2011 Summer School on Program Synthesis

August 8-12, 2011

Schloss Dagstuhl, Germany

Topics

- Controller and automata synthesis
- Inductive synthesis and partial program synthesis
- Deductive synthesis
- Hands-on exercises and contests

Organizers

- Ras Bodik, UC Berkeley
- Sumit Gulwani, Microsoft Research
- Viktor Kuncak, EPFL
- Eran Yahav, Technion

How do I apply?

If you are a student or a postdoc and want to attend this summer school, send your application to bodik@cs.berkeley.edu. Include your CV a short essay of why you are interested in the material. The deadline for applications is June 30 but we will notify you of acceptance a few days after your

Logistics

The summer school will accommodate 25 students. Students will share double rooms. The cost of room and board will be 225EUR for the 5-day school.

Students are expected to arrive on Sunday, Aug 7, evening. The school will end with lunch on Friday Aug 12.

application arrives.

This summer school is supported in part by a travel grant from COST Action IC0901: Rich-Model Toolkit - An Infrastructure for Reliable Computer Systems.

The official Dagstuhl web page for this summer school.

Lectures (Schedule)

Program Synthesis is a Game **Barbara Jobstmann** , Verimag, CNRS Abstract and bio, slides.	• Basics about games • Program synthesis and repair using games • Recent synthesis algorithms and quantitative synthesis
SAT and SMT for Synthesis **Vijay Ganesh**, MIT Abstract and bio. Slides for lecture 1 and lecture 2.	• Standard-issue SAT solver • DPLL(T) and overview of SMT solvers and theories • Lazy/eager encodings for SMT theories •
Functional Synthesis via Quantifier Elimination and Theorem Proving **Viktor Kuncak**, EPFL abstract and bio, slides for lecture 1 and lecture 2.	• Synthesis of functional programs via quantifier elimination • Embedding synthesis language constructs into Scala • Interactive synthesis of code snippets via theorem proving
AutoBayes and synthesis by program derivation **Johann Schumann**, SGT, Inc. NASA Ames	• schema based algorithm synthesis • symbolic calculation and synthesis • the AutoBayes synthesis system

abstract and bio. Autobayes how-to and excercises.	
Constraint-based synthesis with Sketch **Armando Solar-Lezama**, MIT abstract and bio, lecture 1, online demo, simplellr.sk.	• Specifying candidate implementations with partial programs • Executable specifications and safety specifications • Synthesis with constraint solvers
Synthesis and Abstract Interpretation **Eran Yahav**, Technion abstract and bio, lecture 1, lecture 2.	• Synthesis of synchronization in concurrent programs • Static analysis for concurrent programs • Abstraction refinement and synthesis
Miscellaneous **Ras Bodik**, UC Berkeley abstract and bio, school opening, lecture, closing brainstorming.	• Learning-Based Synthesis with Version Space Algebra • Theorem-Prover-Based Synthesis, Rewrite-Based Synthesis • Interactive Synthesis

Abstracts

Program Synthesis is a Game

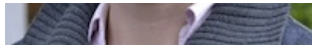
Automatically constructing a program from a specification can be seen as a game between two players: (i) the environment and (ii) the program. The environment chooses the input values on which the program is executed. The program chooses how to use these values to produce output values that are correct with respect to a given specification. The goal of the

Barbara Jobstmann *is a CNRS researcher in Verimag, an academic research laboratory belonging to the National Center for Scientific Research (CNRS) and the University of Grenoble in France. She joint Verimag in October 2009 after spending two years at the Ecole Polytechnique Federale de Lausanne, Switzerland. She received a Ph.D. degree in*

program is to compute correct output values for all allowed input values. So, no matter how the environment plays (i.e., which input values it provides to the program), the results will always satisfy the given specification.

The structure and the duration of the game depends on the given specification and on the program we are aiming for. E.g., if we aim for a sequential program that takes integer values as inputs and returns integer values as outputs, then the synthesis has two steps. In Step 1, the environment picks an input value, in Step 2, the program picks an output value. However, in each step there are infinitely many options from which the players can choose. Alternatively, we can ask to construct a reactive program (e.g., a user interface or implementations of a communication protocol). Such programs run forever and continuously react to their environments, i.e., they regularly read new inputs values. A play in the corresponding synthesis game has infinitely many steps in which the environment and the program pick input and output values, respectively. A usually assumption for games with infinite duration is that each player has only a finite set of possible values to choose from. The branch of program synthesis dealing with programs with finite memory but infinite durations is called reactive program synthesis and is the focus of this lecture.

The lecture is split into three parts. The first part gives an introduction to automata and automata-based game theory, the underlying techniques used in reactive program synthesis. In this part, I will also introduce the logic LTL (Linear-Time Temporal Logic), which provides a convenient way to describe the desired behavior of a reactive program. The second part of the lecture is devoted to classical program synthesis and repair techniques of reactive programs with respect to a given LTL specification. The third part of the lecture introduces recently developed and quite efficient synthesis techniques as well as extensions to more non-standard techniques as quantitative synthesis.



Computer Science from the University of Technology in Graz, Austria in 2007.

Her research focuses on developing game theory techniques and applying them to the problem of constructing correct and reliable computer systems. In particular, she used games to automatically construct, correct, and analyse computer systems and their specifications. Lately she is working on combined qualitative and quantitative specifications to construct systems that are correct and optimal.

She is co-chairing MEMOCODE 2011, the ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign. She is leading the Working Group 4 on high-level synthesis within the European research network "Rich-Model Toolkit - An Infrastructure for Reliable Computer Systems" (COST Action IC0901).

From SAT to SMT

In the last decade SAT/SMT solvers have seen an amazing improvement in efficiency and expressive power. The result has essentially been a dramatic rise in the use of SAT/SMT solvers in many areas of software engineering research such formal methods, program analysis and testing. It is safe to say that SAT/SMT solving is a disruptive technology. Irrespective of one's strategic frame of thought in the context of software reliability research, SAT/SMT solvers are an indispensable tactic.

In this series of talks, I will present the key technical ideas (developed by many researchers including myself) behind the success of SAT/SMT solvers, a description of some applications, some historical perspective, and future directions.

Vijay Ganesh is a research scientist at MIT since October 2007, and will soon join the IMDEA Software Institute, Madrid, Spain as an assistant professor. He completed his PhD in computer science from Stanford University in September 2007.

His primary research interests are SAT/SMT solvers, and their applications to software reliability, computer security and biology. He works on both the theory and practice of solvers. He has designed and implemented several solvers, most notably, STP and HAMPI. STP was one of the first solvers to enable an exciting new testing technique called systematic dynamic testing (or concolic testing). STP has been used in more than 100 research projects relating to software reliability and computer security. More recently he designed another solver, HAMPI, aimed at solving string constraints generated by the analysis of PHP, JavaScript and Perl programs. His paper on HAMPI won the ACM Distinguished Paper Award in 2009. STP was the winner of the SMTCOMP competition for bit-vector solvers in 2006 and 2010.

Functional Synthesis via Quantifier Elimination and Theorem Proving

To integrate synthesis into programming languages, software synthesis algorithms should ideally behave in a predictable way: they should succeed for a well-defined class of specifications. Moreover, software synthesis algorithms should support unbounded data types of programming languages, including numbers and data structures. I describe how to systematically generalize decision procedures into synthesis procedures, and use them to compile implicitly specified computations

***Prof. Viktor Kuncak** received a Ph.D. degree from MIT in 2007 working with Martin C. Rinard in CSAIL. In his dissertation he developed techniques for automated reasoning about data structures in imperative programs and formed the foundation of the Jahob verification system, which he designed and to a large extent implemented. Thomas Wies has made profound contributions to this system with his work on symbolic shape analysis and the Bohne tool. His PhD work also included several*

embedded inside functional and imperative programs. Synthesis procedures are predictable because they are guaranteed to find code that satisfies the specification whenever such code exists. To illustrate our method, I derive synthesis procedures by extending quantifier elimination algorithms for integer arithmetic and set data structures. I then show that an implementation of such synthesis procedures can extend a compiler to support implicit value definitions and advanced pattern matching.

decidability and undecidability results for constraints arising from program analysis. He has a M.Sc. degree also from MIT, and a B.Sc. degree in Computer Science from the University of Novi Sad.

Schema-based program Synthesis and the AutoBayes system

This lecture will combine theoretical background of schema based program synthesis with the hands-on study of a powerful, open-source program synthesis system (AutoBayes).

Schema-based program synthesis is a popular approach toward program synthesis. The lecture will provide an introduction into this topic and discuss how this technology can be used to generate customized algorithms.

The synthesis of advanced numerical algorithms requires the availability of a powerful symbolic (algebra) system. Its task is to symbolically solve equations, simplify expressions, or to symbolically calculate derivatives (among others) such that the synthesized algorithms become as efficient as possible. We will discuss the use and importance of the symbolic system for synthesis.

Any synthesis system is a large and complex piece of code. In this lecture, we will study Autobayes in detail. AutoBayes has been developed at NASA Ames and has been made open source. It takes a compact statistical specification and generates a customized data analysis algorithm (in C/C++) from it. AutoBayes is written in SWI Prolog and many concepts from rewriting, logic, functional, and symbolic programming. We will

*Dr. habil. **Johann Schumann** is Chief Scientist for Computational Sciences with SGT, Inc. and working at the Robust Software Engineering Group at the NASA Ames Research Center. He is engaged in research on software health management, verification and validation of advanced air traffic control algorithms and IVHM systems, and the generation of reliable code for data analysis and state estimation. Dr. Schumann's general research interests focus on the application of formal and statistical methods to improve design and reliability of advanced safety- and security-critical software. Dr. Schumann is author of a book on theorem proving in software engineering (Springer) and has published more than 90 articles on automated deduction and its applications, automatic program generation, V&V of safety-critical systems, and neural network oriented topics.*

discuss the system architecture, the schema libary and the extensive support infra-structure.

Practical hands-on experiments and exercises will enable the student to get insight into a realistic program synthesis system and provides knowledge to use, modify, and extend Autobayes.

An extensive AutoBayes Users Manual can be found here.

Constraint-based synthesis with Sketch

This talk introduces an approach to synthesis based on combinatorial search of a space of candidate programs guided by a set of constraints on the shape and the behavior of the desired solution.

In this paradigm, the user describes a space of candidate implementations by writing a partial program, a "program with holes" which leaves unspecified those parts of the program that are too complex or error-prone to write by hand. The partial program imposes constraints on the structure of the desired solution, allowing the user to take control over certain parts of the implementation while leaving others in the hands of the synthesizer.

In addition to the partial program, the user provides a set of constraints on the behavior of the desired solution. These constraints can range from simple assertions in the code, to unit tests, to reference implementations whose behavior must be emulated by the synthesized code. The role of the synthesizer is to combine the behavioral constraints with the structural constraints imposed by the partial program, and to produce an implementation that is consistent with both.

The lecture will describe the constraint-based synthesis paradigm as implemented in the Sketch synthesis system. The first part will describe

Armando Solar-Lezama *is an assistant professor at the Massachusetts Institute of Technology, where he leads the Computer Aided Programming group. Before joining MIT, he was a graduate student at Berkeley, where he graduated with a PhD in the fall of 2008.*

constraint based synthesis from the user's perspective; in this part, students will gain first-hand experience on the advantages and disadvantages of the constraint-based approach to synthesis, and will learn about some of the open problems in improving the usability of this form of synthesis. The second part of the talk will dive into the details of the counterexample guided inductive synthesis algorithm (CEGIS) as implemented in the Sketch synthesizer. This part of the talk will describe some of the strengths and weaknesses of the algorithm and potential opportunities for new research to improve the scalability of the approach.

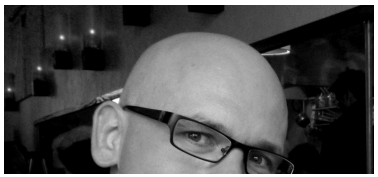
Synthesis and Abstract Interpretation

This talk will present a framework for synthesizing efficient synchronization in concurrent programs, a task known to be difficult and error-prone when done manually. The framework is based on abstract interpretation and can infer synchronization for infinite state programs. Given a program, a specification, and an abstraction, we infer synchronization that avoids all (abstract) interleavings that may violate the specification, but permits as many valid interleavings as possible. The klecture will show application of this general idea for automatic inference of atomic sections and memory fences in programs running over relaxed memory models.

Prof. Eran Yahav *Eran Yahav is a faculty member of the Computer Science department at the Technion, Israel. Prior to his position at Technion, he was a Research Staff Member at the IBM T.J. Watson Research Center in Hawthorne, New York (2004-2010). He received his Ph.D. from Tel Aviv University (2004) and his B.Sc. (cum laude) from the Technion-Israel Institute of Technology (1996). His research interests include static and dynamic program analysis, program synthesis, and program verification. Eran is a recipient of the prestigious Alon Fellowship for Outstanding Young Researchers, and the Andre Deloro Career Advancement Chair in Engineering.*

Miscellaneous

This series of talks will introduce topics not covered by guest lecturers. We will start with synthesis based on machine learning, specifically



Ras Bodik *is an associate professor at UC Berkeley, where he heads projects in program synthesis and mobile web browsers.*

version space algebra, using SmartEdit of Tessa Lau *et al* and Quick Code of Sumit Gulwani *et al*. Next we introduce rewrite-based synthesizers FFTW of Frigo and the CMU Spiral, which have been used to produce highly efficient linear filters. As an example of theorem-based synthesizer, we will cover Nelson *et al* Denali. Finally, we will discuss what interactive program synthesis might look like, using Angelic programming as an example.

Schedule

schedule : summer school

Time	Location	Mon	Tue	Wed	Thu	Fri
7:30 - 8:45	*restaurant*	breakfast	breakfast	breakfast	breakfast	breakfast
8:45 - 10:00		**Introduction (Ras)**	**Eran**	**Eran**	**Viktor**	**Vijay**
10:00 - 10:30	*classroom*	coffee break	coffee break	coffee break	coffee break	coffee break
10:30 - 12:00		**Barbara**	**Barbara**	**Johann**	**Johann**	**Discussion (Ras)**
12:15 - 2:00	*restaurant*	lunch	lunch	lunch	lunch	lunch
2:00 - 3:30		**Armando**	**Armando**	*afternoon*	**Armando and Johann**	departure
3:30 - 4:00	*restaurant*	coffee and cake	coffee and cake	*outing to nearby*	coffee and cake	
4:00 - 5:30		**Viktor**	**Vijay**	*attraction*	**Ras**	
6:00 - 7:30	*restaurant*	dinner	dinner	dinner	dinner	
8:00 - tbd	*wine cellar*	**discussions; AutoBayes install help**	***Vernissage at 7:30; discuss.***	**discussions, hands-on, etc**	**discussions, hands-on, etc**	

summer school