Foundations of Probabilistic Programming/Program Synthesis

Introduction to seminar
Summer Semester 2019; April 3, 2019

Joost-Pieter Katoen, Thomas Noll et al.
Software Modeling and Verification Group
RWTH Aachen University

`https://moves.rwth-aachen.de/teaching/ss-19/fpp/`
`https://moves.rwth-aachen.de/teaching/ss-19/pgmsyn/`

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline

Foundations of Probabilistic Programming

Probabilistic Programs

Probabilistic programs extend sequential programs with **random sampling** and (hard and soft) **conditioning**.
They are used in machine learning, robotics, computer vision, etc.
Almost every programming language (C, Python, Prolog, etc.) has a probabilistic extension.

```
def pulls(strength real):
    normal(strength, 1)

def winner(a real, b real):
    a_pull <~ pulls(a)
    b_pull <~ pulls(b)
    return (a_pull > b_pull)

alice <~ normal(0,1)
bob   <~ normal(0,1)
carol <~ normal(0,1)

match1 <~ winner(alice, bob)
match2 <~ winner(bob, carol)
match3 <~ winner(alice, carol)
```

Foundations of Probabilistic Programs

- What do probabilistic programs exactly mean? **Semantics.**
- How to reason about probabilistic programs? **Correctness.**
- When does a probabilistic program terminate? **Termination.**
- How long does a probabilistic program run? **Complexity.**
- When do two programs behave (almost) the same ? **Equivalence.**

Program Synthesis

Program Synthesis

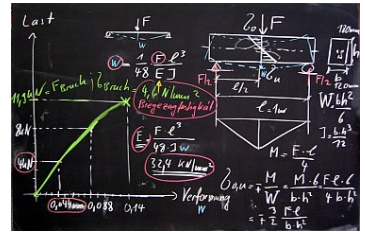
Program synthesis is the task of **automatically finding a program** in the underlying programming language that **satisfies the user intent** expressed in the form of some specification.

Dream of Program Synthesis

Programming language

Synthesiser

User intent



```
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\n")
        # write the original contents
```

Program

Overview

Dream of Program Synthesis

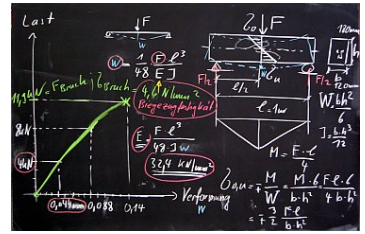
Programming language

→

Synthesiser

←

User intent



↓

```
                print "ncfiles: Urllib2 error (%s)" % msg
            print "ncfiles: Socket error (%s) for host %s (%s)" % (errno,
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\n")
        # write the original contents
```

Program

Challenges

- Intractability of program space
- Diversity of user intent

Applications of Program Synthesis

Data wrangling: cleaning, transforming, and/or preparing data for analysis and presentation

- often using programming-by-example (e.g., `john.doe@rwth.de` $\rightarrow$ `John Doe`)

Graphics: synthesis of programs for constructing graphical objects to enable interactive editing experiences and efficient animations

Code Repair: given buggy program P that violates specification φ, find new P' that satisfies φ

Code Suggestions: advanced form of "autocompletion"

Probabilistic modelling: learning of probabilistic programs that explain empirical evidence

Superoptimisation: synthesising an optimal sequence of instructions

- e.g., computing average of unsigned integers x and y without conditionals: $(x + y)/2$ can overflow, but $(x \mid y) - ((x \oplus y) \gg 1)$ works

Concurrent Programming: determine placement of minimal synchronisation constructs to avoid data races

Outline

Aims of this Seminar

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 paper
    - “subset”: present core ideas and omit too specific details
    - “extended”: more extensive explanations, examples, ...
    - discuss 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 **10–15 pages**
- First milestone: **detailed outline**
  - not: “1. Introduction/2. Main part/3. Conclusions”
  - but: 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
- **Language**: German or English
- We expect the **correct usage** of spelling and grammar
  - $\geq$ 10 errors per page $\Longrightarrow$ abortion of correction
- **LaTeX template** will be made available on seminar web page

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
- **Language:** German or English
- No spelling mistakes please!
- Finish **in time**. Overtime is bad
- Ask for **questions**
- Have **backup slides** ready for expected questions
- **LaTeX/beamer template** will be made available on seminar web page

Outline

Important Dates

Deadlines

- 6 May: Detailed outline of report due
- 3 June: Full report due
- 1 July: Presentation slides due
- 11 July (afternoon): Seminar

Important Dates

Deadlines

- 6 May: Detailed outline of report due
- 3 June: Full report due
- 1 July: Presentation slides due
- 11 July (afternoon): Seminar

Missing a deadline causes **immediate exclusion** from the seminar

Important Dates

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 **by Friday (5 April)** via e-mail (`noll@cs.rwth-aachen.de`) or to secretary.
- 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.

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 **by Friday (5 April)** via e-mail (`noll@cs.rwth-aachen.de`) or to secretary.
- 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.

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

The *Foundations of Probabilistic Programming* Topics

Logical essentials of Bayesian reasoning

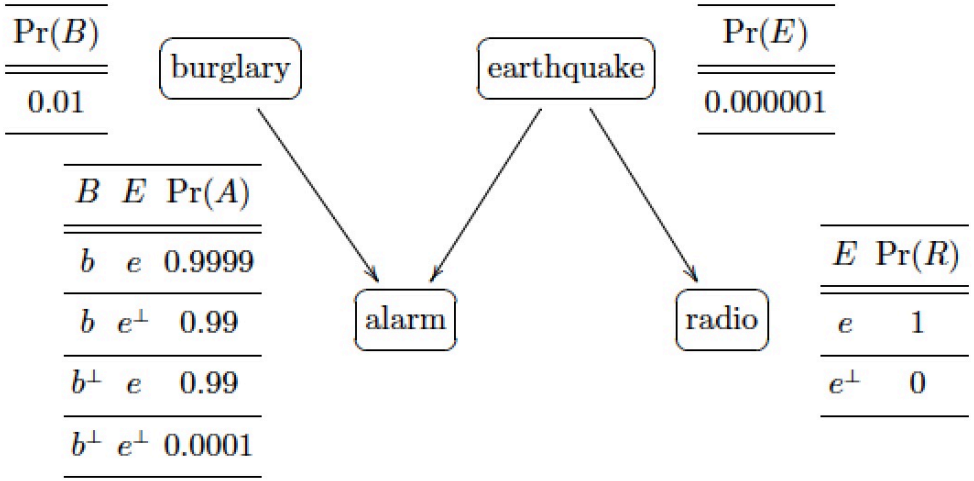
Book chapter: B. Jacobs and F. Zanasi: *Logical essentials of Bayesian reasoning*, 2019

Goal: understanding the essence of Bayesian inference

Programs: a uniform, structured and expressive language for describing Bayesian phenomena in terms of familiar programming concepts, like channel, predicate transformation and state transformation

Approach: channel-based approach to Bayesian probability theory

Example:



The *Foundations of Probabilistic Programming* Topics

Probabilistic couplings

Book chapter: J. Hsu and G. Barthe: *Probabilistic Couplings from Program Logics*, 2019

Goal: Prove whether two probabilistic programs behave (almost) the same.

Programs: Probabilistic programs without conditioning

Approach: Probabilistic couplings. Relational Hoare logics.

Example:

$k \xleftarrow{\$} \textbf{Flip};$ $r \leftarrow k \oplus s$	$k \xleftarrow{\$} \textbf{Flip};$ $r \leftarrow k$

If the distribution of r is the same in both programs, then the XOR cipher is secure

The *Foundations of Probabilistic Programming* Topics

Termination analysis with martingales

Book chapter: K. Chatterjee *et al.*: *Termination Analysis of Probabilistic Programs with Martingales*, 2019

Goal: How to check whether:

- A probabilistic program terminates with probability one?
- A probabilistic program terminates in finitely many expected number of steps?

Programs: finite/infinite probabilistic choice, non-determinism

Approach: martingale theory, partial automation (algorithms)

Example:

```
1: x := 100;
2: while x ≥ 0 do
3:    if prob(0.5) then
4:       x := x + 1
      else
5:       x := x − 1
      fi;
   od
```

The *Foundations of Probabilistic Programming* Topics

Improving program testing by probabilistic programming

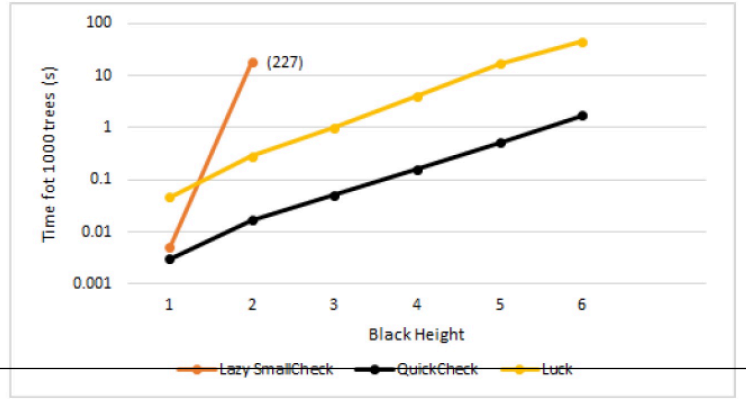
Book chapter: L. Leonidas, B. Pierce *et al.*: *Luck: A Probabilistic Language for Testing*, 2019

Goal: How to efficiently generate test cases for random testing of programs?

Programs: the probabilistic domain-specific language Luck in which test generators are conveniently expressed. Predicates with lightweight annotations to control the distribution of generated values and the amount of constraint solving needed.

Approach:
- formal semantics to the language Luck
- soundness and completeness of random generation w.r.t. a standard predicate semantics
- case studies showing effectiveness of bug-finding in programs

Example:



The *Foundations of Probabilistic Programming* Topics

Concentration measures

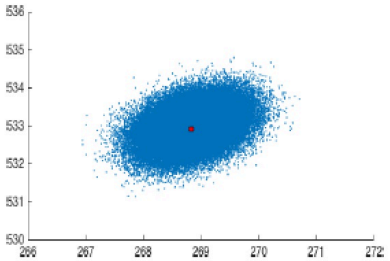
Book chapter: S. Sankaranarayanan: *Quantitative Analysis of Programs with Probabilities and Concentration of Measure Inequalities*, 2019

Goal: Quantitative analysis of probabilistic programs, in particular concentration of measure inequalities to reason about bounds on the probabilities of assertions.

Technique: measure inequalities, i.e., a set of mathematical ideas that characterize how functions of random variables deviate from their expected value.

Example:

```
1  angles = [10, 60, 110, 160, 140, ...
2            100, 60, 20, 10, 0]
3  x := TruncGaussian(0,0.05,-0.5,0.5)
4  y := TruncGaussian(0, 0.1,-0.5,0.5)
5  for reps in range(0,100):
6  #iterate through angles
7    for theta in angles:
8      # Distance travelled variation
9      d = Uniform(0.98,1.02)
10     # Steering angle variation
11     t = deg2rad(theta) * (1 + ...
12         TruncGaussian(0,0.01,-0.05,0.05))
13     # Move distance d with angle t
14     x = x + d * cos(t)
15     y = y + d * sin(t)
16 #Probability that we went too far?
17 assert(x >= 277)
```



Outline

Syntax-guided synthesis

Paper: R. Alur et al.: *Syntax-guided synthesis*, FMCAD 2013

Goal: synthesizing implementations given by grammar template from declarative specifications

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

Output: implementation satisfying the specification in the theory

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

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

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

Synthesis of pointer programs

Paper: N. Polikarpova, I. Sergey: *Structuring the synthesis of heap-manipulating programs*, POPL 2019

Goal: synthesizing imperative programs with pointers from declarative specifications

Programming language: simple pointer-manipulating language

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

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

Technique: deductive search

Example: specification

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

yields program

$$\mathtt{a1} = *\mathtt{x}; \mathtt{letb1} = *\mathtt{y}; *\mathtt{y} = \mathtt{a1}; *\mathtt{x} = \mathtt{b1}$$

Synthesis of probabilistic programs

Paper: A.V. Nori, S. Ozair, S.K. Rajamani, D. Vijaykeerthy: *Efficient synthesis of probabilistic programs*, PLDI 2015

Goal: synthesising probabilistic programs from real-world datasets

Programming language: imperative programs with sampling and conditioning

Specification: data set and “sketch” of probabilistic program with “holes”

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

Technique: Markov Chain Monte Carlo based synthesis

Example: sketch

$$c = ??(a, b)$$

may be replaced by

$$c = (a > b)$$

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.

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!