Probabilistic Programs

Expressing and Verifying Probabilistic Assertions

Dustin Hütter
Prof. Dr. Ir. Joost-Pieter Katoen

Winter term 2014/2015

Outline

Part I Introduction

Part II Verification

Part III Optimizations

Part IV Evaluation

Part V Final Judgement

Part I

Introduction

Classic Assertions

- Statements evaluating to a Boolean value
- Desired to hold at a certain point in a formal system e.g. in a Markov chain, code snippet, finite automaton...
- Huge bandwidth of techniques in formal verification to check them

Example (Assertion)

```
...
float a = 0, b = 0;
a = some_calculation_1();
b = some_calculation_2();
assert b != 0:
return a/b;
```

Probabilistic Assertions

- Classic assertions have to hold in every execution of the considered formalism
- Algorithms in machine learning, approximate and quantum computing inherently don't yield the exact same result for every execution

$\rightarrow$ Classic assertions would be too strict

Solution? Probabilistic assertions passert e p c stating that the Boolean statement e has to hold with probability p and a confidence level c

Probabilistic Programs

PROBCORE is a simple imperative probabilistic language whose grammar is given by:

$$
\begin{aligned}
P &\equiv\ S \text{ ;; passert } C \\
C &\equiv\ E < E \ \mid\ E = E \ \mid\ C \wedge C \ \mid\ C \vee C \ \mid\ \neg C \\
E &\equiv\ E + E \ \mid\ E * E \ \mid\ E \div E \mid\ R \ \mid\ V \\
S &\equiv\ V := E \mid\ V \leftarrow D \mid\ S; S \mid\ \mathit{skip} \mid\ \text{if } C\ S\ S \mid\ \text{while } C\ S \\
R &\in \mathbb{R}, V \in \text{Variables}, D \in \text{Distributions (e.g. Gaussian, uniform...)}
\end{aligned}
$$

Why do the parameters for the probability and confidence level not occur in the grammar?

$\rightarrow$ Handled one abstraction level higher

Semantics

- Big-step semantics
  - Interpret syntactic constructs in an appropriate domain
  - E.g. an arithmetic operation E evaluates to a real number R denoted by $E \Downarrow R$ ($2 + 3 \Downarrow 5$)

- Small-step semantics
  - Models computation steps of a program
  - Steps are transitions from one configuration say C_1 to another say C_2 denoted by $C_1 \rightarrow C_2$
  - Configurations include variable valuations and the program fragment to be evaluated
  - $\rightarrow^*$ denotes the transitive and reflexive closure of the transition relation

Semantics

$$? \begin{cases} P \equiv \ S \text{ ;; passert } C \end{cases}$$

$$\text{Big-steps} \begin{cases} C \equiv \ E < E \ | \ E = E \ | \ C \wedge C \ | \ C \vee C \ | \ \neg C \\ E \equiv \ E + E \ | \ E * E \ | \ E \div E \,| \ R \ | \ V \end{cases}$$

$$\text{Small-steps} \begin{cases} S \equiv \ V := E \,| \ V \leftarrow D \,| \ S; S \,| \ \mathit{skip} \,| \ \text{if } C \ S \ S \,| \ \text{while } C \ S \end{cases}$$

$$R \in \mathbb{R}, V \in \text{Variables}, D \in \text{Distributions}$$

We will see a subset of the inference rules that model a concrete execution of a *PROBCORE* program

Concrete Semantics

What is concrete at the following semantics?

- Take random draws when variables are assigned with probabilistic values and proceed straight-forward with control flow
- In contrast, we will later see a symbolic approach

Expression Evaluation

Heap H for variable valuations

Arithmetic operations with $\circ \in \{+, *, \div\}$:

$$\frac{(H, e_1) \Downarrow_c v_1 \quad (H, e_2) \Downarrow_c v_2}{(H, e_1 \circ e_2) \Downarrow_c v_1 \circ v_2}$$

Conditions with $\circ' \in \{\wedge, \vee\}$:

$$\frac{(H, c_1) \Downarrow_c b_1 \quad (H, c_2) \Downarrow_c b_2}{(H, c_1 \circ' c_2) \Downarrow_c b_1 \circ' b_2}$$

Statement Semantics

Sequence of draws Σ for generation of random samples

Sample statements:

$$\frac{\Sigma = \sigma : \Sigma'}{(\Sigma, H, v \leftarrow d) \rightarrow_c (\Sigma', (v \mapsto d(\sigma)) : H, \text{ skip})}$$

If statements:

$$\frac{(H, c) \Downarrow_c \text{ true}}{(\Sigma, H, \text{if c } s_1 s_2) \Downarrow_c (\Sigma, H, s_1)}$$

Statement Semantics

Loops:

$$\frac{}{(\Sigma, H, \text{while c s}) \rightarrow_c (\Sigma, H, \text{if c (s ; while c s) skip})}$$

Passert:

$$\frac{(\Sigma, H_0, s) \rightarrow_c^* (\Sigma', H', \text{skip}) \quad (H', c) \Downarrow_c b}{(\Sigma, H_0, s\ ;\ ;\ \text{passert}\ c) \Downarrow_c b}$$

Concrete Semantics

Naive decision procedure:

- Execute a PROBCORE program P under the concrete semantics several times
- Compare the relative share r where the condition e of the passert e p c is met with p
- When $r \geq p$ holds return true, otherwise return false

Concrete Semantics

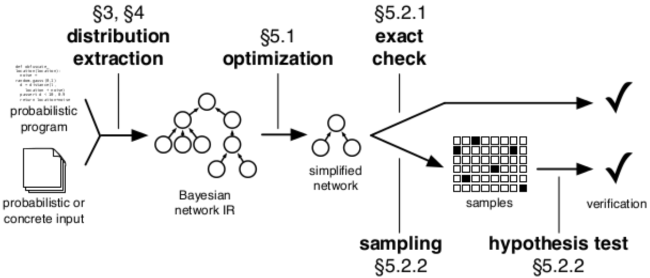
Problems of this approach:

- Repetition of redundant deterministic computations
- We might only have to consider those parts of a program that contribute to the passert
- Possible reductions of a program not exploited

Part II

Verification

General Scheme



→ Implementation in a tool called *MAYHAP*

Distribution Extraction

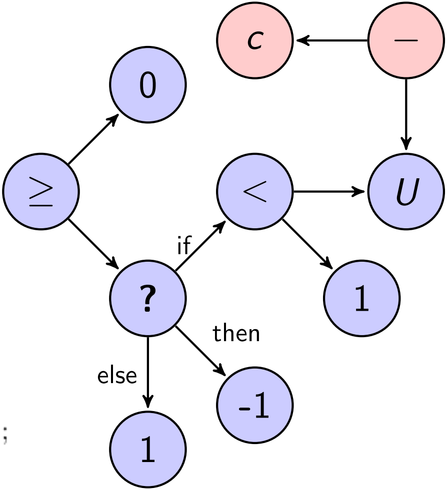
Idea:

- Instead of taking random draws, represent probabilistic values symbolically by their according distribution
- Evaluate deterministic parts concretely and keep probabilistic parts symbolically
- *Bayesian network* is extracted from *Expression DAG*

$\rightarrow$ How to generate the Expression DAG?

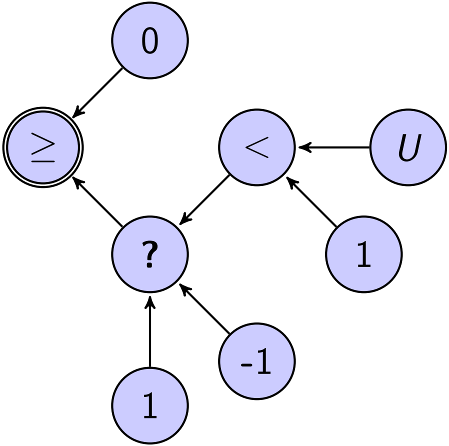
Distribution Extraction

$v_1 := \texttt{detProc}(a);$
$v_2 := \texttt{detProc}(b);$
$v_2 := v_2 + v_1;$
$v_3 \leftarrow \texttt{Unif}(0, 2);$
$v_4 := v_3 - v_2;$
$v_5 := 0;$
$\textbf{if}\ v_3 < 1 \quad v_5 := -1 \quad v_5 := 1;;$
$\texttt{passert}\ v_5 \geq 0$



Distribution Extraction

- Bayesian network is obtained by reverting the direction of the edges of the Expression DAG
- Nodes model random variables
- Edges capture the dependencies between the random variables
- Constants are modeled by point-mass distributions



Handling Loops

Problem:

Loops can induce cycles in the Bayesian network violating the DAG property

Example (Repeated coin flip)

$v_1 \leftarrow \mathrm{Bernoulli}(0.5)$;
while $v_1 = \mathrm{HEAD}$ $v_1 \leftarrow$ Bernoulli(0.5);

Handling Loops

Symbolic handling of loops:

- Implementation:
  - Distribution generation for loops with 'deterministic conditions' is often possible
  - Otherwise path pruning: Do not consider paths having a lower probability than some threshold
- Formalization:
  - Symbolic approach only handles terminating loops with deterministic conditions

$\rightarrow$ Extension e.g. non-terminating loops is left to future work

Expression/Statement Semantics

Values in the symbolic semantics correspond to expression trees denoted by curly braces. Exemplary consider:

Arithmetic operations with $\circ \in \{+, *, \div\}$:

$$\frac{(H, e_1) \Downarrow_s \{x_1\} \quad (H, e_2) \Downarrow_s \{x_2\}}{(H, e_1 \circ e_2) \Downarrow_s \{x_1 \circ x_2\}}$$

Sample statement:

$$\frac{}{(n, H, v \leftarrow d) \rightarrow_s (n+1, (v \mapsto \{(d, n)\}) : H, \mathrm{skip})}$$

$\{x_1 \circ x_2\}$ denotes an element in the expression tree where the curly braces indicate delayed evaluation

Expression/Statement Semantics

If statements:

$$\frac{(H,c) \Downarrow_s \{x\} \quad (n,H,b_t) \rightarrow_s^* (m_t,H_t,\text{skip}) \quad (n,H,b_f) \rightarrow_s^* (m_f,H_f,\text{skip})}{(n,H,\text{if } c\ b_t\ b_f) \rightarrow_s (\{\text{if } x\ m_t\ m_f\}, merge(H_t,H_f,\{x\}),\text{skip})}$$

While loops:

$$\frac{(H,c) \Downarrow_s \{x\} \quad \forall\Sigma(\Sigma,\{x\}) \Downarrow_o \text{false}}{(n,H,\text{while } c\ s) \rightarrow (n,H,\text{skip})}$$

Passert:

$$\frac{(0,H_0,s) \rightarrow_s^* (n,H',\text{skip}) \quad (H',c) \Downarrow_s \{x\}}{(H_0,s\ ;\ ;\ \text{passert}\ c) \Downarrow_s \{x\}}$$

Evaluation Relation

- The symbolic semantics yields an expression tree $\{x\}$ for the condition of the corresponding passert
- $\{x\}$ is evaluated by $\Downarrow_o$ for a given Σ denoted by $(\Sigma,\{x\}) \Downarrow_o v$

Exemplary consider ($\circ \in \{+, *, \div\}$):

$$\frac{(\Sigma, e_1) \Downarrow_o v_1 \quad (\Sigma, e_2) \Downarrow_o v_2}{(\Sigma, e_1 \circ e_2) \Downarrow_o v_1 \circ v_2}$$

$$\frac{}{(\Sigma,(d,k)) \Downarrow_o d(\sigma_k)}$$

Concrete vs. Symbolic Evaluation

$x \leftarrow \mathrm{Gauss}(0, 1)\,;$
$\mathbf{if}\;\; x > 0.1 \quad x := 1 \quad x := -1 \;\;;;$
$\mathtt{passert}\;\; x = 1$

Premise 1:
$(\Sigma = \sigma_0 : \Sigma', \emptyset, x \leftarrow \mathrm{Gauss}(0{,}1)) \rightarrow_c$
$(\Sigma', \{x \mapsto d_G(\sigma_0) = 0.2\}, \mathrm{skip}) \rightarrow_c$
$(\Sigma', \{x \mapsto 0.2\}, \text{if } x > 0.1 \; x := 1 \; x := -1) \rightarrow_c$
$(\Sigma', \{x \mapsto 0.2\}, \text{if } 0.2 > 0.1 \; x := 1 \; x := -1) \rightarrow_c$
$(\Sigma', \{x \mapsto 0.2\}, \text{if true } x := 1 \; x := -1) \rightarrow_c$
$(\Sigma', \{x \mapsto 0.2\}, x := 1) \rightarrow_c$
$(\Sigma', \{x \mapsto 1\}, \mathrm{skip})$

Premise 2:
$(\{x \mapsto 1\}, x = 1) \Downarrow_c \text{true}$

$x \leftarrow \mathrm{Gauss}(0,1)\,;$
$\mathbf{if}\ \ x > 0.1 \quad x := 1 \quad x := -1 \ \ ;\,;$
$\mathtt{passert}\ \ x = 1$

Premise 1:
$(0, \emptyset, x \leftarrow \mathrm{Gauss}(0{,}1)) \rightarrow_s$
$(1, \{x \mapsto \{(d_G{,}0)\}\}, \mathrm{skip}) \rightarrow_s$
$(1, \{x \mapsto \{(d_G{,}0)\}\}, \text{if } x > 0.1\ x := 1\ x := -1) \rightarrow_s$
$(\{\text{if } (d_G{,}0) > 0.1\ 1\ 1\}, \mathit{merge}(\{x \mapsto \{1\}\},\{x \mapsto \{-1\}\},\{(d_G{,}0) > 0.1\}), \mathrm{skip}) \rightarrow_s$
$(\{\text{if } (d_G{,}0) > 0.1\ 1\ 1\}, \{x \mapsto \{\text{if } (d_G{,}0) > 0.1\ 1\ \ -1\}\}, \mathrm{skip})$

Premise 2:
$(\{x \mapsto \{\text{if } (d_G{,}0) > 0.1\ 1\ \ -1\}\}, x = 1) \Downarrow_s \{\text{if } (d_G{,}0) > 0.1\ 1\ \ -1 = 1\}$

$\rightarrow (\Sigma,\{\text{if } (d_G{,}0) > 0.1\ 1\ \ -1 = 1\}) \Downarrow_o \text{true}$

Final Result

Theorem

Let $(0,H_0,p) \Downarrow_s \{x\}$, where x is a finite (terminating) program. Then $(\Sigma,H_0,p) \Downarrow_c b$ if and only if $(\Sigma,x) \Downarrow_o b$.

Proof by structural induction

Intuition:

Concrete evaluation of a program yields the same result as the evaluation of the extracted distribution

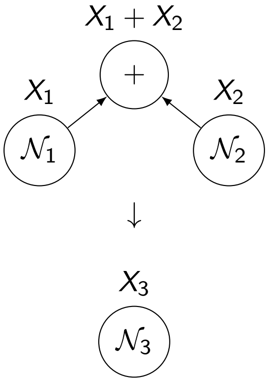
Part III

Optimizations

Intention

- Exploit stochastic knowledge in order to reduce the Bayesian network
- Direct verification if possible
- Otherwise, sample the optimized Bayesian network

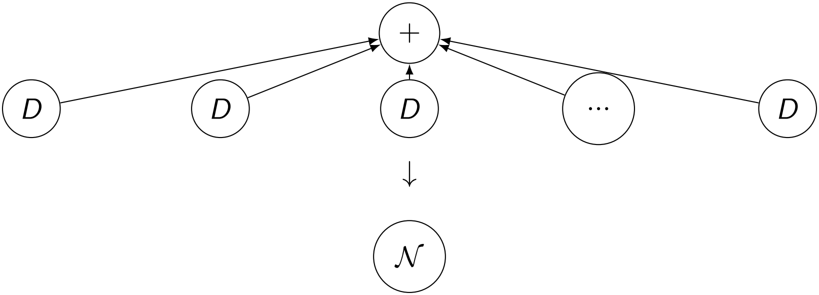
Arithmetic Operations on Common Distributions



$X_1 \sim \mathcal{N}_1(\mu_{X_1} = 1, \sigma^2_{X_1} = 16) \wedge X_2 \sim \mathcal{N}_2(\mu_{X_2} = 5, \sigma^2_{X_2} = 9) \Rightarrow X_1 + X_2 = X_3 \sim \mathcal{N}_3(\mu_{X_1} + \mu_{X_2} = 6, \sigma^2_{X_1} + \sigma^2_{X_2} = 25)$

Central Limit Theorem (CLT)

CLT: "The sum of a large amount of independent random variables that are identically distributed and have a finite expected value and variance converges to a normal distribution."



Sampling Approach

Idea:

- Given a passert *e p c*, its satisfaction can be modeled by a *Bernoulli* variable (either satisfied or not)
- Take n samples X_i with $i \in \{1,...,n\}$ for the passert and estimate p by $p_\sim = \frac{1}{n}\sum_{i=1}^{n} X_i$

Question:

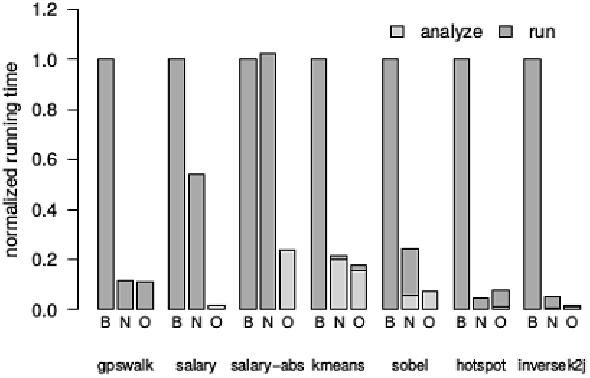
How many samples n are needed in order to satisfy the desired accuracy ϵ and confidence α, respectively does $Pr(p_\sim \in [p-\epsilon, p+\epsilon]) \geq 1-\alpha$ hold?

$\rightarrow$ Using the *two-sided Chernoff-bound* yields $n \geq \frac{2+\epsilon}{\epsilon^2} ln(\frac{2}{\alpha})$

Part IV

Evaluation

Evaluation



B: stress testing, N: unoptimized symbolic approach, O: optimized symbolic approach

Part V

Final Judgement

Final Judgement

Advantages:

- Promising results on considered benchmarks
- Eliminating redundant deterministic computation & parts not contributing to a passert
- Reduction of the obtained model by applying stochastic knowledge

Disadvantages:

- Formalization of loop handling is very rough
- Soundness proof for the symbolic approach on the optimized Bayesian network is missing
- Partially sloppy formalization

Thanks for your attention!

Questions?