Program Analysis for System Security and Reliability

Martin Vechev
Spring 2018

http://www.srl.inf.ethz.ch

http://ice.ethz.ch

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networks: so far

In lecture 2, we studied **automated analysis of configurations** and specifically the Batfish system. With this approach, we can automatically check whether a specific configuration is correct.

In lecture 3, we studied **automated synthesis of configurations** and specifically the SyNET system. In this way we can automatically generate the desired configuration!

In this lecture, we will study **probabilistic analysis of network configurations**, showing how to use probabilistic programming to prove key properties under **uncertainty** (e.g., network link failures). In particular we will cover the Bayonet system.

Probabilistic Programs are…

"…usual functional or imperative programs with two added constructs:

the ability to draw values at random from distributions,

and the ability to condition values of variables in a program via observations…"

"Probabilistic programming." In Proceedings of On The Future of Software Engineering (2014)., Gordon, Henzinger, Nori, and Rajamani

Probabilistic Programs

Extend Standard (Deterministic) Programs

Distribution	`X := Uniform(0, 1);`
Assertion	`assert ( X >= 0 );`
Observation	`observe ( X >= 0.5 );`
Query	`return X;`

Observations and Assertions

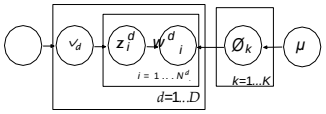
Example1:

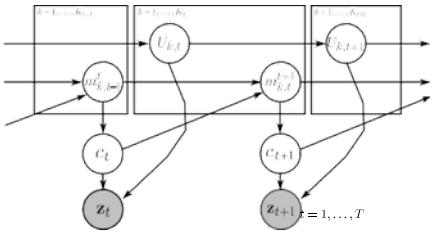
```
X := Uniform(0, 1);
observe ( X > 0.5 );
return X;
```

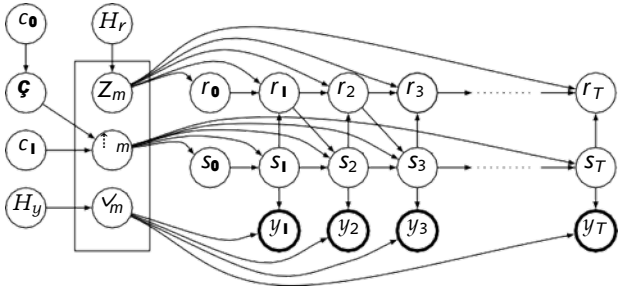
Example 2:

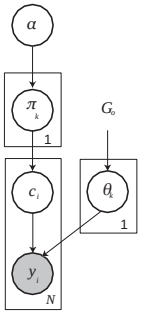
```
X := Uniform(0, 1);
assert ( X > 0.5 );
return X;
```

Probabilistic Models



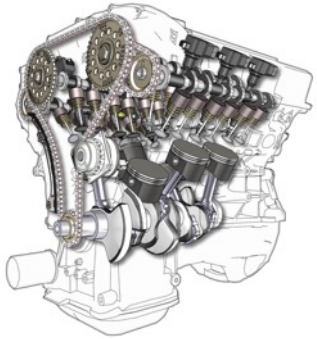






Language Representation / Abstraction Layer

Inference engines



Inference Engines

Approximate Inference

Sampling (Rejection – Church)

Sampling (MCMC – Church & Stan & WebPPL & PSI)

Variational Inference (Fun & Infer.NET)

Exact Inference

Symbolic Inference (PSI, Hakaru)

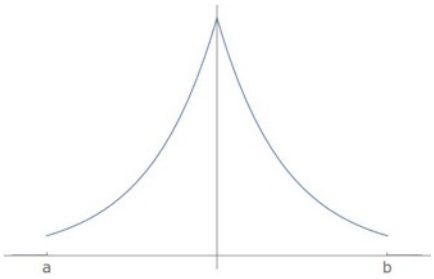
Discrete Inference (ProbLog): for discrete inference

What is the Probability Density:

$Z = X1 + X2 + X3$

X1, X2, X3
i.i.d.

Truncated Laplace



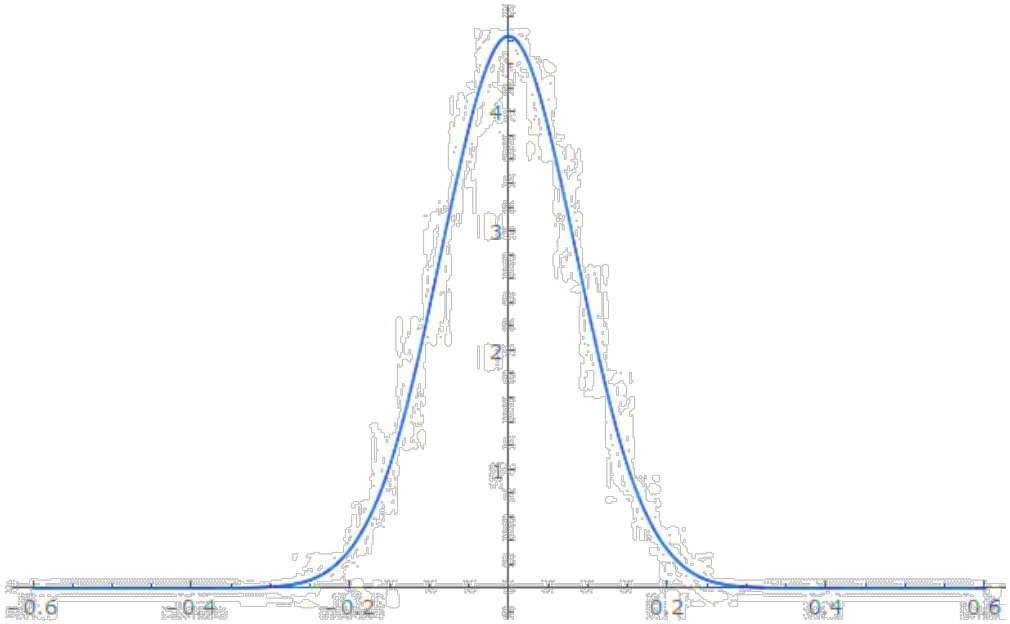
Z

?

Exact Final Distribution (PSI)

$ psi sum_laplace.prb

$$p(x) = -[x\neq 0]\cdot[x\leq 0]\cdot e^{x}\cdot x\cdot 1/16+-[x\leq 0]\cdot e^{x}\cdot x\cdot 1/8+3/32\cdot[-x\leq 0]\cdot[x\neq 0]\cdot 1/e^{x}+3/32\cdot[-x\leq 0]\cdot 1/e^{x}+3/32\cdot[x\neq 0]\cdot[x\leq 0]\cdot e^{x}+3/32\cdot[x\leq 0]\cdot e^{x}+[-x\leq 0]\cdot[x\neq 0]\cdot x\cdot 1/16\cdot 1/e^{x}+[-x\leq 0]\cdot x^{2}\cdot 1/16\cdot 1/e^{x}+[x\leq 0]\cdot x\cdot 1/8\cdot 1/e^{x}+[x\leq 0]\cdot e^{x}\cdot x^{2}\cdot 1/16$$



How to use existing probabilistic programming engines (PPE)?

Probability/Noise

The non-probabilistic problem: systems, security, networks

Step 1: define a probabilistic DSL to capture domain

Step 2: ``Compile'' programs in DSL to programs in existing PPE

Translate

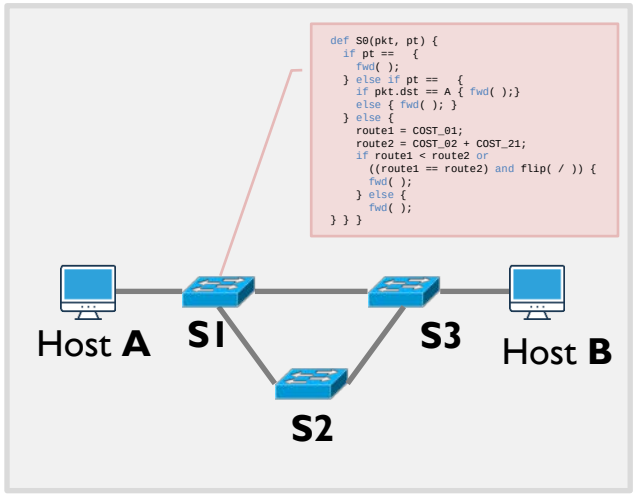
Perform inference using existing PPE (e.g., PSI, WebPPL)

benefit: reuse all advances in probabilistic programming

The above is a good first step when we approach a problem (to not reinvent the wheel). However, once done, if we see the need, we can design **custom probabilistic inference procedures** for the particular application domain and we then have a good first baseline to compare against.

Application: Probabilistic Network Analysis (the Bayonet system)

Bayonet DSL



Translate

PSI Program

```
dat __D{
  __H0 : __h0_ty,
  __H1 : __h1_ty,
  __S0 : __s0_ty,
  __S1 : __s1_ty,
  __S2 : __s2_ty;
  curAction: ℝ;
  def __D(){
    __H0 = __h0_ty(), __H1 = __h1_ty(), __S0 = __s0_ty(),
    __S1 = __s1_ty(), __S2 = __s2_ty();'
    curAction = 0;
  }
  def scheduler(){
    /*action := if flip(1/2) {RunSw;} else {FwdQ;};
   ...
   }
}
def main(){
  __d := __D();
  __d.__H0.__run();
  repeat num_steps {
    __d.__step();
  }
  assert(!(__d.__H0.Q_in.size() || __d.__H0.Q_out.size()
          || __d.__H1.Q_in.size() || __d.__H1.Q_out.size()
          || __d.__S0.Q_in.size() || __d.__S0.Q_out.size()
          || __d.__S1.Q_in.size() || __d.__S1.Q_out.size()
          || __d.__S2.Q_in.size() || __d.__S2.Q_out.size()));
  q₁ := Expectation(((if H1 == 0 { __d.__H0.pkt_count }
                      else if H1 == 1 { __d.__H1.pkt_count }
                      else { assert(0) }) < 2) !=0);
  return (q₁);
}
...
```

PSI

Result

0.442

Property

Pr [pktcnt@B < 3]

Leverages existing probabilistic solvers

Part I: We look at exact discrete probabilistic inference

Part II: We study Bayonet and see examples in networks

Simple Probabilistic Language

$$
\begin{array}{lcl}
r & \in & \mathbb{R} \\
x & \in & \text{Vars} \\
\mathcal{T} & ::= & \texttt{bool} \\
\textbf{uop} & ::= & \texttt{not} \\
\textbf{bop} & ::= & \texttt{and} \mid \texttt{or} \\
\\
\mathcal{D} & ::= & \mid \mathcal{T}\, x_1, x_2, \ldots, x_n \\
\\
\mathcal{E} & ::= & \\
 & & x \\
 & & \mid c \\
 & & \mid \mathcal{E}_1 \ \textbf{bop} \ \mathcal{E}_2 \\
 & & \mid \textbf{uop} \ \mathcal{E}_1
\end{array}
\qquad
\begin{array}{lcl}
\mathcal{S} & ::= & \\
 & & x := \mathcal{E} \\
 & & \mid x := \texttt{Bernoulli}(r) \\
 & & \mid \texttt{observe}\ (\mathcal{E}) \\
 & & \mid \texttt{skip} \\
 & & \\
 & & \mid \mathcal{S}_1 ; \mathcal{S}_2 \\
 & & \mid \texttt{if}\ \mathcal{E}\ \texttt{then}\ \mathcal{S}_1\ \texttt{else}\ \mathcal{S}_2 \\
 & & \mid \texttt{while}\ \mathcal{E}\ \texttt{do}\ \mathcal{S}_1 \\
 & & \\
\mathcal{P} & ::= & \mathcal{D}\ \mathcal{S}
\end{array}
$$

Claret et al. Bayesian inference using data flow analysis (FSE'13)

Analysis of Probabilistic Programs

$$\sigma = (v_1, v_2, \ldots v_n) \in \Sigma = \{\mathbf{True}, \mathbf{False}\}^n$$

State: ~~$\sigma = \{x_1 \to v_1, \ldots, x_n \to v_n\} \in \Sigma$~~

Distribution: $\varphi \in \Phi = \Sigma \to [0, 1]$ (probability mass f.)

Analysis: $Z: \Phi \times S \to \Phi$ (distribution transformer)

Analysis Input:

(1) Input distribution φ_i over the states of the program P

(2) Statement (or a program), s

Analysis Output:

Output distribution φ_o over the states of the program P

Skip

Semantics $(\text{skip}, \sigma) \longrightarrow (\sigma, 1.0)$

Analysis intuition Skip statement does not change the distribution over states

Analysis $\mathsf{Z}(\varphi, skip) = \lambda\, \sigma.\ \varphi(\sigma)$

Bernoulli

Semantics

$$(x = \mathrm{Bern}(p_B), \sigma) \xrightarrow{p_B} \sigma[x \leftarrow \mathrm{True}]$$
$$(x = \mathrm{Bern}(p_B), \sigma) \xrightarrow{1-p_B} \sigma[x \leftarrow \mathrm{False}]$$

Analysis intuition

(1) Find all initial states that after the statement result in our final state with value (a) true, (b) false

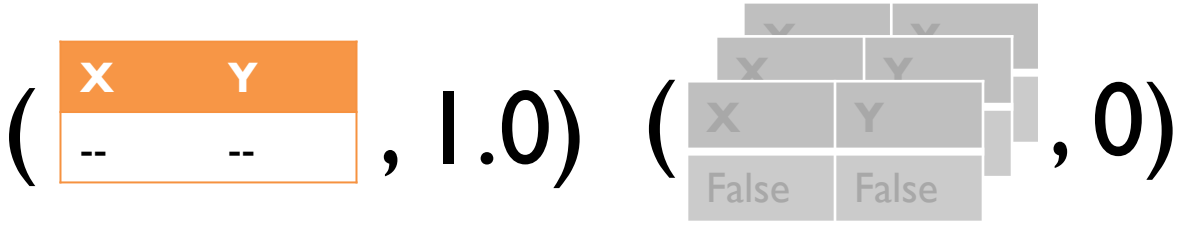
(2) Compute weighted sum of probabilities of the initial states multiplied by the transition probability

Analysis

$$\mathrm{Z}(\varphi, x = \mathrm{Bern}(p_B)) =$$
$$\lambda\, \sigma.\, p_B \cdot \sum \varphi(\sigma') + (1 - p_B) \cdot \sum \varphi(\sigma'')$$

s.t. $\sigma' \in \{\sigma' \mid \sigma'[x \leftarrow True] == \sigma\}$
$\sigma'' \in \{\sigma'' \mid \sigma''[x \leftarrow False] == \sigma\}$

Example: Bernoulli




```
X := Bernoulli(0.7);
```

$$Z(\varphi, x = \mathrm{Bern}(p_B)) =$$
$$\lambda\,\sigma.\, p_B \cdot \sum\varphi(\sigma') + (1 - p_B) \cdot \sum\varphi(\sigma'')$$

s.t. $\sigma' \in \{\sigma' \mid \sigma'[x \leftarrow True] == \sigma\}$
$\sigma'' \in \{\sigma'' \mid \sigma''[x \leftarrow False] == \sigma\}$

Example: Bernoulli

(

, 1.0)

```
X := Bernoulli(0.7);
```

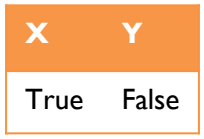
(

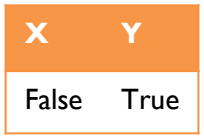
, 0.7), (

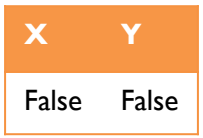
, 0.3)

```
Y := Bernoulli(0.7);
```

(

, 0.49), (

, 0.21), (

, 0.21), (

, 0.09)

Deterministic Assignment

Semantics $(x = e, \sigma) \longrightarrow (\sigma[x \leftarrow A(e)], 1.0)$

Analysis intuition

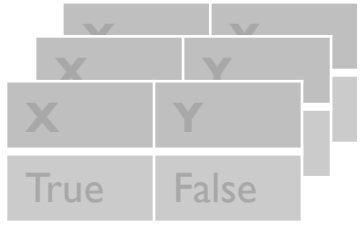
(1) Find all initial states that after the statement result in our final state with value $A(e, \sigma)$

(2) Do the weighted sum of the initial probabilities multiplied by the transition probability

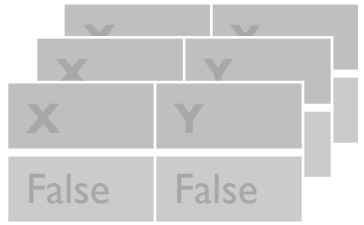
Analysis $\mathsf{Z}(\varphi, x = e) = \lambda\, \sigma\, .\ \ \sum \varphi(\sigma')$

s.t. $\sigma' \in \{\sigma' \mid \sigma'[x \leftarrow A(e, \sigma')] == \sigma\}$

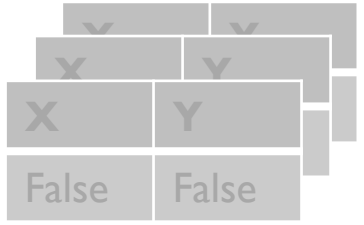
Example: Bernoulli

(| X | Y | | -- | -- | , 1.0) ( , 0)

```
X := Bernoulli(0.7);
```

(| X | Y | | True | -- | , 0.7), (| X | Y | | False | -- | , 0.3) ( , 0)

```
Y := not X;
```

(| X | Y | | True | False | , 0.7), (| X | Y | | False | True | , 0.3) ( , 0)

Control Flow

Sequence $\mathrm{Z}(\varphi, S1; S2) = \mathrm{Z}(\,\mathrm{Z}(\varphi, S1), S2\,)$

Observation $\mathrm{Z}(\varphi, o\mathrm{bserve}\ e) =$

$\lambda\,\sigma\,.\,\varphi(\sigma)$ if $A(e,\sigma) = \mathrm{True}$ else 0

Normalization done at the end of the analysis

Condition $\mathrm{Z}(\varphi, \mathrm{if}\ e\ \mathrm{then}\ S1\ \mathrm{else}\ S2) =$

$\lambda\,\sigma\,.\,\mathrm{let}\ \varphi_t = \lambda\sigma'.\,\varphi(\sigma')$ if $A(e,\sigma) = \mathrm{True}$ else 0

and $\varphi_f = \lambda\sigma'.\,\varphi(\sigma')$ if $A(e,\sigma) = \mathrm{False}$ else 0

in $\mathrm{Z}(\varphi_t, S1)(\sigma) + \mathrm{Z}(\varphi_f, S2)(\sigma)$

Normalization with Observations

✓ - observe is true for these

✗ - observe is false for these

Normalize:

$$\text{Pr\_new}(✓) = \frac{\text{Pr}(✓)}{\sum \text{Pr}(✓)}$$

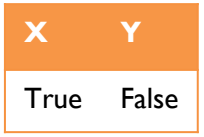
Example: Bernoulli Program

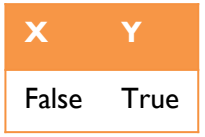
(

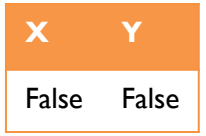
, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

, 0.49), (

, 0.21), (

, 0.21), (

, 0.09)

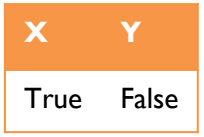
Example: Bernoulli Program

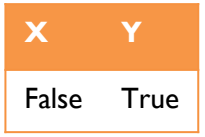
(

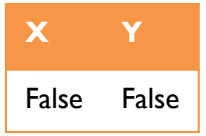
, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

, 0.49), (

, 0.21), (

, 0.21), (

, 0.09)

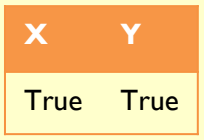
```
observe ( X == True );
return Y;
```

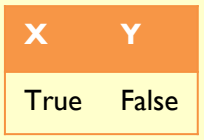
Example: Bernoulli Program

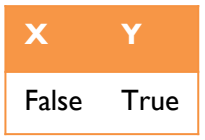
(

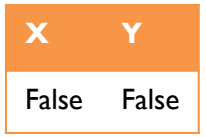
, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

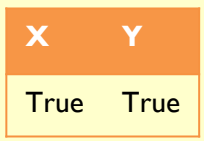
, 0.49), (

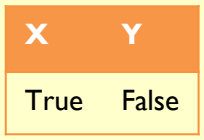
, 0.21), (

, 0.21), (

, 0.09)

```
observe ( X == True );
return Y;
```

(

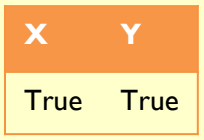
, 0.49/0.7), (

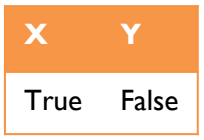
, 0.21/0.7),

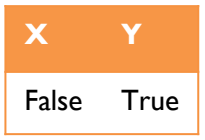
Example: Bernoulli Program

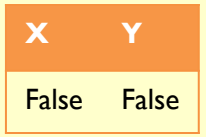
, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

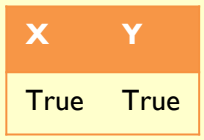
, 0.49), (

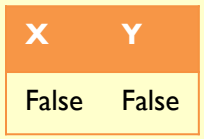
, 0.21), (

, 0.21), (

, 0.09)

```
observe ( X == Y );
return Y;
```

(

, 0.49/0.58), (

, 0.09/0.58)

We should not normalize right away when we see observe but re-normalize at the end of the program.

[Example on Board]

Part I: We look at exact discrete probabilistic inference

Part II: We study Bayonet and see examples in networks