SE-5302: Formal Methods

Course Coordinator: J. Chandy

Lecturers: J. Chandy, P. Duggirala

Course Objectives:

This course is designed to provide students with an introduction to formal methods as a framework for the specification, design, and verification of software-intensive embedded systems. Topics include automata theory, model checking, theorem proving, and system specification. Examples are driven by control systems and software systems.

The course is addressed to students in engineering who have had at least a year of software or embedded systems design experience and who are pursuing the Embedded Systems curriculum track.

Anticipated Student Outcomes

By the end of SE-5302, students will have achieved the following objectives:

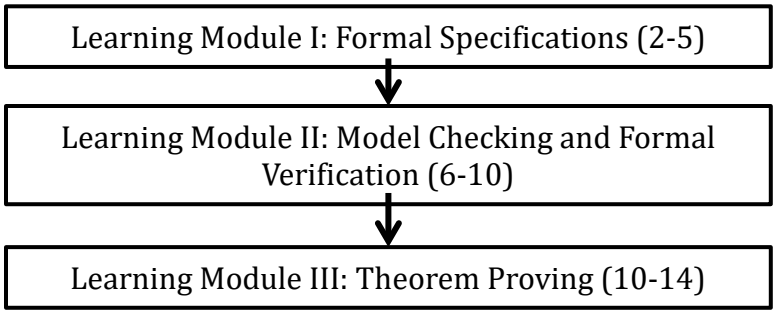
1) Gain familiarity with current system design flows in industry used for embedded system design, implementation and verification;
2) Learn what formal methods are and how they are used in embedded systems design?
3) Learn how to translate informal requirements to formal specifications
4) Learn languages for formal specifications and the applicability and appropriateness of various language choices for expressivity and efficiency.
5) Learn how formal specifications and formal methods can be used in verification
6) Learn what model checking is and how it can be used in embedded systems verification
7) Learn the theory behind SAT solvers, SMT solvers, and bounded model checking
8) Learn how model checking can be used for real timed, continuous, and hybrid systems
9) Learn about program analysis – both static and dynamic
10) Learn about theorem proving and its use in embedded systems verification
11) The course is intended to serve as a key component to achieve standard work proficiency levels L2-L3 in embedded system design.

Pre-requisites

SE 5301

Course Organization

The course is organized into three learning modules:



Structuring of these 3 learning modules into 14 lectures of a one semester course, along with the topics and references, is included below. In the following, each entry labeled "Lecture" corresponds to 2h: 30m of teaching. The lectures may be split into two 1h:15m sessions or include a 30m break in the middle. All lectures will be recorded to be viewed later if the scheduled lecture time does not work.

Learning Module I: Formal Specification (Lectures 2-5)

This module is meant to introduce students to formal specifications and its use in embedded systems design and verification. The module starts with a review and introduction of background concepts that provide the mathematical foundation of formal specifications followed by a lecture on formal models and semantics. The next lecture is focused how to translate requirements to formal specifications and an introduction to languages (such as temporal logic) to express these specifications. The module closes with a lecture on the application of formal specifications and formal methods to verification as a lead in to learning modules II and III. The module will rely on use cases and examples from the relevant industry.

Learning Module II: Model checking and Formal Verification (Lectures 6-10)

This module introduces students to model checking and its use in embedded systems design and verification. The module starts with an introduction of model checking concepts and an overview of the variety of tools available for model checking. The next two lectures dive into to theory of model checking with a discussion of SAT solvers, SMT solvers, and bounded model checking. The next lecture discusses the construction of models for timed, continuous, and hybrid systems, and how to use these models in the verification of these systems. The module closes with a lecture on program analysis and the information gained by doing static analysis vs. dynamic analysis.

Learning Module III: Theorem Proving and Review (Lectures 11-14)

This module introduces basic theorem proving paradigms and its use in embedded systems design and verification. The module will start with an introduction to proofs and what it means for a function to be correct. This is followed by lectures on various prover techniques including inductive invariants, liveness

proofs, and others. The module will introduce students to a wide variety of tools that can be used in practical theorem proving.

Topics listed in orange are advanced and may be skipped based on feedback.

Course Outline

Lecture 1: Introduction to Product Development Processes for Embedded Systems

Reading:

Alur, Chapter 1

Topics:

- Current industry product development process
- Current process enhanced by platform-based design principles
- Review of Embedded Systems Abstractions and Models
- Introduction to formal methods
    - What can we do?
    - What can we not do?
- Where does this course fit in: Requirements, Requirements Compliance
- Verification vs. validation
- Comparisons to other verification methods – simulations, review, etc.
- Regulations in the aerospace
- Domain of applications

Assignment:

Using a Simulink/Stateflow example (such as this elevator example: http://www.mathworks.com/help/stateflow/examples/modeling-an-elevator-system-using-atomic-subcharts.html or others from here: http://www.mathworks.com/help/stateflow/examples.html), simulate the model. What are some informal requirements that the system should obey? What is the difference between verification and validation?

Lecture 2: Background Material

Reading:

Aho and Ullman, Chapter 12 (particularly 12.2, 12.3, and 12.4, remainder may be skimmed, except skip 12.6, as they will come up again later in the course) and Chapter 14 (14.1, 14.2, 14.3, 14.4)

Topics:

- Motivation of background material with the formal verification problem

- Background and preliminaries
- Set theory, set builder notation, predicates (reference: Aho and Ullman, Chapter 7, particularly 7.1, 7.2, 7.3, 7.7, and 7.10)
- Induction (reference: Aho and Ullman, Chapter 2.3, 2.4, and 2.6)
- Propositional logic/calculus (reference: Aho and Ullman, Chapter 12)
- Predicate/first-order logic (reference: Aho and Ullman, Chapter 14)
- Higher-order logics
- Duality between formulas and sets of states

Assignment:

Reinforcing propositional logic and first-order logic with the Z3 SMT solver. In this assignment, students will use an SMT solver to reinforce propositional logic and first-order logic proofs. Each step of a proof will be shown to imply a subsequent step until the desired goal theorem is proven.

Lecture 3: Formal Models and Semantics

Reading:

Alur, Chapters 2 and 4

Topics:

- Formal models
  - Labeled transition systems
  - Parallel compositions (synchronous vs. asynchronous)
- Formal semantics
  - Operational semantics with executions
  - Executions, invariants, traces
  - Abstract and concrete specifications, and the notion of implementation
  - Trace inclusion: abstraction (simulation, implementation) functions/relations
- Case studies on toy systems / programs
  - Composition of a toy system

Assignment:

Using a Simulink/Stateflow example (such as this elevator example: http://www.mathworks.com/help/stateflow/examples/modeling-an-elevator-system-using-atomic-subcharts.html or others from here: http://www.mathworks.com/help/stateflow/examples.html), derive a formal model (e.g., as an extended state machine or composition thereof with transition system semantics) of the discrete-time Stateflow (i.e., software) components. Show executions, traces, and invariants of the model.

Lecture 4: Formal Specifications

Reading:

Alur, Section 3.1, Section 5.1

Topics:

- Informal specifications
- Safety and liveness specifications
- Formal specification languages
- Translating informal specifications to formal specifications
- Formal verification of formal specifications
- Temporal logic: LTL, CTL, PSL
    - How do you select which one?
    - Expressiveness of temporal logics: comparisons between LTL, CTL, CTL*
    - Tradeoffs with respect to model checking algorithms and efficiency
    - Infinite traces
    - Tie specification languages to capabilities of algorithms
    - Examples
- Complexity
- Formal specifications
    - Applicable problems
    - Requirements modeling of hardware and software systems for formal methods
    - State-based and transition-based specifications (redux?)
    - Functional specifications
- Data structures for specification languages: abstract syntax trees (ASTs), etc.
- Specification languages for tools
    - NuSMV
    - C: ACSL / Frama-C
    - Promela assertions (for SPIN)
    - Z-notation, Statechart, ASL
    - Others: Java: JML, Alloy, Spec#

Assignment:

Using a Simulink/Stateflow example (such as this elevator example: http://www.mathworks.com/help/stateflow/examples/modeling-an-elevator-system-using-atomic-subcharts.html), derive informal specifications for the discrete-time Stateflow (i.e., software) components. Formalize the specifications in one of the languages discussed. Classify the specifications into broad classes such as safety and liveness specifications. Theory questions: Expressiveness of specification languages: if possible, find a specification that may be expressed in LTL that may not be expressed in CTL, and vice-versa. If it is not possible, prove this.

Lecture 5: Formal Verification of Formal Specifications for Formal Models

Reading:

Alur Sections 3.2 and 5.2 (skim 5.2 for now)

Topics:

- Overview and examples of formal verification given models and specifications
  - Types of formal verification: automatic vs. manual, etc.
  - Overview of methods (model checking, abstract interpretation / static analysis, theorem proving, etc.)
- Abstract model checking algorithms as examples of formal verification
  - Reachability algorithms for verifying safety specifications / invariants
  - LTL model checking algorithm
- Case studies on real systems
  - Composition of a real system
- Advanced topics in specification
  - Real-time temporal logics: STL, MITL, …
  - Probabilistic temporal logics in Prism (PCTL, CSL, PLTL, PCTL*)
  - Fairness assumptions

Assignment:

Pick one of 5 possible papers with recent successes of formal methods. Write a report on benefits, challenges, limitations, etc. of using formal methods in actual engineering design. Are formal methods most useful as an engineering design process tool (the act of formalization is the main benefit?) or in really finding bugs / ensuring designs meet specifications (showing there are no bugs)?

Lecture 6: Model Checking Overview

Reading:

Alur, Sections 3.3, 3.4 (except 3.4.3 on ROBDDs), 5.2, NuSMV Manual

Topics:

- Model checking examples
- Fundamental concepts
  - What can model checking do and what it cannot do?
- State space explosion
- State representations: symbolic vs. explicit
  - CNF vs. DNF, other normal forms like PNF
  - Equivalence between formulas in different normal forms and algorithms for translation between, tradeoffs thereof (exponential blowups)

- Explicit state model checking
- Fixed-point computations
- Automata theoretic model checking and model checking as language inclusion
- Symbolic model checking
- Data structures and engines
    - Binary decision diagrams
    - OBDDs
- Invariant checking with reachability analysis
- Model checking tools
    - NuSMV / nuXmv
    - Spin
    - Murphi
    - Maybe TLA+/TLC

Assignment:

Use NuSMV to verify various examples, requiring model modification, specification modifications, etc.

Lecture 7: Satisfiability (SAT) and Satisfiability Modulo Theories (SMT)

Reading:

TBA

Topics:

- Review of normal forms like CNF, DNF, PNF, etc.
- Boolean Satisfiability (SAT) Problem
    - NP-Completeness of 3SAT
- SMT Problem
- Quantified problems (QBF, etc.)
- What can be encoded as SAT, SMT, and QBF problems?
- SAT solvers
    - DPLL
- SMT solvers
    - DPLL(T)
- Tools
    - Z3: main choice
    - Others to mention:
        - Why3
        - Yices

Assignment:

Use Z3 to generate test cases for code represented in SSA form. Write Z3 tactics to convert between CNF / DNF.

Lecture 8: Bounded Model Checking, Abstractions, and Advanced Topics

Reading:

Alur Chapter 5.2, NuSMV Manual

Topics:

An obvious challenge is how to represent problem as something model checker can solve, referred sometimes as the art of modeling. How do we simplify / abstract / model the problem, including the system and the specifications? How can we use the right compositional framework to set up whatever you need to prove?

- Abstract bounded model checking (BMC) algorithms
- BMC with SAT
- BMC with SMT
- Abstractions
- Refinements
- CEGAR
- Assume-guarantee reasoning and compositional reasoning
- Mention state of software model checking / verification
    - CBMC
    - JavaPathfinder
    - VCC
    - Frama-C
    - SLAM
    - BLAST
    - Intermediate verification languages
        - Boogie: http://research.microsoft.com/en-us/projects/boogie/

Assignment:

Use Z3 to implement a basic BMC algorithm.

Lecture 9: Timed, Continuous, Hybrid, and Probabilistic Verification

Reading:

Alur, Chapter 7

Topics:

- Models for timed, hybrid, and probabilistic systems and semantics thereof
  - Executions of timed and hybrid automata
  - Probabilistic models
- Uppaal
- Reachability analysis: SpaceEx
- Prism
- Simulation-based verification

Assignment:

Use Uppaal to verify some timed examples. Use SpaceEx to verify safety properties by computing reachable states for some examples.

Lecture 10: Static and Dynamic Analysis

Topics:

- Layered formal methods: different techniques for different abstraction levels
- Types of static analysis
  - Information provided by static analysis
- Conformance to code standards with static analysis (MISRA, etc.)
- Static vs. dynamic analysis
- Dynamic specification creation using dynamic analysis
- Tools: Frama-C, Lint

Assignment:

Formalize specifications in ACSL and use Frama-C to check them for some C program examples.

Lecture 11: Theorem Proving Intro and Proofs using Inductive Invariance

Reading:

Aho and Ullman, Chapter 12 (review sections 12.7, 12.8, read 12.9, 12.10, 12.11), Alur, Chapter 3.2 (review) and 5.3 (skim)

Topics:

- Meaning of proofs
  - 4-color theorem, other more recent successes
- Summarize state of recent research results leveraging theorem proving
  - seL4
  - CompCert
- Automated and interactive theorem proving

- Theorem proving tools (one in depth, mention others)
    - PVS or Isabelle
    - Mention
        - Coq
        - TLA+
- Modeling systems in theorem provers
- Inductive invariants
    - Relation to strongest postcondition / weakest precondition
    - Relation to invariants

Assignment:

Verify invariants in distributed algorithms in Isabelle.

Lecture 12: Liveness Proofs and Other Proof Rules

Reading:

Alur, Chapter 5.3

Topics:

- Ranking functions
- Loop invariants (reference: Aho and Ullman, Chapter 2.5 and 2.9)
- Weakest precondition, strongest postcondition
- Operational vs. axiomatic vs. denotational semantics
- Other theorem prover features (model checker integration, etc.)

Assignment:

Verify liveness in distributed algorithms in Isabelle.

Lecture 13: Abstract Interpretation

Reading:

Nipkow and Klein, Chapter 13

Topics:

- Functional correctness
- Meaning of correctness
- Verified Design by Contract
- Abstract interpretation tools
    - T2 is open: https://github.com/mmjb/T2
- Theorem prover integration for C programs
    - Frama-C / Why3

Assignment:

Use Isabelle to verify properties for C code (using appropriate C-to-Isabelle parsers), a la what's done in seL4.

Lecture 14: Comprehensive Embedded System Verification Case Study

Topics:

- Demonstrate detailed, comprehensive example of verification in an embedded system case study leveraging all the components of the course
- Capabilities of embedded systems design environments
    - Simulink/Stateflow
        - Static analysis with Polyspace
    - Esterel / Lustre ?
- Verification of controller-plant interaction using SpaceEx
- Verification of software implementation of controller using Frama-C
- Wrap up, review

Assignment:

Take home final exam.

USEFUL READING

Primary course textbook

- *Principles of Cyber-Physical Systems*, Rajeev Alur, MIT Press, 2015, http://mitpress.mit.edu/books/principles-cyber-physical-systems

Software References (available free online)

- *nuXmv v1.0 User Manual: https://es.fbk.eu/tools/nuxmv/downloads/nuxmv-user-manual.pdf*
- *NuSMV v2.5 User Manual: http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf*

Other Reference Books (available free online)

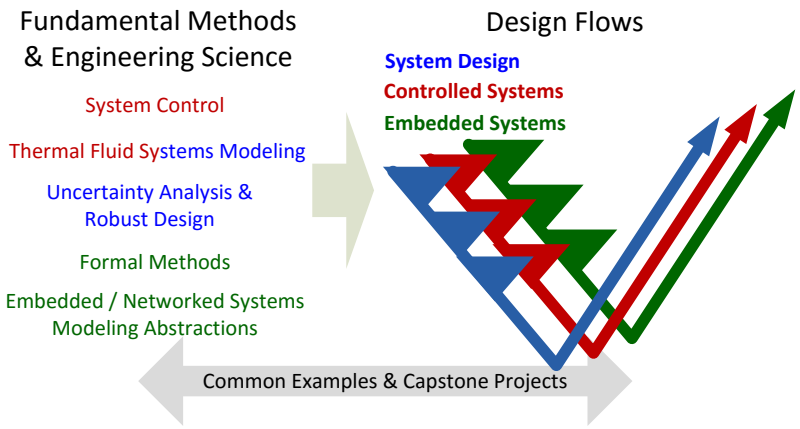
- Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Ed Lee and Sanjit Seshia, 2014, http://leeseshia.org
- Background material (logic, discrete math, deduction, etc.): Foundations of Computer Science, Al Aho and Jeff Ullman, 1994, http://infolab.stanford.edu/~ullman/focs.html
- Theorem proving material: Concrete Semantics with Isabelle/HOL, Tobias Nipkow and Gerwin Klein, 2015, http://www.concrete-semantics.org/

Other Reference Books (for purchase)

- Model Checking, Edmund Clarke, Orna Grumberg, and Doron Peled, MIT Press, 1999, http://mitpress.mit.edu/books/model-checking

Helpful links:

- Virtual Computer Lab at UConn: http://skybox.uconn.edu/
- Course Material: https://lms.uconn.edu
- Institute for Advanced Systems Engineering: http://www.utc-iase.uconn.edu/



Coursework Targeting Student Outcomes

During the semester, students will be challenged in areas that are designed to help them to successfully realize proficiency in the student outcomes: Participation, Homework, Oral Presentations, and Project Report. The final course grade will be based on the following:

Grading

- Homework assignments, 65%
  - We expect approximately one assignment per week over the duration of the course.
- Quizzes (online via HuskyCT), 15%
  - We expect approximately one short quiz per week over the duration over the course. The quizzes will primarily be multiple choice, true/false, etc. to reinforce the definitions and basic theoretical concepts, while the homework assignments will emphasize the use of formal methods and verification tools.
- Final take home exam, 20%
  - The take home final exam will be similar to the homeworks, but extended and incorporating material from the three modules.

Homework

Homework assignments will be posted on HuskyCT. Homework assignment due dates will be given with the assignment. NO late homework will be accepted as the homework will often be discussed in class. Each problem will be graded on a scale of 0-100.

Project, Presentations and Project Report

A project is to be developed by student groups, which is expected to evolve during the entirety of the track. The portion of the project that is to be executed in this course refers mainly to design project

identification, challenge quantification, significance and relevance to the MBD philosophy and plan of attack. The final deliverable (presentation) should identify all the aforementioned elements in a quantifiable manner and suggest a strategy for solution.

Software

This course will make extensive use of formal methods and verification software tools, including:

- Matlab Simulink/Stateflow (especially Stateflow)
- NuSMV: http://nusmv.fbk.eu/
- nuXMV: https://nuxmv.fbk.eu/ (We will try to use the latest version of NuSMV called nuXmv, but NuSMV is more stable)
- Z3 (runnable online via an SMT frontend at http://rise4fun.com/z3/, but if you want the Python API, it needs to be installed locally)
- Uppaal: http://www.uppaal.org/
- Isabelle/HOL: https://www.cl.cam.ac.uk/research/hvg/Isabelle/
- We may use some other tools in parts of the course, some possible tools are listed throughout the syllabus (e.g., Frama-C, Spin, PVS, TLA+), but we plan to stick primarily to Stateflow, NuSMV, Z3, and Uppaal, with some use of Isabelle in the theorem proving module.

Other Policies

Student Conduct:

http://www.dosa.uconn.edu/student_code.html. Students are responsible for adherence to the University of Connecticut student code of conduct. Perhaps the most important policy to pay attention to is the section on Student Academic Misconduct. “Academic misconduct is dishonest or unethical academic behavior that includes, but is not limited, to misrepresenting mastery in an academic area (e.g., cheating), intentionally or knowingly failing to properly credit information, research or ideas to their rightful originators or representing such information, research or ideas as your own (e.g., plagiarism).” Examples of academic misconduct in this class include, but are not limited to: copying solutions from the solutions manual, using solutions from students who have taken this course in previous years, copying your friends’ homework, looking at another student’s paper during an exam, lying to the professor or TA and incorrectly filling out the student workbook.

Attendance:

Attendance will not be taken; however, it is practically impossible to follow the class if classes are missed.

Absences:

Make-up of missed exams requires permission from the Dean of Students, see "Academic Regulations." Midterm-exams are treated the same as Final Examinations. Students involved in official University activities that conflict with class time must inform the instructor in writing prior to the anticipated absence and take the initiative to make up missed work in a timely fashion. In addition, students who will miss class for a religious observance must "inform their instructor in writing within the first three weeks of the semester, and prior to the anticipated absence, and should take the initiative to work out with the instructor a schedule for making up missed work."

Course Schedule*

Date	Lecture	References	Lecturers
	Lecture 1: Systems Engineering Fundamentals	Course Notes, Ref. [7]	T. Glagowski
	Lecture 2: Systems Engineering in Industry	Course notes Ref. [7]	T. Glagowski
	Lecture 3: Systems Engineering in Industry	Course notes Ref. [7]	T. Glagowski
	Lecture 4: Embedded Systems Requirements	Ref. [6] Ref. [7]	T. Glagowski
	Lecture 5: Embedded System Design Considerations	Course Notes Ref. [7]	T. Glagowski
	Lecture 6: Platform-based design	Course Notes Ref. [7]	T. Glagowski
	Lecture 7: Finite State Machines	Course Notes Ref. [7]	L. Michel
	Lecture 8: Functional Modeling II	Course Notes Ref.[7]	B. Wang, L. Michel
	Lecture 9: Software Modeling Design	Course Notes Ref. [7]	B. Wang, L. Michel
	Lecture 10: System and Architecture Modeling	Ref. [7]	L Michel
	Lecture 11: RTOS Modeling	Course Notes Ref. [7]	Song Han
	Lecture 12: Industry Case Studies	Course Notes	
	Lecture 13: Distributed Systems Modeling I	Course Notes	M. Khan
	Lecture 14: Distributed Systems Modeling II	Course Notes	M. Khan
	Final/Project Presentations		