	#	U/G	Tool(s) used
Verify a python implementation of an A*-based pathfinding algorithm for a robot avoiding obstacles to traverse a maze via a shortest path using Linear Temporal Logic MissiOn Planning (LTLMoP). Validation included representing the same model in multiple tools and cross-validating model behaviors	2	U/G	(Py)NuSMV, PRISM, LTLMoP
Solve chess puzzles (puzzles over the pieces and rules of chess) via model checking focusing first on the mate-in-one-move problem	2	G	Spin
Verify a Mars rover mission sequence including coordination of a launch vehicle, ejection of the rover, executing a landing sequence, and commencing ground operations; confirm that mission goals are upheld including when faults occur and mitigation plans are executed	2	G	nuXMV
Explore the level of privacy maintained by users despite datamining, first through replication of a study on formal verification of privacy constraints on loan applications, then by devising a scalable model of e-voting machine data with user-specified privacy settings. An unsuccessful venture in Coq was followed by a successful re-imagining of the project using nuXMV	3	U ² /G	nuXMV, Coq
Compositionally verify an autonomous drone racing system with dissimilar components: localization (PVS), path planning (mCRL2), and the high-level architecture (Belief-Desire-Intent programming in AgentSpeak using Jason, Spin). Each student leads the verification of one subcomponent; ultimately the effort was unsuccessful due to integration challenges	3	U/G ²	PVS, mCRL2, Spin
Three students compete using three different tools to solve the same verification challenge (a Rubik’s cube) and compare their results, performance, and which parts of the problem were easier/harder with each tool; creative methods of cross-validation took advantage of overlap between tools, e.g., nuXMV and MiniSat. Models started with $2 \times 2 \times 2$ cubes and scaled the difficulty and size of the cubes	3	G	Spin, nuXMV, CBMC, MiniSat and CaDiCaL SAT solvers
Model and verify a realistic subsystem of UTM (UAS Traffic Management) for near mid-air collision (NMAC) avoidance based on [12, 17, 30, 36]. Use nuXMV to verify preflight, enroute, and emergency situations; further explore properties of enroute (like probability of a route change to avoid an NMAC) using PRISM	3	G	nuXMV, PRISM

Legend:

- # Number of students in the group
- U All students in group are undergraduate students
- G All students in group are graduate students

6 Conclusions and Outlook

In post-course surveys, students overwhelmingly identified details of tool use to be the aspect of the course they struggled with most; this includes the challenge of exposure to multiple new modeling/specification languages, details of tool installation/setup/debugging, and the gap between the level of detail required by formal methods tools versus their previous experiences, e.g., with pencil-and-paper proofs and informal (or no) system requirements. The majority of students identified the theorem proving tool (either PVS or Isabelle) as the most difficult to learn. When asked in hindsight (a year or more after course

completion) what aspect(s) of the course turned out to be most useful, nearly every part of the course was listed by some student. The course project and the survey of formal methods were each identified by over half of the former students as most useful, citing in particular the perspective gained through experience. Other popular responses include the students' sound theoretical understanding of formal methods, the comparative discussions of specification languages, and in-class exercises (which some students felt so strongly about they questioned the ability to scale the course to include more students or online students). Several students particularly appreciated learning about the (ab)use of SAT solvers for a variety of applications including scheduling, specification debugging, and reduction of other problems to SAT. Nearly every student surveyed, both during the course and in hindsight, wrote an impassioned essay about the paper presentation section of the course, including the value of individualized feedback from the professor and other students, the opportunity to improve their analysis/presentation skills, exposure to the breadth of research frontiers and case studies in formal methods, the perspective they gained on verification in the wild, and the ability to steer the topics of the second half of the course to match the class' interests.

When asked how the course could be improved, students have overwhelmingly focused on small details of individual exercises; this feedback is continuously used to improve lectures, slides, and assignment descriptions. Examples include more in-class demonstrations of the quirkier aspects of tools, more details on industry standards requiring formal methods, and more information on community resources such as the active mailing lists for many tools, especially Isabelle and PVS. Students have requested add-on or follow-on courses such as a research paper reading group that offers an expanded version of the paper presentation portion of the class, and a large-scale application option where students work in groups to verify a real system over a whole semester simulating an industry setting. This is consistent with the most-requested course improvement: each semester students request more information on the end-to-end formal verification process, such as a universal flow-chart with all of the aspects of verification from initial conception to system maintenance laid out in fine detail.

Applied Formal Methods is currently taught as an elective; it counts toward one required technical elective for undergraduate and graduate students in Aerospace Engineering, Computer Science, and Computer Engineering, and has (so far) always been approved for replacing technical electives in other areas of engineering. Going forward, we look to integrate it as a required course in a track, e.g., in an avionics or intelligent systems concentration or minor within aerospace or in a cybersecurity or other interdisciplinary major. At its current size of 12–20 students per semester, the high level of participation and multiple presentations by each student in the course is both practical and advantageous: each student can participate actively in the course and receive personalized instruction in applying formal methods to a project tailored to her/his interests. Maintaining learning outcomes while potentially scaling the class to a larger size will be a formidable challenge. End-of-semester student ratings of the course have been

consistently very high; if the course becomes required instead of purely elective, some adjustments may have to be made to accommodate a broader audience with more diverse interests.

One goal of publishing materials on the course is to receive feedback that can lead to continuous improvement; another is to open course materials for others to use and build upon. As formal methods teaching at the undergraduate and beginning graduate levels becomes more widespread there may be enough materials across the teaching community to support a tool-wise central repository of exercises, exam questions, and other teaching resources. We hope to contribute to such a repository, especially for tools like NUXMV and Spin, which remain popular for student use. Such materials could also be used to create industrial courses, such as the PVS Course at NASA Langley research center. We are continuously looking for industrial guest speakers to visit or give virtual lectures on their experiences applying formal methods in industrial practice. Traditionally, these lectures have received rave reviews and resulted in extra students showing up to class, in addition to those enrolled in the course. We hope to build up a club of regular industrial guest speakers as well as new lecturers to continue to inspire future students to apply formal methods in practice.

Acknowledgments. Information on our recent work can be found at: <http://laboratory.temporallogic.org>. Thanks to the Aerospace Engineering departments at Iowa State University and the University of Cincinnati for their forward thinking in recognizing the need to develop such a course. AERE/COMS 407/507 was developed over the Spring 2017, and Fall 2017 and 2018 semesters at ISU; parts of the class were first developed during the Spring 2015 and 2016 semesters at UC. Thanks to all of the students who actively participated in those courses, especially for coming up with such fantastic half-semester projects. Some course materials were inspired by or directly derived from The TeachLogic Project (<https://www.cs.rice.edu/~tlogic/>); special thanks goes to Ian Barland, John Greiner, and Moshe Vardi for their brilliant teaching tools. Thanks to the NASA Langley Formal Methods Group for providing an excellent PVS course both in-person [6] and online with a rich collection of regularly-updated teaching materials. (<https://shemesh.larc.nasa.gov/PVSClass2012/>). Thanks to the many guest speakers including: Nikolaj Bjørner, Jonathan Hoffman, Yogananda Jeppu, César Muñoz, Lucas Wagner.

References

1. Ameur, Y.A., Boniol, F., Wiels, V.: Toward a wider use of formal methods for aerospace systems design and verification. *Int. J. Softw. Tools Technol. Transf.* **12**(1), 1–7 (2010)
2. Basir, N., Denney, E., Fischer, B.: Constructing a safety case for automatically generated code from formal program verification information. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 249–262. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87698-4_22
3. Behm, P., Benoit, P., Faivre, A., Meynadier, J.-M.: Météor: a successful application of B in a large project. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 369–387. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48119-2_22

4. Bérard, B., et al.: Systems and Software Verification: Model-checking Techniques and Tools. Springer, Heidelberg (2013). https://www.amazon.com/Systems-Software-Verification-Model-Checking-Techniques/dp/3642074782/ref=sr_1_1?ie=UTF8&qid=1483572091&sr=8-1&keywords=systems+and+software+verification
5. Bittner, B., et al.: An integrated process for FDIR design in aerospace. In: Ortmeier, F., Rauzy, A. (eds.) IMBSA 2014. LNCS, vol. 8822, pp. 82–95. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12214-4_7
6. Butler, R., et al.: NASA/NIA PVS Class 2012. NIA, Hampton, Virginia, USA, October 9–12 (2012). <https://shemesh.larc.nasa.gov/PVSClass2012/online.html>
7. Butler, R., Maddalon, J., Geser, A., Muñoz, C.: Simulation and verification I: formal analysis of air traffic management systems: the case of conflict resolution and recovery. In: Proceedings of the 35th Conference on Winter Simulation: Driving Innovation, pp. 906–914. Winter Simulation Conference (2003)
8. CENELEC, EN50126: Railway applications-the specification and demonstration of reliability. Availability, Maintainability and Safety (RAMS) (2001). <https://www.cenelec.eu/standardsdevelopment/ourproducts/europeanstandards.html>
9. CENELEC, EN50128: Railway applications-communication, signaling and processing systems-software for railway control and protection systems (2011). <https://www.cenelec.eu/standardsdevelopment/ourproducts/europeanstandards.html>
10. Denney, E., Pai, G., Pohl, J.: Heterogeneous aviation safety cases: integrating the formal and the non-formal. In: 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems, pp. 199–208. IEEE (2012)
11. EN50129, CENELEC: Railway applications-communication, signalling and processing systems-safety related electronic systems for signalling. British Standards Institution, United Kingdom. ISBN, pp. 0580–4181 (2003)
12. von Essen, C., Giannakopoulou, D.: Analyzing the next generation airborne collision avoidance system. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 620–635. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_54
13. Fisher, M.: An introduction to practical formal methods using temporal logic, vol. 82. Wiley Online Library (2011). https://www.amazon.com/Introduction-Practical-Formal-Methods-Temporal-ebook/dp/B005E8AID2/ref=sr_1_1?ie=UTF8&qid=1483648485&sr=8-1&keywords=practical+formal+methods+using+temporal+logic
14. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: automated air traffic control design space exploration. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_1
15. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 215–230. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_18
16. Guarro, S., et al.: Formal framework and models for validation and verification of software-intensive aerospace systems. In: AIAA Information Systems-AIAA Infotech@ Aerospace, p. 0418 (2017)
17. Kochenderfer, M.J., Chryssanthacopoulos, J.: Robust airborne collision avoidance through dynamic programming. Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371 (2011)

18. Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., Rozier, K.Y.: Comparing different functional allocations in automated air traffic control design. In: Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2015), Austin, Texas, USA. IEEE/ACM, September 2015
19. Radio Technical Commission for Aeronautics: DO-333 – formal methods supplement to DO-178C and DO-278A (2011). <https://www.rtca.org/content/standards-guidance-materials>
20. Radio Technical Commission for Aeronautics: DO-178C/ED-12C – software considerations in airborne systems and equipment certification (2012). <https://www.rtca.org/content/standards-guidance-materials>
21. Radio Technical Commission for Aeronautics (RTCA): DO-178B: Software considerations in airborne systems and equipment certification, December 1992
22. Radio Technical Commission for Aeronautics (RTCA): DO-254: Design assurance guidance for airborne electronic hardware, April 2000
23. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 357–372. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_24
24. Rozier, K.Y., Schumann, J., Ippolito, C.: Intelligent hardware-enabled sensor and software safety and health management for autonomous UAS. Technical Memorandum NASA/TM-2015-218817, NASA, NASA Ames Research Center, Moffett Field, CA 94035, USA, May 2015
25. Rozier, K.: Linear temporal logic symbolic model checking. *Comput. Sci. Rev. J.* 5(2), 163–203 (2011). <https://doi.org/10.1016/j.cosrev.2010.06.002>
26. Rozier, K., Rozier, E.: Reproducibility, correctness, and buildability: the three principles for ethical public dissemination of computer science and engineering research. In: IEEE International Symposium on Ethics in Engineering, Science, and Technology, Ethics 2014, pp. 1–13. IEEE, May 2014
27. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73370-6_11
28. Rozier, K.Y., Vardi, M.Y.: A multi-encoding approach for LTL symbolic satisfiability checking. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 417–431. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_31
29. Rozier, K.Y., Vardi, M.Y.: Deterministic compilation of temporal safety properties in explicit state model checking. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC 2012. LNCS, vol. 7857, pp. 243–259. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39611-3_23
30. NASA UTM Research Transition Team (RTT): NASA UTM NextGen concept of operations v1.0, May 2018. <https://utm.arc.nasa.gov/docs/2018-UTM-ConOps-v1.0.pdf>
31. Rushby, J.: A safety-case approach for certifying adaptive systems. In: AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference, pp. 1–16 (2009)
32. Rushby, J.: Logic and epistemology in safety cases. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 1–7. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_1

33. Schumann, J., Moosbrugger, P., Rozier, K.Y.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 233–249. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_15
34. U.S. Department of Transportation Federal Aviation Administration: Introduction to TCAS II version 7.1, February 2011. hQ-111358. https://www.faa.gov/documentlibrary/media/advisory_circular/tcas%20ii%20v7.1%20intro%20booklet.pdf
35. Vardi, M.Y.: Branching vs. linear time: final showdown. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 1–22. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45319-9_1
36. Wei, P., Atkins, E., Schnell, T., Rozier, K.Y., Hunter, G.: NSF PFI:BIC: pre-departure dynamic geofencing, en-route traffic alerting, emergency landing and contingency management for intelligent low-altitude airspace UAS traffic management, July 2017. https://www.nsf.gov/awardsearch/showAward?AWD_ID=1718420
37. Wiels, V., Delmas, R., Doose, D., Garoche, P.L., Cazin, J., Durrieu, G.: Formal verification of critical aerospace software. *AerospaceLab* (4), 1–8 (2012). <https://hal.archives-ouvertes.fr/hal-01184099>
38. Zhao, Y., Rozier, K.Y.: Formal specification and verification of a coordination protocol for an automated air traffic control system. In: Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012). *Electronic Communications of the EASST*, vol. 53. European Association of Software Science and Technology (2012)
39. Zhao, Y., Rozier, K.Y.: Formal specification and verification of a coordination protocol for an automated air traffic control system. *Sci. Comput. Program. J.* **96**(3), 337–353 (2014)
40. Zhao, Y., Rozier, K.Y.: Probabilistic model checking for comparative analysis of automated air traffic control systems. In: Proceedings of the 33rd IEEE/ACM International Conference On Computer-Aided Design (ICCAD 2014), San Jose, California, USA, pp. 690–695. IEEE/ACM, November 2014