Seminars *Program Synthesis* and *Deductive Verification*

Introduction

Winter 2023/24; October 11, 2023

Thomas Noll and Philipp Schroer
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-23-24/synthesis/
https://moves.rwth-aachen.de/teaching/ws-23-24/verification/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline

Formal Methods in General

Formal methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Formal Methods in General

Formal methods

- Rigorous, mathematically based techniques for the specification, development and verification of software and hardware systems
- Aim at improving correctness, reliability and robustness of such systems

Classifications

- According to design phase
  - specification, development, implementation, testing, ...
- According to specification formalism
  - source code, logical formulae, ...
- According to underlying mathematical theories
  - model checking, theorem proving, static analysis, term rewriting, ...

Areas Covered by the Seminars

Topic areas

- Program Synthesis
  A. Syntax-Guided Synthesis
  B. Synthesis by Deductive Search
  C. Synthesis from Input/Output Examples
  D. Synthesis by Equational Term Rewriting
  E. Synthesis using Large Language Models
  F. Synthesis by Reinforcement Learning

- Deductive Verification
  A. Deductive Verification in Practice
  B. Verification Languages
  C. Verification of Probabilistic Programs

Outline

Goals

Aims of this seminar

- Independent understanding of a scientific topic
- Acquiring, reading and understanding scientific literature
  - given references sufficient in most cases
- Writing of your own report on this topic
  - far more that just a translation/rewording
  - usually an “extended subset” of original literature
    - “subset”: present core ideas and omit too specific details (e.g., related work or optimisations)
    - “extended”: more extensive explanations, examples, ...
    - discuss contents with supervisor!
- Oral presentation of your results
  - can be “proper subset” of report
  - generally: less (detailed) definitions/proofs and more examples

Requirements on Report

Your report

- Independent writing of a report of 12–15 pages
- First milestone: detailed outline
  - not: “1. Introduction/2. Main part/3. Conclusions”
  - rather: overview of structure (section headers, main definitions/theorems) and initial part of main section (one page)
- Complete set of references to all consulted literature
- Correct citation of important literature
- Plagiarism: taking text blocks (from literature or web) without source indication causes immediate exclusion from this seminar
- Font size 12pt with “standard” page layout
  - LaTeX template will be made available on seminar web page
- Language: German or English
- We expect the correct usage of spelling and grammar
  - $\geq$ 10 errors per page $\Longrightarrow$ abortion of correction

Requirements on Talk

Your talk

- Talk of 30 minutes
- Available: projector, presenter, [laptop]
- Focus your talk on the audience
- Descriptive slides:
  - $\leq$ 15 lines of text
  - use (base) colors in a useful manner
  - number your slides
  - LaTeX/beamer template will be made available on seminar web page
- Language: German or English
- No spelling mistakes please!
- Finish in time. Overtime is bad
- Ask for questions
- Have backup slides ready for expected questions

Outline

Important Dates

Deadlines

- October 16: Topic preferences due
- November 20: Detailed outline due
- December 18: Full report due
- January 15: Presentation slides due
- January 22–23 (?): Seminar talks

Important

- Missing a deadline causes immediate exclusion from the seminar
- Please notify us if you decide to quit

Selecting Your Topic

Procedure

- You obtain(ed) a list of topics of this seminar.
- Indicate the preference of your topics (first, second, third).
- Return sheet here or via e-mail (`noll@cs.rwth-aachen.de`) by Monday (October 16).
- We do our best to find an adequate topic-student assignment.
  - disclaimer: no guarantee for an optimal solution
- Assignment will be published on web site next week.
- Then also your supervisor will be indicated.
- Please give language preference (unsure $\Longrightarrow$ German).

Withdrawal

- You have up to three weeks to refrain from participating in this seminar.
- Later cancellation (by you or by us) causes a not passed for this seminar and reduces your (three) possibilities by one.

Outline

Overview

Aims of this Seminar

Important Dates

The *Program Synthesis* Topics [Thomas Noll]

The *Deductive Verification* Topics [Philipp Schroer]

Final Hints

The Dream of Program Synthesis

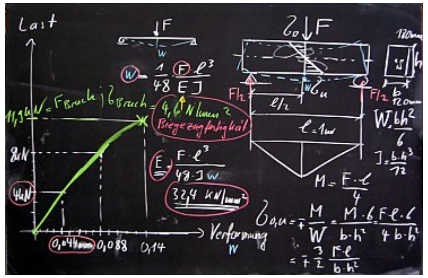
Programming language

→

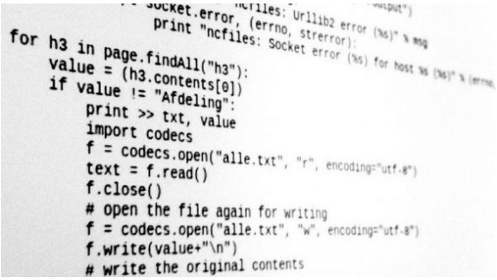
Synthesiser

←

User intent



↓



Program

The Dream of Program Synthesis

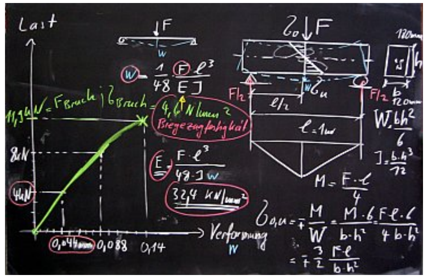
Programming language

→

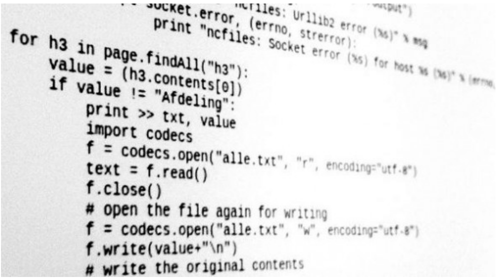
Synthesiser

←

User intent



↓



Program

Challenges

- Intractability of program space (usually infinite)
- Diversity of user intent

Syntax-Guided Synthesis

Approach: Provide syntactic restrictions to reduce search space

Paper 1: R. Alur et al.: *Syntax-guided synthesis*

Specification: syntactic set of candidate implementations given by grammar, background theory, semantic correctness specification given by logical formula

Technique: counter-example-guided-inductive-synthesis (CEGIS)

Output: implementation satisfying the specification in the theory

Example: specification

$$\max(x, y) \geq x \wedge \max(x, y) \geq y \wedge (\max(x, y) = x \vee \max(x, y) = y)$$

with candidate grammar

$$E \to E + E \mid E - E \mid E \leq E \mid E \,?\, E : E \mid 0 \mid 1$$

(interpreted over integers) yields implementation

$$x \leq y \,?\, y : x$$

Syntax-Guided Synthesis (continued)

Paper 2: A. Solar-Lezama: *Program Sketching*

Specification: partial program (“sketch”) with assertions, syntactic set of candidate implementations given by grammar

Technique: counter-example-guided-inductive-synthesis (CEGIS)

Output: implementation satisfying the specification

Paper 3: A.V. Nori, S. Ozair, S.K. Rajamani, D. Vijaykeerthy: *Efficient synthesis of probabilistic programs*

Specification: real-world dataset and sketch of probabilistic program with “holes”

Technique: Markov Chain Monte Carlo based synthesis

Output: replacement of holes with program fragments such that execution is consistent with data

Synthesis by Deductive Search

Approach: Use automated theorem provers to construct proof of user-provided specification and extract corresponding program

Paper 4: S. Srivastava, S. Gulwani, J.S. Foster: *From program verification to program synthesis*

Specification: input/output functional specification, atomic operations in the programming language, specification of program's looping structure

Technique: generation of program sketch and deductive proof search

Output: implementation satisfying the specification

Paper 5: N. Polikarpova, I. Sergey: *Structuring the synthesis of heap-manipulating programs*

Specification: pair of assertions (φ, ψ) (pre- and postcondition) from Separation Logic (= Hoare Logic + pointers)

Technique: deductive proof search

Output: imperative program with pointers that transforms any heap state satisfying φ into one satisfying ψ

Example: specification

$$\{\mathrm{x} \mapsto a * \mathrm{y} \mapsto b\} P \{\mathrm{x} \mapsto b * \mathrm{y} \mapsto a\}$$

yields program

$$\mathrm{a1} = *\mathrm{x}; \mathrm{b1} = *\mathrm{y}; *\mathrm{y} = \mathrm{a1}; *\mathrm{x} = \mathrm{b1}$$

Synthesis from Input/Output Examples

Approach: Synthesise programs that implement a given (finite) input/output behaviour

Paper 6: J. Hamza, V. Kuncak: *Minimal Synthesis of String To String Functions From Examples*

Specification: pairs of input/output strings

Technique: generation of functional non-deterministic Mealy machine implementing string-to-string transformation

Output: smallest Mealy machine consistent with examples

Paper 7: S. Chasins, P.M. Phothilimthana: *Data-Driven Synthesis of Full Probabilistic Programs*

Specification: one-dimensional (large) data set

Technique: generation of program sketch from input data, completion of sketch by simulated annealing

Output: complete probabilistic program implementing distribution

Paper 8: K. Shi, J. Steinhardt, P. Liang: *FrAngel: component-based synthesis with control structures*

Specification: set of input/output examples, library of components

Technique: mining of promising code fragments from remembered programs (with with unspecified control structure), instantiation of “angelic conditions”

Output: concrete program implementing I/O behaviour

Synthesis using Large Language Models

Approach: Synthesise programs that implement a given (finite) input/output behaviour

Paper 9: J. Austinet al.: *Program Synthesis with Large Language Models*

Specification: natural language description of program behaviour

Technique: evaluation of collection of LLMs on Mostly Basic Programming Problems (MBPP) and MathQA-Python benchmarks

Output: success rates and performance figures

Paper 10: E. Nijkampet al.: *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis*

Specification: given by Multi-Turn Programming Benchmark (MTPB; follows multi-step paradigm: single program specification factorised into multiple prompts specifying subproblems)

Technique: training of CodeGen

Output: CodeGen family of large language models, training library JAXFORMER

Synthesis by Reinforcement Learning

Approach: Use term rewriting techniques to synthesise programs

Paper 11: D.J. Mankowitz et al.: *Faster sorting algorithms discovered using deep reinforcement learning*

Specification: improvement of sorting routine (bounded array size) modelled as single-player game (action: appending assembly instruction)

Technique: training of deep reinforcement learning agent (AlphaDev) to play game

Output: small sorting algorithms for arrays of size ≤ 5 outperforming human benchmarks (integrated into LLVM standard C++ sort library)

Outline

Using Deductive Semantics to Verify Programs

In short: We look at program behavior symbolically to formally prove that it behaves according to a specification.

A. Deductive Verification In Practice
B. Verification Languages
C. Probabilistic Programs

Deductive Verification In Practice

- The Prusti Project: Formal Verification for Rust
- Deductive Verification of Smart Contracts with Dafny
- Formally Validating a Practical Verification Condition Generator
- egg: Fast and extensible equality saturation

Verification Languages

- Automating Induction with an SMT Solver
- Verified Calculations
- VeyMont: Parallelising Verified Programs Instead of Verifying Parallel Programs
- One Logic to Use Them All

Probabilistic Programs

- This is the moment for probabilistic loops
- Lower Bounds for Possibly Divergent Probabilistic Programs
- Quantitative strongest post: a calculus for reasoning about the flow of quantitative information
- Exact Bayesian Inference on Discrete Models via Probability Generating Functions: A Probabilistic Programming Approach

Outline

Some Final Hints

Hints

- Take your time to understand your literature.
- Be proactive! Look for additional literature and information.
- Discuss the content of your report with other students.
- Be proactive! Contact your supervisor on time.
- Prepare the meeting(s) with your supervisor.
- Forget the idea that you can prepare a talk in a day or two.

We wish you success and look forward to an enjoyable and high-quality seminar!