Seminar on Foundations of Probabilistic Programming WS 17/18
Kick–off Meeting

Joost-Pieter Katoen Benjamin Kaminski
Christoph Matheja

Rheinisch–Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 2

16.10.2017

Motivation

Probabilistic Programs

Motivation

Probabilistic Programs

- Like ordinary programs, except:
  - Program may toss a (possibly unfair) coin
  - Further execution of the program is influenced by outcome of the coin toss

Motivation

Probabilistic Programs

- Like ordinary programs, except:
  - Program may toss a (possibly unfair) coin
  - Further execution of the program is influenced by outcome of the coin toss
- Applications in:
  - Randomized algorithms
  - Security
  - Machine learning

Aims of this Seminar

Goals to Achieve

Aims of this Seminar

Goals to Achieve

- Independently understand a scientific topic

Aims of this Seminar

Goals to Achieve

- Independently understand a scientific topic
- Acquire, read, and understand suitable scientific literature

Aims of this Seminar

Goals to Achieve

- Independently understand a scientific topic
- Acquire, read, and understand suitable scientific literature
- Write of your own report on your topic

Aims of this Seminar

Goals to Achieve

- Independently understand a scientific topic
- Acquire, read, and understand suitable scientific literature
- Write of your own report on your topic
- Give a didactically appealing oral presentation on your topic

Aims of this Seminar

Demands on Your Report

Aims of this Seminar

Demands on Your Report

- Independently written report of 10 – 15 pages

Aims of this Seminar

Demands on Your Report

- Independently written report of 10 – 15 pages
- Normal font size (i.e. 11pt) with "normal" page layout

Aims of this Seminar

Demands on Your Report

- Independently written report of 10 – 15 pages
- Normal font size (i.e. 11pt) with "normal" page layout
- Correct spelling and grammar. More than 10 errors per page lead to abortion of correction

Aims of this Seminar

Demands on Your Report

- Independently written report of 10 – 15 pages
- Normal font size (i.e. 11pt) with "normal" page layout
- Correct spelling and grammar. More than 10 errors per page lead to abortion of correction
- Complete list of references to all consulted literature

Aims of this Seminar

Demands on Your Report

- Independently written report of 10 – 15 pages
- Normal font size (i.e. 11pt) with "normal" page layout
- Correct spelling and grammar. More than 10 errors per page lead to abortion of correction
- Complete list of references to all consulted literature
- Correct citation of important literature. Copying or slightly modifying text blocks without citation constitutes plagiarism and causes immediate exclusion from this seminar!

Aims of this Seminar

Demands on Your Talk

Aims of this Seminar

Demands on Your Talk

- Talk duration: about 30 minutes talk + 5 minutes questions. Overtime is bad!

Aims of this Seminar

Demands on Your Talk

- Talk duration: about 30 minutes talk + 5 minutes questions. Overtime is bad!
- Focus your talk on the audience

Aims of this Seminar

Demands on Your Talk

- Talk duration: about 30 minutes talk + 5 minutes questions. Overtime is bad!
- Focus your talk on the audience
- Prepare descriptive slides:
  - use less than 15 lines of text
  - use (base) colors in a useful manner

Aims of this Seminar

Demands on Your Talk

- Talk duration: about 30 minutes talk + 5 minutes questions. Overtime is bad!
- Focus your talk on the audience
- Prepare descriptive slides:
  - use less than 15 lines of text
  - use (base) colors in a useful manner
- Include slide numbers

Aims of this Seminar

Demands on Your Talk

- Talk duration: about 30 minutes talk + 5 minutes questions. Overtime is bad!
- Focus your talk on the audience
- Prepare descriptive slides:
  - use less than 15 lines of text
  - use (base) colors in a useful manner
- Include slide numbers
- Ask and prepare yourself for questions

Aims of this Seminar

Preparation of Your Talk

1

Aims of this Seminar

Preparation of Your Talk

- Either send in your slides as PDF or bring your own laptop.

1

Aims of this Seminar

Preparation of Your Talk

- Either send in your slides as PDF or bring your own laptop.
  - If you do decide to bring your own laptop, you need to have one of the following connectors available[1]:
    - VGA
    - HDMI (speak to your supervisor a few days in advance)
    - Mac Displayport

[1] We are strict about this because we had problems with this in the past.

Aims of this Seminar

Preparation of Your Talk

- Either send in your slides as PDF or bring your own laptop.
  - If you do decide to bring your own laptop, you need to have one of the following connectors available[1]:
    - VGA
    - HDMI (speak to your supervisor a few days in advance)
    - Mac Displayport
- Bring multiple copies of your slides (laptop, USB, web, email slides to supervisor, etc.)

[1] We are strict about this because we had problems with this in the past.

Important Dates/Deadlines

18.10.2017	Topic priorities due
23.10.2017	Student–topic–matching notification
6.11.2017	Detailed outline due
only until 13.11.2017	Drop out without consequences
4.12.2017	Report due
18.12.2017	Camera–ready report due
8.01.2018	Slides due
22.01.2018	Camera–ready slides due
5.02.2018	Seminar Talks
6.02.2018	Seminar Talks (cont'd)

You can also find these dates on the seminar's website!

Important Dates/Deadlines

18.10.2017	Topic priorities due
23.10.2017	Student–topic–matching notification
6.11.2017	Detailed outline due
only until 13.11.2017	Drop out without consequences
4.12.2017	Report due
18.12.2017	Camera–ready report due
8.01.2018	Slides due
22.01.2018	Camera–ready slides due
5.02.2018	Seminar Talks
6.02.2018	Seminar Talks (cont'd)

You can also find these dates on the seminar's website!

Important Dates/Deadlines

18.10.2017	Topic priorities due
23.10.2017	Student–topic–matching notification
6.11.2017	Detailed outline due
only until 13.11.2017	Drop out without consequences
4.12.2017	Report due
18.12.2017	Camera–ready report due
8.01.2018	Slides due
22.01.2018	Camera–ready slides due
5.02.2018	Seminar Talks
6.02.2018	Seminar Talks (cont'd)

You can also find these dates on the seminar's website!

All deadlines are firm!

Missing a deadline causes immediate exclusion from this seminar!

Topic Selection Procedure

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number
2. We give a short presentation of the topics

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number
2. We give a short presentation of the topics
3. You indicate your name and your three priorities for topics on that sheet

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number
2. We give a short presentation of the topics
3. You indicate your name and your three priorities for topics on that sheet
4. We do our best to find a good student–topic–matching (Disclaimer: no guarantee for an optimal solution)

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number
2. We give a short presentation of the topics
3. You indicate your name and your three priorities for topics on that sheet
4. We do our best to find a good student–topic–matching (Disclaimer: no guarantee for an optimal solution)
5. We announce the matching ASAP on the seminar's website. There you will also find out who your supervisor is

Topic Selection Procedure

1. We hand out a sheet with list of topics with a topic number
2. We give a short presentation of the topics
3. You indicate your name and your three priorities for topics on that sheet
4. We do our best to find a good student–topic–matching (Disclaimer: no guarantee for an optimal solution)
5. We announce the matching ASAP on the seminar's website. There you will also find out who your supervisor is
6. You contact your supervisor to get things started

Stand by while your sheet is being handed out to you...

The Topics

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?
  - Existing programming language yield incorrect results

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?
  - Existing programming language yield incorrect results
- Approach of the paper:

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?
  - Existing programming language yield incorrect results
- Approach of the paper:
  - Give a precise notion of previous samples

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?
  - Existing programming language yield incorrect results
- Approach of the paper:
  - Give a precise notion of previous samples
  - Develop correct MCMC algorithm for probabilistic programs

1: MCMC sampling of probabilistic programs

- Probabilistic programming relieves developers from writing inference code
- MCMC is a popular sampling technique that takes previous samples into account
- Applying MCMC to probabilistic programs is hard
  - What are "previous samples" if variables are assigned more than once?
  - Existing programming language yield incorrect results
- Approach of the paper:
  - Give a precise notion of previous samples
  - Develop correct MCMC algorithm for probabilistic programs
  - Implemented in Microsoft's Infer tool

2: Slicing Probabilistic Programs

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values
  - $Slice$ should be efficient: Slicing should be done as fast as possible

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values
  - $Slice$ should be efficient: Slicing should be done as fast as possible
- Traditional program slicing techniques do not suffice for probabilistic programs as they are not correct in this setting

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values
  - $Slice$ should be efficient: Slicing should be done as fast as possible
- Traditional program slicing techniques do not suffice for probabilistic programs as they are not correct in this setting
- Approach of the paper:

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values
  - $Slice$ should be efficient: Slicing should be done as fast as possible
- Traditional program slicing techniques do not suffice for probabilistic programs as they are not correct in this setting
- Approach of the paper:
  - Correct approach for probabilistic program slicing

2: Slicing Probabilistic Programs

- A probabilistic program P returns a distribution over return values
- Goal: Obtain a simpler program $Slice(P)$
  - $Slice$ should be correct: Slicing should preserve the distribution overt he return values
  - $Slice$ should be efficient: Slicing should be done as fast as possible
- Traditional program slicing techniques do not suffice for probabilistic programs as they are not correct in this setting
- Approach of the paper:
  - Correct approach for probabilistic program slicing
  - Algorithm to compute the best possible slice

3: Program analysis using martingales

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions
  - Invariants for reasoning about loops

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions
  - Invariants for reasoning about loops
- Probabilistic programs:
  - Weakest pre–expectations

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions
  - Invariants for reasoning about loops
- Probabilistic programs:
  - Weakest pre–expectations
  - (Ranking) supermartingales for reasoning about loops

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions
  - Invariants for reasoning about loops
- Probabilistic programs:
  - Weakest pre–expectations
  - (Ranking) supermartingales for reasoning about loops
    - (Ranking) supermartingales for probabilistic programs play the role of invariants (variants) for ordinary programs

3: Program analysis using martingales

- Ordinary programs:
  - Hoare logic / weakest preconditions
  - Invariants for reasoning about loops
- Probabilistic programs:
  - Weakest pre–expectations
  - (Ranking) supermartingales for reasoning about loops
    - (Ranking) supermartingales for probabilistic programs play the role of invariants (variants) for ordinary programs
- Use martingales to reason about:
  - Correctness
  - Almost–sure termination (i.e. termination with probability 1)

4: Semantics of probabilistic programs

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

$$\texttt{let } x = t \texttt{ in}$$
$$\texttt{let } y = u \texttt{ in}$$
$$\texttt{expression}$$

$$\texttt{let } y = u \texttt{ in}$$
$$\texttt{let } x = t \texttt{ in}$$
$$\texttt{expression}$$

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

$\texttt{let } x = t \texttt{ in}$	$\texttt{let } y = u \texttt{ in}$
$\texttt{let } y = u \texttt{ in}$	$\texttt{let } x = t \texttt{ in}$
$\texttt{expression}$	$\texttt{expression}$

- Why is this interesting?

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

`let` $x = t$ `in`	`let` $y = u$ `in`
`let` $y = u$ `in`	`let` $x = t$ `in`
`expression`	`expression`

- Why is this interesting?
  - Compiler optimizations for reordering statements

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

$\texttt{let } x = t \texttt{ in}$	$\texttt{let } y = u \texttt{ in}$
$\texttt{let } y = u \texttt{ in}$	$\texttt{let } x = t \texttt{ in}$
$\texttt{expression}$	$\texttt{expression}$

- Why is this interesting?
  - Compiler optimizations for reordering statements
  - Boost efficiency of symbolic and approximate inference

4: Semantics of probabilistic programs

- A language is commutative if only data flow is relevant

$\texttt{let } x = t \texttt{ in}$	$\texttt{let } y = u \texttt{ in}$
$\texttt{let } y = u \texttt{ in}$	$\texttt{let } x = t \texttt{ in}$
$\texttt{expression}$	$\texttt{expression}$

- Why is this interesting?
  - Compiler optimizations for reordering statements
  - Boost efficiency of symbolic and approximate inference
- Paper proves commutativity for functional probabilistic programming languages

5: Concurrent probabilistic programs

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms
- Rationale: Accumulate constraints instead of evaluating program states

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms
- Rationale: Accumulate constraints instead of evaluating program states
- Problem: No mechanism to deal with uncertainty

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms
- Rationale: Accumulate constraints instead of evaluating program states
- Problem: No mechanism to deal with uncertainty
- Approach of this paper:

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms
- Rationale: Accumulate constraints instead of evaluating program states
- Problem: No mechanism to deal with uncertainty
- Approach of this paper:
  - Develop a process algebra with random variables and timing

5: Concurrent probabilistic programs

- Concurrent constraint programming: Compositional specification technique for concurrentprograms
- Rationale: Accumulate constraints instead of evaluating program states
- Problem: No mechanism to deal with uncertainty
- Approach of this paper:
  - Develop a process algebra with random variables and timing
  - Compare operational and denotational semantics

6: Conditioning and recursion

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes
- How can we reason about other people's reasoning?
- Example: Alice and Bob speculate where they should meet...

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes
- How can we reason about other people's reasoning?
- Example: Alice and Bob speculate where they should meet...

$$P(\textit{Alice' choice} \mid \textit{AliceBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$
$$P(\textit{Alice' choice} \mid \textit{BobBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes
- How can we reason about other people's reasoning?
- Example: Alice and Bob speculate where they should meet...

$$P(\textit{Alice' choice} \mid \textit{AliceBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$
$$P(\textit{Alice' choice} \mid \textit{BobBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$

- **How to perform inference with nested conditioning?**

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes
- How can we reason about other people's reasoning?
- Example: Alice and Bob speculate where they should meet...

$$P(\textit{Alice' choice} \mid \textit{AliceBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$
$$P(\textit{Alice' choice} \mid \textit{BobBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$

- **How to perform inference with nested conditioning?**
- This paper:
  - Model cognitive science problems as probabilistic programs

6: Conditioning and recursion

- Bayesian statistics popular to model reasoning processes
- How can we reason about other people's reasoning?
- Example: Alice and Bob speculate where they should meet...

$$P(\textit{Alice' choice} \mid \textit{AliceBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$
$$P(\textit{Alice' choice} \mid \textit{BobBelieves}(\textit{Alice' choice} = \textit{Bob's choice}))$$

- **How to perform inference with nested conditioning?**
- This paper:
  - Model cognitive science problems as probabilistic programs
  - Dynamic programming approach for Bayesian inference with nested conditioning

7: Probabilistic Kleene algebra

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras
  - Introduce concurrency

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras
  - Introduce concurrency
  - Introduce quantitative information to reason about probabilities

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras
  - Introduce concurrency
  - Introduce quantitative information to reason about probabilities
  - Can be used to reason about probabilistic programs

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras
  - Introduce concurrency
  - Introduce quantitative information to reason about probabilities
  - Can be used to reason about probabilistic programs
  - Unifies nondeterminism, concurrency, and randomness

7: Probabilistic Kleene algebra

- Kleene algebras:
  - Generalization of regular expressions
  - Can be used to reason about programs
  - While–loops can be modeled with a star:

$$\big([loop\ guard] \cdot loop\ body\big)^* \cdot [\neg loop\ guard]$$

- Probabilistic concurrent Kleene algebras
  - Introduce concurrency
  - Introduce quantitative information to reason about probabilities
  - Can be used to reason about probabilistic programs
  - Unifies nondeterminism, concurrency, and randomness
  - Equational reasoning about probabilistic programs

8: Software–defined networks

8: Software–defined networks

- SDNs use software to program network architechtures

8: Software–defined networks

- SDNs use software to program network architechtures
- ProbNetKAT:

8: Software–defined networks

- SDNs use software to program network architechtures
- ProbNetKAT:
  - KAT: Kleene Algebra with Tests

8: Software–defined networks

- SDNs use software to program network architechtures
- ProbNetKAT:
  - KAT: Kleene Algebra with Tests
  - NetKAT: KAT based language for packet switching networks:
    - Has primitives for forwarding, filtering, modifying packets

8: Software–defined networks

- SDNs use software to program network architechtures
- ProbNetKAT:
  - KAT: Kleene Algebra with Tests
  - NetKAT: KAT based language for packet switching networks:
    - Has primitives for forwarding, filtering, modifying packets
  - ProbNetKAT: Probabilistic enhancement of NetKAT

8: Software–defined networks

- SDNs use software to program network architechtures
- ProbNetKAT:
  - KAT: Kleene Algebra with Tests
  - NetKAT: KAT based language for packet switching networks:
    - Has primitives for forwarding, filtering, modifying packets
  - ProbNetKAT: Probabilistic enhancement of NetKAT
- Main problem considered in the paper:
  - Description of the semantics of ProbNetKAT
  - Semantics (suprisingly) yield continuous probability measures

9: Expected runtimes

9: Expected runtimes

- What is the average runtime of a probabilistic program?

9: Expected runtimes

- What is the average runtime of a probabilistic program?

```
repeat
    {b := heads} [1/2] {b := tail};
until(b = heads)
```

9: Expected runtimes

- What is the average runtime of a probabilistic program?

```
repeat
    {b := heads} [1/2] {b := tail};
until(b = heads)
```

$$\mathbb{E}[t] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots = 2$$

9: Expected runtimes

- What is the average runtime of a probabilistic program?

```
repeat
    {b := heads} [1/2] {b := tail};
until(b = heads)
```

$$\mathbb{E}[t] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots = 2$$

- Approach of this paper:

9: Expected runtimes

- What is the average runtime of a probabilistic program?

```
repeat
    {b := heads} [1/2] {b := tail};
until(b = heads)
```

$$\mathbb{E}[t] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots = 2$$

- Approach of this paper:
  - Use a continuation passing style runtime transformer

9: Expected runtimes

- What is the average runtime of a probabilistic program?

```
repeat
    {b := heads} [1/2] {b := tail};
until(b = heads)
```

$$\mathbb{E}[t] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots = 2$$

- Approach of this paper:
  - Use a continuation passing style runtime transformer
  - Develop sound and complete proof rules

10: Computational hardness

10: Computational hardness

- Termination problems for ordinary programs are undecidable

10: Computational hardness

- Termination problems for ordinary programs are undecidable
- How hard is deciding probabilistic termination?
  - It's even "more undecidable"!

10: Computational hardness

- Termination problems for ordinary programs are undecidable
- How hard is deciding probabilistic termination?
  - It's even "more undecidable"!
- There are different degrees of undecidability
  - Arithmetical hierarchy

10: Computational hardness

- Termination problems for ordinary programs are undecidable
- How hard is deciding probabilistic termination?
  - It's even "more undecidable"!
- There are different degrees of undecidability
  - Arithmetical hierarchy
- This paper:
  - Classification of verification problems for probabilistic programs in terms of levels in the arithmetical hierarchy

10: Computational hardness

- Termination problems for ordinary programs are undecidable
- How hard is deciding probabilistic termination?
  - It's even "more undecidable"!
- There are different degrees of undecidability
  - Arithmetical hierarchy
- This paper:
  - Classification of verification problems for probabilistic programs in terms of levels in the arithmetical hierarchy
  - Almost–sure termination
  - Positive almost–sure termination
  - Expected values of program variables

11: Usefulness of non–termination

11: Usefulness of non–termination

- Ordinary programs either terminate or not

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1
- This paper argues:
  - Restriction to Almost–surely terminating probabilistic programs is not natural

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1
- This paper argues:
  - Restriction to Almost–surely terminating probabilistic programs is not natural
  - Line between almost–sure termination and non–almost–sure termination is "elusive, as well as arbitrary"

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1
- This paper argues:
  - Restriction to Almost–surely terminating probabilistic programs is not natural
  - Line between almost–sure termination and non–almost–sure termination is "elusive, as well as arbitrary"
  - Practical implications in learning theory

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1
- This paper argues:
  - Restriction to Almost–surely terminating probabilistic programs is not natural
  - Line between almost–sure termination and non–almost–sure termination is "elusive, as well as arbitrary"
  - Practical implications in learning theory
- Disclaimer: This is a cognitive science / philosophy paper.

11: Usefulness of non–termination

- Ordinary programs either terminate or not
- Probabilistic programs terminate with some probability q, which is either 1 or less than 1
- This paper argues:
  - Restriction to Almost–surely terminating probabilistic programs is not natural
  - Line between almost–sure termination and non–almost–sure termination is "elusive, as well as arbitrary"
  - Practical implications in learning theory
- Disclaimer: This is a cognitive science / philosophy paper.
  - Disclaimer: It's nevertheless a (formal) math paper.

12: Algorithmic analysis of termination

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime
- Approach in this paper:

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime
- Approach in this paper:
  - Restrict to affine probabilistic programs

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime
- Approach in this paper:
  - Restrict to affine probabilistic programs
  - Use linear ranking supermartingales to solve above problems

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime
- Approach in this paper:
  - Restrict to affine probabilistic programs
  - Use linear ranking supermartingales to solve above problems
  - Attempt synthesis of linear ranking supermartingales

12: Algorithmic analysis of termination

- Qualitative problems considered:
  - Almost–sure termination (i.e. with probability 1)
  - Positive almost–sure termination (i.e. in finite expected time)
- Quantitative problems considered:
  - Expected runtime
  - Tail bounds on expected runtime
- Approach in this paper:
  - Restrict to affine probabilistic programs
  - Use linear ranking supermartingales to solve above problems
  - Attempt synthesis of linear ranking supermartingales
  - Side product: tail bounds on expected runtime

13: Stochastic Invariants

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S
- ranking + repulsing supermartingales can be used for:

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S
- ranking + repulsing supermartingales can be used for:
  - Witnessing lower bounds for termination probabilities

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S
- ranking + repulsing supermartingales can be used for:
  - Witnessing lower bounds for termination probabilities
  - Witnessing non–almost–sure termination

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S
- ranking + repulsing supermartingales can be used for:
  - Witnessing lower bounds for termination probabilities
  - Witnessing non–almost–sure termination
  - Witnessing persistence properties (eventually stay in set of desired states)

13: Stochastic Invariants

- Ranking supermartingale intuition:
  - Program cannot avoid entering a set of states S indefinitely
- New concept: repulsing supermartingales:
  - Starting outside a set of states S, the program tends to avoid that set S
- ranking + repulsing supermartingales can be used for:
  - Witnessing lower bounds for termination probabilities
  - Witnessing non–almost–sure termination
  - Witnessing persistence properties (eventually stay in set of desired states)
- Paper also considers algorithmic problem of synthesis of (linear) martingales

14: Lambda-calculus

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...
- This paper: Semantics of probabilistic λ-calculus

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...
- This paper: Semantics of probabilistic λ-calculus
  - Distribution-based operational semantics

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...
- This paper: Semantics of probabilistic λ-calculus
  - Distribution-based operational semantics
  - Sampling-based semantics: trace of samples $\mapsto$ output

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...
- This paper: Semantics of probabilistic λ-calculus
  - Distribution-based operational semantics
  - Sampling-based semantics: trace of samples $\mapsto$ output
  - Equivalence of different semantics

14: Lambda-calculus

- Theoretical foundations of functional probabilistic programming languages
  - Church, Anglican, Venture, WebPPL, ...
- This paper: Semantics of probabilistic λ-calculus
  - Distribution-based operational semantics
  - Sampling-based semantics: trace of samples $\mapsto$ output
  - Equivalence of different semantics
- Gentle warning: You should agree with your supervisor on a subset of this paper

Thank you for your attention!
Prioritize your favorite topics NOW!