CSCI 699: Computer-Aided Verification

Units: 4

Spring 2020—MW—5:00—6:50pm

Location: GFS220

Http: https://r-mukund.github.io/teaching/sp2020-csci699/index.html

Instructor: Mukund Raghothaman

Office: SAL 308

Office Hours: Friday 3:00—5:00pm

Contact Info:

Http: https://r-mukund.github.io

Email: raghotha@usc.edu

Course Description

How does a programmer convince their colleagues that a piece of code does what was intended? How do they find bugs in the software they ship? And can the tools used for these activities also help in developing code that is correct, *by construction*?

This course will equip the student to answer these questions, and serve as an introduction to the principles and practices of software verification and program synthesis. We will study fundamental ideas such as symbolic execution, invariants, and abstract interpretation. We will also study how constraint solvers work, how they can be used to reason about code, and how they can be used as the building blocks of sophisticated program synthesis systems.

The course will cover both the theory underlying the design of these tools and their use in practice. Furthermore, by studying their application in areas such as networking and security, we will emphasize their relevance to the working computer scientist.

Learning Objectives

After this course, the successful student will be able to:

1. Prove properties of simple programs by devising appropriate invariants and ranking functions
2. Use constraint solvers to solve various problems
3. Explain the operation of modern constraint solvers, including unit propagation, back-tracking, clause learning, and theory combination
4. Explain the ideas behind abstract interpretation, including specific instantiations such as predicate abstraction and dataflow analyses
5. Implement simple static analyses for a tiny imperative programming language
6. Use and explain the operation of CEGIS-based program synthesizers

Prerequisite(s):

1. The course will expect a certain amount of mathematical maturity from its students, at least at the level of CSCI 170, and preferably at the level of CSCI 270.
2. We will be *reasoning about code*: we expect that the student will already be proficient in writing it. CSCI 301, CSCI 350, and similar courses are appropriate baselines.

Technological Proficiency and Hardware/Software Required

For the practical component of homework assignments, we will use various freely available software packages. Most Linux laptop and desktop workstations will be suitable for this purpose. The preliminary setup and low-level mechanics may be different, but Mac and Windows computers could potentially also be used.

Required Readings

We will assign readings from the following sources:

1. Michael Gordon. *Background Reading on Hoare Logic.* 2016.
   Accessible from https://www.cl.cam.ac.uk/archive/mjcg/HL/Notes/Notes.pdf.

2. Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View.* 2nd edition. Springer, 2016.

The book may be accessed freely from within USC at http://link.springer.com/978-3-662-50497-0. Supplementary material is available at http://www.decision-procedures.org/.

3. Anders Møller and Michael Schwartzbach. *Static Program Analysis.* Department of Computer Science, Aarhus University. 2018.
   Accessible from http://cs.au.dk/~amoeller/spa/.

4. Susan Horwitz. *Abstract Interpretation.* 2013.
   Accessible from http://pages.cs.wisc.edu/~horwitz/CS704-NOTES/10.ABSTRACT-INTERPRETATION.html.

5. Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. *Program Synthesis.* Foundations and Trends in Programming Languages, 2017.
   Accessible from https://ieeexplore.ieee.org/document/8187066.

We will be supplementing this material with recent research papers.

Description and Assessment of Assignments

1. There will be four homework assignments. Each assignment will include a theoretical component to assess conceptual understanding and a practical component to provide familiarity with using verification and synthesis tools on real programs. Homeworks will be done independently.

2. Students will complete a small research project over the course of the semester. This project can be done either individually or in groups of two. They can either:
   a. Find a new application of the verification and synthesis techniques presented in class, or
   b. select a recent research paper and reimplement its algorithm.

   At the end of the semester, they will submit a report and present their findings to the class.

   Students are strongly encouraged to consult the instructor while selecting an appropriate research project. In addition to the initial project proposal and final project report and presentation, it is recommended that they regularly update the instructor with their progress, both over email and during office hours.

Grading Breakdown

Assessment Tool (assignments)	**Points**	**% of Grade**
Assignment 1	100	20
Assignment 2	100	20
Assignment 3	100	20
Assignment 4	100	20
Research project (*incl.* presentation and report)	100	20
TOTAL	500	100

Assignment Submission Policy and Grading Timeline

In case of late submissions, students will incur a penalty of 20% for each day after the announced deadline. Feedback will be provided on submissions within two weeks of the deadline.

Course Schedule: A Weekly Breakdown

Module 1	**Techniques of Proof** **Reading:** Chapters 1, 2 and 5 of (Gordon, 2016). **Lab:** Verifying simple programs using the Boogie IVL

Week 1	**Course introduction** 1. Logistics 2. Motivation, overview of concepts 3. Pre-conditions, post-conditions and symbolic execution
Week 2	**Generalizing over execution paths** 1. Inductive invariants and program correctness 2. Verification conditions and the Boogie IVL 3. Termination and ranking functions
	Homework 1 out
Week 3	**Data-driven methods and applications** 1. Dynamically detecting likely invariants using Daikon (Ernst et al., ICSE 1999) 2. Inferring invariants using C2I (Sharma and Aiken, CAV 2014) 3. Verifying software defined networks using VeriCon (Ball et al., PLDI 2014)

Module 2	**Engines of Reason** **Reading:** Chapters 2, 3 and 4 of (Kroening and Strichman, 2016). **Lab:** Solving puzzles such as Sudoku, Kakuro, Numberlink and Slitherlink using Z3

Week 4	**Boolean satisfiability** 1. Propositional logic, satisfiability, validity 2. CNF formulas and Tseitin's transform 3. SAT solvers: DPLL and unit propagation
	Homework 1 due, homework 2 out
Week 5	**Satisfiability (*contd.*) and SMT** 1. Horn-SAT and CDCL 2. Theories and the SMT problem 3. Notable theories: Integers, uninterpreted functions, arrays, bit-vectors
	Project proposals due
Week 6	**SMT (*contd.*) and applications** 1. DPLL(T) and Nelson-Oppen 2. Verifying deep neural networks using Reluplex (Katz et al., CAV 2017)

Module 3	**Static Analysis and Abstract Interpretation** **Reading:** Chapters 2, 5 and 9 of (Møller and Schwartzbach, 2018)

Week 7	**Predicate abstraction** 1. Predicate abstraction and software model checking 2. Counter-example guided abstraction refinement (CEGAR) 3. Verifying Windows device drivers using SLAM (Ball et al., PLDI 2001)
	Homework 2 due, homework 3 out
Week 8	**Static analysis** 1. Dataflow analysis 2. Pointer analysis 3. Constraint-based analysis
Week 9	**Basics of abstract interpretation (Horwitz, 2013)** 1. Abstractions, concretizations and Galois connections 2. Transfer functions and widening operators

Module 4	**Program Synthesis** **Reading:** (Gulwani, Polozov and Singh, 2017) **Lab:** Synthesizing program invariants using SyGuS

Week 10	**The program synthesis problem** 1. Specifying user intent: Input-output examples and logical formulas 2. Encoding bias: Grammars and costs 3. The syntax-guided synthesis (SyGuS) framework
	Homework 3 due, homework 4 out
Week 11	**Designing solvers** 1. Version spaces and counter-example guided inductive synthesis (CEGIS) 2. Enumerative, symbolic and stochastic search 3. Type-based program synthesis (Osera et al., PLDI 2015)
Week 12	**Building applications** 1. Program repair using S3 (Le et al., FSE 2017) 2. Code generation using Chlorophyll (Phothilimthana et al., PLDI 2014) 3. Proving termination using SyGuS (Fedyukovich et al., CAV 2018)

Module 5	**Conclusion**

Week 13	**Student project presentations**
	Homework 4 due
Week 14	**Student project presentations (*contd.*)**
Week 15	**Looking back and looking forward** 1. Course recap 2. Discussion of future research directions
	Project reports due

Potential Project Ideas

A small list of potential project ideas is attached below. Students are welcome to explore topics and papers not from the list, as long as they are related to the topic of the course and contain a strong research component. In any case, they are strongly encouraged to consult the instructor early and often, and keep him updated on their progress.

1. **Modeling the heap.** The toy programming languages used in the course will exclude pointers and dynamically allocated data (provided by the library functions `malloc` and `free`, and keywords `new` and `delete`). This limitation is not only pedagogical, as accurately describing the effects of these functions is an important research challenge in the field. *Separation logic* (O'Hearn et al., CSL 2001) is a particularly influential approach. One research project involves building solvers for various fragments of separation logic, such as that proposed by (Piskac et al., CAV 2014).

2. **Relational verification.** The view of verification taken in the course will mostly be whether each execution of a program individually satisfies some property. Unfortunately, this fails to capture many interesting program properties, including privacy (the output should not depend on some sensitive feature of the input) and robustness (small changes in the input should only trigger small changes in the output), where the property involves comparing the behavior of the program on two close-but-distinct inputs. Students can build program verifiers for such *hyper-properties*, basing their work on papers such as (Barthe et al., FM 2011).

3. **Symbolic execution and concurrency.** An important feature in the modern programming landscape is concurrency, where the program is composed of multiple threads which coordinate by

mutating shared variables. Unfortunately, concurrent programs typically have a large number of possible thread interleavings, and this non-determinism makes symbolic execution hard. There is a wealth of research on analyzing such programs, for example (Wang et al., FSE 2009), which could form the basis of the course project.

4. **Min-UNSAT cores and maximum satisfiability.** The central problem we address in module 2 of the course is that of finding a satisfying assignment to a set of constraints. While a satisfying assignment is evidently immediately useful, reports of unsatisfiability clearly raise further questions. Users may wish to know: "*Why* was this problem instance unsatisfiable?", or in several applications, may wish to follow up with, "Can we *maximize* the number of satisfied constraints?" These questions naturally point to the problems of finding UNSAT-cores and of Max-SAT respectively. These problems are the topic of intense research, and students may wish to use an existing SAT solver as a black box, and reproduce recent algorithms such as (Neves et al., SAT 2015).

5. **Randomly sampling solutions and model counting.** One prominent assumption we make in our discussion of SAT solvers is that we are content with arbitrary satisfying assignments. However, in many cases, such as in the verification of probabilistic programs and several applications in machine learning, this is insufficient, and users would like an algorithm which *uniformly samples* from the set of satisfying assignments. The problem of random solution sampling is closely related to the problem of finding the *number* of satisfying assignments to a given formula; this latter problem is the famous prototype of the #P complexity class. As with the previous project idea, students may use an existing SAT solver as a black box, and attempt to reproduce the algorithms described in (Chakraborty et al., AAAI 2014) and related papers.

6. **Algorithms for synthesis.** Since the work on SyGuS, several new algorithms and frameworks have been developed for program synthesis. One important example has been the approach to combining top-down and bottom-up enumeration using a technique called *unification* (Alur et al., TACAS 2017). Another important question has been the notion of quantitative cost measures, such as the problem of finding the smallest or most high-performance solution. One algorithm is presented in (Hu and D'Antoni, CAV 2018). A third direction of research is in using learned syntactic features to accelerate the search for solution programs (Lee et al., PLDI 2018). Any of these papers would form a good basis for a course project.

Statement on Academic Conduct and Support Systems

Academic Conduct:

Plagiarism – presenting someone else's ideas as your own, either verbatim or recast in your own words – is a serious academic offense with serious consequences. Please familiarize yourself with the discussion of plagiarism in SCampus in Part B, Section 11, "Behavior Violating University Standards" policy.usc.edu/scampus-part-b. Other forms of academic dishonesty are equally unacceptable. See additional information in SCampus and university policies on scientific misconduct, policy.usc.edu/scientific-misconduct.

Support Systems:

Counseling and Mental Health - (213) 740-9355 – 24/7 on call
studenthealth.usc.edu/counseling
Free and confidential mental health treatment for students, including short-term psychotherapy, group counseling, stress fitness workshops, and crisis intervention.

National Suicide Prevention Lifeline - 1 (800) 273-8255 – 24/7 on call
suicidepreventionlifeline.org
Free and confidential emotional support to people in suicidal crisis or emotional distress 24 hours a day, 7 days a week.

Relationship and Sexual Violence Prevention and Services (RSVP) - (213) 740-9355(WELL), press "0" after hours – 24/7 on call
studenthealth.usc.edu/sexual-assault
Free and confidential therapy services, workshops, and training for situations related to gender-based harm.

Office of Equity and Diversity (OED)- (213) 740-5086 | Title IX – (213) 821-8298
equity.usc.edu, titleix.usc.edu
Information about how to get help or help someone affected by harassment or discrimination, rights of protected classes, reporting options, and additional resources for students, faculty, staff, visitors, and applicants. The university prohibits discrimination or harassment based on the following *protected characteristics*: race, color, national origin, ancestry, religion, sex, gender, gender identity, gender expression, sexual orientation, age, physical disability, medical condition, mental disability, marital status, pregnancy, veteran status, genetic information, and any other characteristic which may be specified in applicable laws and governmental regulations. The university also prohibits sexual assault, non-consensual sexual contact, sexual misconduct, intimate partner violence, stalking, malicious dissuasion, retaliation, and violation of interim measures.

Reporting Incidents of Bias or Harassment - (213) 740-5086 or (213) 821-8298
usc-advocate.symplicity.com/care_report
Avenue to report incidents of bias, hate crimes, and microaggressions to the Office of Equity and Diversity |Title IX for appropriate investigation, supportive measures, and response.

The Office of Disability Services and Programs - (213) 740-0776
dsp.usc.edu
Support and accommodations for students with disabilities. Services include assistance in providing readers/notetakers/interpreters, special accommodations for test taking needs, assistance with architectural barriers, assistive technology, and support for individual needs.

USC Support and Advocacy - (213) 821-4710
uscsa.usc.edu
Assists students and families in resolving complex personal, financial, and academic issues adversely affecting their success as a student.

Diversity at USC - (213) 740-2101
diversity.usc.edu
Information on events, programs and training, the Provost's Diversity and Inclusion Council, Diversity Liaisons for each academic school, chronology, participation, and various resources for students.

USC Emergency - UPC: (213) 740-4321, HSC: (323) 442-1000 – 24/7 on call
dps.usc.edu, emergency.usc.edu
Emergency assistance and avenue to report a crime. Latest updates regarding safety, including ways in which instruction will be continued if an officially declared emergency makes travel to campus infeasible.

USC Department of Public Safety - UPC: (213) 740-6000, HSC: (323) 442-120 – 24/7 on call
dps.usc.edu
Non-emergency assistance or information.