From Probabilistic Circuits to Probabilistic Programs and Back

Guy Van den Broeck

RWTH Aachen - Oct 6, 2020
Let me be provocative

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.
Let me be provocative

Probabilistic graphical models is how we do probabilistic AI!

*Graphical models of variable-level (in)dependence are a broken abstraction.*

\[3.14\] \text{Smokes}(x) \land \text{Friends}(x,y) \Rightarrow \text{Smokes}(y)\]
Let me be provocative

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.

Bean Machine

\[\begin{align*}
\mu_k &\sim \text{Normal}(\alpha, \beta) \\
\sigma_k &\sim \text{Gamma}(\nu, \rho) \\
\theta_k &\sim \text{Dirichlet}(\kappa) \\
x_i &\sim \begin{cases} 
\text{Categorical}(\text{init}) & \text{if } i = 0 \\
\text{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 
\end{cases} \\
y_i &\sim \text{Normal}(\mu_{x_i}, \sigma_{x_i})
\end{align*}\]
Let me be provocative

We may have gotten stuck in a local optimum

The choice of representing a distribution primarily by its variable-level (in)dependencies is a little arbitrary…

What if we made some different choices?
Computational Abstractions

*Let us think of probability distributions as objects that are computed.*

Abstraction = Structure of Computation

‘closer to the metal’

Two examples:
1. Probabilistic Circuits
2. Probabilistic Programs
Probabilistic Circuits
The Alphabet Soup of probabilistic models
Intractable and tractable models
"Every talk needs a joke and a literature overview slide, not necessarily distinct"
- after Ron Graham
a unifying framework for tractable models
Input nodes $c$ are tractable (simple) distributions, e.g., univariate gaussian or indicator $p_c(X=1) = [X=1]$
Product nodes are factorizations $\prod_{c \in \text{in}(n)} p_c(x)$
Sum nodes are mixture models $\sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x)$
If $p(x) = \sum_i w_i p_i(x)$, (smoothness):

$$\int p(x) dx = \int \sum_i w_i p_i(x) dx =$$

$$= \sum_i w_i \int p_i(x) dx$$

⇒ integrals are “pushed down” to children

[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]
Smoothness + decomposability = tractable MAR

If \( p(x, y, z) = p(x)p(y)p(z) \), (decomposability):

\[
\int \int \int p(x, y, z) \, dx \, dy \, dz = \\
= \int \int \int p(x)p(y)p(z) \, dx \, dy \, dz = \\
= \int p(x) \, dx \int p(y) \, dy \int p(z) \, dz
\]

\[\Rightarrow \text{ integrals decompose into easier ones} \]
Smoothness + decomposability = tractable MAR

Forward pass evaluation for MAR

\[ \Rightarrow \text{linear in circuit size!} \]

E.g. to compute \( p(x_2, x_4) \):
- leafs over \( X_1 \) and \( X_3 \) output \( Z_i = \int p(x_i) dx_i \)
  \[ \Rightarrow \text{for normalized leaf distributions: 1.0} \]
- leafs over \( X_2 \) and \( X_4 \) output \( \text{EVI} \)
- feedforward evaluation (bottom-up)
<table>
<thead>
<tr>
<th></th>
<th>MAR</th>
<th>CON</th>
<th>MOM</th>
<th>MAP</th>
<th>MMAP</th>
<th>ENT</th>
<th>DIV</th>
<th>EXP</th>
<th>expected predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>smoothness</td>
<td>SMO</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>decomposability</td>
<td>DEC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>consistency</td>
<td>CON</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>determinism</td>
<td>DET</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>marginal determinism</td>
<td>MAR-DET</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>structured decomposability</td>
<td>STR-DEC</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>paired str. decomposability</td>
<td>P-STR-DEC</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
tractability is a spectrum
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Circuits (ACs) [Darwiche 2003]</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Sum-Product Networks (SPNs) [Poon et al. 2011]</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Cutset Networks (C Nets) [Rahman et al. 2014]</td>
<td>✅</td>
<td>✅</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Probabilistic Decision Graphs [Jaeger 2004]</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>(Affine) ADDs [Hoey et al. 1999; Sanner et al. 2005]</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>AndOrGraphs [Dechter et al. 2007]</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>PSDDs [Kisa et al. 2014a]</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
Expressive models without compromises
## How expressive are probabilistic circuits?

### density estimation benchmarks

<table>
<thead>
<tr>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nltcs</td>
<td>-5.99</td>
<td>-6.02</td>
<td>-6.04</td>
<td>-5.99</td>
<td>dna</td>
<td>-79.88</td>
<td>-80.65</td>
<td>-82.77</td>
<td>-94.56</td>
</tr>
<tr>
<td>msnbc</td>
<td>-6.04</td>
<td>-6.04</td>
<td>-6.06</td>
<td>-6.09</td>
<td>kosarek</td>
<td>-10.52</td>
<td>-10.83</td>
<td>-</td>
<td>-10.64</td>
</tr>
<tr>
<td>plants</td>
<td>-11.84</td>
<td>-12.65</td>
<td>-12.32</td>
<td>-12.34</td>
<td>book</td>
<td>-33.82</td>
<td>-36.41</td>
<td>-33.95</td>
<td>-33.19</td>
</tr>
<tr>
<td>audio</td>
<td>-39.39</td>
<td>-40.50</td>
<td>-38.95</td>
<td>-38.67</td>
<td>movie</td>
<td>-50.34</td>
<td>-54.37</td>
<td>-48.7</td>
<td>-47.43</td>
</tr>
<tr>
<td>jester</td>
<td>-51.29</td>
<td>-51.07</td>
<td>-52.23</td>
<td>-51.54</td>
<td>webkb</td>
<td>-149.20</td>
<td>-157.43</td>
<td>-149.59</td>
<td>-146.9</td>
</tr>
<tr>
<td>netflix</td>
<td>-55.71</td>
<td>-57.02</td>
<td>-55.16</td>
<td>-54.73</td>
<td>cr52</td>
<td>-81.87</td>
<td>-87.56</td>
<td>-82.80</td>
<td>-81.33</td>
</tr>
<tr>
<td>accidents</td>
<td>-26.89</td>
<td>-26.32</td>
<td>-26.42</td>
<td>-29.11</td>
<td>c20ng</td>
<td>-151.02</td>
<td>-158.95</td>
<td>-153.18</td>
<td>-146.9</td>
</tr>
<tr>
<td>retail</td>
<td>-10.72</td>
<td>-10.87</td>
<td>-10.81</td>
<td>-10.83</td>
<td>bbc</td>
<td>-229.21</td>
<td>-257.86</td>
<td>-242.40</td>
<td>-240.94</td>
</tr>
</tbody>
</table>
Want to learn more?

Tutorial (3h)

Probabilistic Circuits

Inference Representations Learning Theory

Antonio Vergari
University of California, Los Angeles

Robert Peharz
TU Eindhoven

Yoosung Choi
University of California, Los Angeles

Guy Van den Broeck
University of California, Los Angeles

September 14th, 2020 - Ghent, Belgium - ECML-PKDD 2020

https://youtu.be/2RAG5-L9R70

Overview Paper (80p)

Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models

YooJung Choi
Antonio Vergari
Guy Van den Broeck

Computer Science Department
University of California
Los Angeles, CA, USA

Contents

1 Introduction 3

2 Probabilistic Inference: Models, Queries, and Tractability 4
   2.1 Probabilistic Models 5
   2.2 Probabilistic Queries 6
   2.3 Tractable Probabilistic Inference 8
   2.4 Properties of Tractable Probabilistic Models 9

Training PCs in Julia with Juice.jl

Training maximum likelihood parameters of probabilistic circuits

```julia
julia> using ProbabilisticCircuits;
julia> data, structure = load(...);
julia> num_examples(data)
17412
julia> num_edges(structure)
270448
julia> @btime estimate_parameters(structure, data);
  63 ms
```

Custom SIMD and CUDA kernels to parallelize over layers and training examples.

[https://github.com/Juice-jl/]
Probabilistic circuits seem awfully general.

Are all tractable probabilistic models probabilistic circuits?
Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

\[
L = \begin{bmatrix}
1 & 0.9 & 0.8 & 0 \\
0.9 & 0.97 & 0.96 & 0 \\
0.8 & 0.96 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{\{1,2\}})
\]

Computing marginal probabilities is *tractable.*

[Zhang et al. UAI20]
We cannot tractably represent DPPs with classes of PCs

- PSDDs: More Tractable
- Deterministic and Decomposable PCs: No
  - Deterministic PCs with no negative parameters: No
  - Deterministic PCs with negative parameters: No
  - Decomposable PCs with no negative parameters (SPNs): No
  - Decomposable PCs with negative parameters: We don’t know

Stay Tuned!

[Zhang et al. UAI20; Martens & Medabalimi Arxiv15]
The AI Dilemma

Pure Logic  Pure Learning
The AI Dilemma

Pure Logic

- Slow thinking: deliberative, cognitive, model-based, extrapolation
- Amazing achievements until this day
- “Pure logic is brittle”
  - noise, uncertainty, incomplete knowledge, …

Pure Learning
The AI Dilemma

Pure Logic

• Fast thinking: instinctive, perceptive, model-free, interpolation
• Amazing achievements recently
• “Pure learning is brittle”

Pure Learning

bias, algorithmic fairness, interpretability, explainability, adversarial attacks, unknown unknowns, calibration, verification, missing features, missing labels, data efficiency, shift in distribution, general robustness and safety fails to incorporate a sensible model of the world
"Pure learning is brittle"

bias, **algorithmic fairness**, interpretability, **explainability**, adversarial attacks, unknown unknowns, calibration, verification, **missing features**, missing labels, data efficiency, shift in distribution, general robustness and safety

fails to incorporate a sensible model of the world
Prediction with Missing Features

Train

Classifier

Predict

Test with missing features
Expected Predictions

Consider all possible complete inputs and reason about the expected behavior of the classifier

\[ \mathbb{E}_{x^m \sim p(x^m|x^o)} \left[ f \left( x^m, x^o \right) \right] \]

\( x^o = \) observed features
\( x^m = \) missing features

Experiment:
- \( f(x) = \) logistic regres.
- \( p(x) = \) naive Bayes

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
What about complex feature distributions?

- feature distribution is a compatible probabilistic circuits
- classifier is a regression circuit

Recursion that “breaks down” the computation.

Expectation of function \( m \) w.r.t. dist. \( n \)?

Solve subproblems: \((1,3), (1,4), (2,3), (2,4)\)

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
Experiments with Probabilistic Circuits

[Graphs showing accuracy and RMSE vs. % Missing for MNIST, FMNIST, Abalone, Delta, and Insurance datasets.]

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);

q9: Is the predictive model biased by gender?
groups = make_observations(["male", "female"])
exps, _ = Expectation(pc, rc, groups);
println("Female : \$ $(exps[2])" );
println("Male : \$ $(exps[1])" );
println("Diff : \$ $(exps[2] - exps[1])" );
Female : $ 14170.125469335406
Male : $ 13196.548926381849
Diff : $ 973.5765429535568
Model-Based Algorithmic Fairness: FairPC

Learn classifier given
- features S and X
- training labels D

Group fairness by demographic parity:
*Fair decision $D_f$ should be independent of the sensitive attribute S*

Discover the latent fair decision $D_f$ by learning a PC.

[Choi et al. Arxiv20]
Probabilistic Sufficient Explanations

Goal: explain an instance of classification (a specific prediction)
Explanation is a subset of features, s.t.
1. The explanation is “probabilistically sufficient”
   Under the feature distribution, given the explanation, the classifier is likely to make the observed prediction.
2. It is minimal and “simple”
“Pure learning is brittle”

- bias, algorithmic fairness, interpretability, explainability, adversarial attacks,
- unknown unknowns, calibration, verification, missing features, missing labels,
- data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world.
Probabilistic Programs
What are probabilistic programs?

```
let x = flip 0.5 in
let y = flip 0.7 in
let z = x || y in
let w = if z then
  my_func(x,y)
else
  ...
in
observe(z);
```

- means “flip a coin, and output true with probability \( \frac{1}{2} \)”
- Standard (functional) programming constructs: let, if, ...
- means “reject this execution if z is not true”
Why Probabilistic Programming?

PPLs are proliferating

Pyro, Edward, HackPPL, Stan, Figaro

Venture, Church, IBAL, WebPPL, Infer.NET, Tensorflow Probability, ProbLog, PRISM, LPADs, CPL logic, CLP(BN), ICL, PHA, Primula, Storm, Gen, PRISM, PSI, Bean Machine, etc. … and many many more

Programming languages are humanity’s biggest knowledge representation achievement!
**Dice** probabilistic programming language

http://dicelang.cs.ucla.edu/

https://github.com/SHoltzen/dice

---

**dice** is a probabilistic programming language focused on fast exact inference for discrete probabilistic programs. For more information on **dice**, see the about page.

Below is an online **dice** code demo. To run the example code, press the "Run" button.

```plaintext
let key = discrete(0.25, 0.25, 0.25) in
let enc = key + gen in
// encrypt the character

let key = discrete(0.25, 0.25, 0.25) in
let gen = discrete(0.75, 0.25) in

// observe the ciphertext CCCCC
```

[Holtzen et al. OOPSLA20]
Possible world = Execution path

<table>
<thead>
<tr>
<th>Execution A</th>
<th>Execution B</th>
<th>Execution C</th>
<th>Execution D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=1</td>
<td>x=1</td>
<td>x=0</td>
<td>x=0</td>
</tr>
<tr>
<td>x=1, y=1</td>
<td>x=1, y=0</td>
<td>x=0, y=1</td>
<td>x=0, y=0</td>
</tr>
<tr>
<td>x=1, y=1, z=1</td>
<td>x=1, y=0, z=1</td>
<td>x=0, y=1, z=1</td>
<td>x=0, y=0, z=0</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>(1,0)</td>
<td>(0,1)</td>
<td>(1,0)</td>
</tr>
</tbody>
</table>

P = 0.4*0.7  
P = 0.4*0.3  
P = 0.6*0.7  
P = 0.6*0.3

let x = flip 0.4 in
let y = flip 0.7 in
let z = x || y in
let x = if z then
   x
else
   1
in (x,y)
Why should I care?

Better abstraction than probabilistic graphical models:

• Beyond variable-level dependencies (contextual)
• modularity through functions reuse (cf. relational graphical models)
• intuitive language for local structure; arithmetic
• data structures
• first-class observations
Probabilistic Program Inference

Path enumeration: find all of them!

---

- **Dice**
- **Dice (Inline)**
- **Psi**
- **Psi DP**
- **WebPPL Exact**
- **Rejection**

---

- **Time (ms)**
  - Log scale: $10^1, 10^2, 10^3, 10^4, 10^5$

- **# Characters**

- **Length**

- **Length**

- **Length**
Probabilistic Program Inference

Compact representation of paths in algebraic decision diagrams
[Storm, PRISM, Claret et al.]

Much better when small number of distinct probabilities!

- Great when there’s parameter symmetry (lifted inference)
- Scales exponentially with distinct parameters

Example: *N coin flips with slightly different bias*
Probabilistic Program Inference

Path-based methods miss key ingredient: factorization .... aka the product nodes

```
1  let x = flip\textsubscript{1} 0.1 in
2  let y = if x then flip\textsubscript{2} 0.2 else
3    flip\textsubscript{3} 0.3 in
4  let z = if y then flip\textsubscript{4} 0.4 else
5    flip\textsubscript{5} 0.5 in z
```
Symbolic Compilation in Dice

- Construct Boolean formula
- Satisfying assignments \( \approx \) paths
- Variables are flips
- Associate weights with flips
- Compile factorized circuit

\[
\begin{align*}
0.1 \cdot & \begin{cases} x=T \end{cases} + 0.2 \cdot & \begin{cases} y=T \end{cases} + 0.4 \cdot & \begin{cases} z=T \end{cases} + 0.1 \cdot & \begin{cases} x=T \end{cases} + 0.8 \cdot & \begin{cases} y=F \end{cases} + 0.5 \cdot & \begin{cases} z=T \end{cases} + 0.9 \cdot & \begin{cases} x=F \end{cases} + 0.3 \cdot & \begin{cases} y=T \end{cases} + 0.4 \cdot & \begin{cases} z=T \end{cases} + 0.9 \cdot & \begin{cases} x=F \end{cases} + 0.7 \cdot & \begin{cases} y=F \end{cases} + 0.5 \cdot & \begin{cases} z=T \end{cases} \\
\end{align*}
\]

\( f_1 f_2 f_4 \lor \bar{f}_1 \bar{f}_2 f_5 \lor \bar{f}_1 f_3 f_4 \lor \bar{f}_1 \bar{f}_3 f_5 \)

```
1  let x = flip1 0.1 in
2  let y = if x then flip2 0.2 else
3      flip3 0.3 in
4  let z = if y then flip4 0.4 else
5      flip5 0.5 in z
```
Symbolic Compilation to Probabilistic Circuits

- Probabilistic Program
- Symbolic Compilation
  - Retains Program Structure
- Weighted Boolean Formula
- Circuit compilation
- Logic Circuit (BDD)
- WMC
- Probabilistic Circuit
Factorized Inference in Dice

Network Verification
First-Class Observations

```kotlin
fun EncryptChar(key:int, obs:char):Bool {
    let randomChar = ChooseChar() in
    let ciphertext = (randomChar + key) % 26 in
    let _ = observe ciphertext = obs in
    true
}
let k = UniformInt(0, 25) in
let _ = EncryptChar(k, 'H') in ...
let _ = EncryptChar(k, 'D') in k
```

Frequency Analyzer for a Caesar cipher in Dice
PPL benchmarks from PL community

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>167</td>
<td>57</td>
<td>1.0</td>
<td>$9.5 \times 10^1$</td>
<td>15</td>
</tr>
<tr>
<td>Burglar Alarm</td>
<td>98</td>
<td>10</td>
<td>1.1</td>
<td>$2.5 \times 10^2$</td>
<td>11</td>
</tr>
<tr>
<td>Coin Bias</td>
<td>94</td>
<td>23</td>
<td>1.0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Noisy Or</td>
<td>81</td>
<td>152</td>
<td>1.0</td>
<td>$1.6 \times 10^4$</td>
<td>35</td>
</tr>
<tr>
<td>Evidence1</td>
<td>48</td>
<td>32</td>
<td>1.0</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Evidence2</td>
<td>59</td>
<td>28</td>
<td>1.0</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Murder Mystery</td>
<td>193</td>
<td>75</td>
<td>1.0</td>
<td>$1.6 \times 10^1$</td>
<td>6</td>
</tr>
</tbody>
</table>
## Scalable Inference

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Parameters</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer [48]</td>
<td>772</td>
<td>46</td>
<td>1.0</td>
<td>10</td>
<td>1.1×10³</td>
<td>28</td>
</tr>
<tr>
<td>Survey [73]</td>
<td>2477</td>
<td>152</td>
<td>2.0</td>
<td>21</td>
<td>1.3×10⁴</td>
<td>73</td>
</tr>
<tr>
<td>Alarm [5]</td>
<td>X</td>
<td>X</td>
<td>9.0</td>
<td>509</td>
<td>1.0×10³⁶</td>
<td>1.3×10³</td>
</tr>
<tr>
<td>Insurance [7]</td>
<td>X</td>
<td>X</td>
<td>75.0</td>
<td>984</td>
<td>1.2×10⁴⁰</td>
<td>1.0×10⁵</td>
</tr>
<tr>
<td>Hepar2 [63]</td>
<td>X</td>
<td>X</td>
<td>54.0</td>
<td>1453</td>
<td>2.9×10⁶⁹</td>
<td>1.3×10³</td>
</tr>
<tr>
<td>Hailfinder [1]</td>
<td>X</td>
<td>X</td>
<td>526.0</td>
<td>2656</td>
<td>2.0×10⁷⁶</td>
<td>6.5×10⁴</td>
</tr>
<tr>
<td>Pigs</td>
<td>X</td>
<td>X</td>
<td>32.0</td>
<td>5618</td>
<td>7.3×10⁴⁹²</td>
<td>35</td>
</tr>
<tr>
<td>Water [43]</td>
<td>X</td>
<td>X</td>
<td>2926.0</td>
<td>1.0×10⁴</td>
<td>3.2×10⁵⁴</td>
<td>5.1×10⁴</td>
</tr>
<tr>
<td>Munin [3]</td>
<td>X</td>
<td>X</td>
<td>1945.0</td>
<td>8.1×10⁵</td>
<td>2.1×10¹⁶²²</td>
<td>1.1×10⁴</td>
</tr>
</tbody>
</table>
import PL.* – Denotational Semantics

- **Goal**: associate with every expression “e” a semantic object.
- **Notation**: semantic bracket: [[.]]
  - In Bayesian network: [[BN]] = Pr_{BN}(.)
  - In probabilistic programs: [[e]](.) for all expressions
  - Accepting and distributional semantics:
    \[
    [e]_A \triangleq \sum_v [e](v), \quad [e]_D (v) \triangleq \frac{1}{[e]_A} [e](v)
    \]
- **Idea**: don’t need to run ‘flip 0.4’ infinite times to know meaning
Denotational Semantics + Formal Inference Rules

\[
\begin{align*}
[v_1](v) & \triangleq (\delta(v_1))(v) \\
[f \text{st} \ (v_1, v_2)](v) & \triangleq (\delta(v_1))(v) \\
[snd \ (v_1, v_2)](v) & \triangleq (\delta(v_2))(v) \\
[\text{if } v_0 \text{ then } e_1 \text{ else } e_2](v) & \triangleq \\
& \begin{cases} 
[e_1](v) & \text{if } v_0 = T \\
[e_2](v) & \text{if } v_0 = F \\
0 & \text{otherwise}
\end{cases} \\
[\text{flip } \theta](v) & \triangleq \\
& \begin{cases} 
\theta & \text{if } v = T \\
1 - \theta & \text{if } v = F \\
0 & \text{otherwise}
\end{cases} \\
[\text{observe } v_1](v) & \triangleq \\
& \begin{cases} 
1 & \text{if } v_1 = T \text{ and } v = T, \\
0 & \text{otherwise}
\end{cases} \\
[f(v_1)](v) & \triangleq (T(f))(v_1)
\end{align*}
\]

[let \( x = e_1 \) in \( e_2 \)](v) \( \triangleq \sum_{v'} [e_1](v') \times [e_2[x \mapsto v']](v) \)
Provably Correct Inference!

\[
\text{flip } 0.1 \rightsquigarrow (\text{f}_{1}, \text{true}, w_1)
\]
\[
x \rightsquigarrow (\text{true}, \text{false}, \emptyset)
\]
\[
\text{flip } 0.4 \rightsquigarrow (\text{f}_{2}, \text{true}, w_2)
\]
\[
\text{let } x = \text{flip } 0.1 \text{ in } \text{flip } 0.4 \lor x \rightsquigarrow (\text{f}_{2}, \text{true}, w_1 \cup w_2)
\]
Better Inference?

Exploit modularity

1. **AI modularity:**
   Discover contextual independencies and **factorize**

2. **PL modularity:**
   Compile procedure summaries and reuse at each call site

Reason about programs! Compiler optimizations.
Quick preview:

3. Flip hoisting optimization
4. Eager compilation
## Compiler Optimizations (sneak preview)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Naive compilation</th>
<th>determinism</th>
<th>flip hoisting + determinism</th>
<th>Eager + flip lifting</th>
<th>Ace baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>156</td>
<td>140</td>
<td>83</td>
<td>69</td>
<td>422</td>
</tr>
<tr>
<td>water</td>
<td>56,267</td>
<td>65,975</td>
<td>1509</td>
<td>941</td>
<td>605</td>
</tr>
<tr>
<td>insurance</td>
<td>140</td>
<td>100</td>
<td>100</td>
<td>128</td>
<td>492</td>
</tr>
<tr>
<td>hepar2</td>
<td>95</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>495</td>
</tr>
<tr>
<td>pigs</td>
<td>3,772</td>
<td>2490</td>
<td>2112</td>
<td>186</td>
<td>985</td>
</tr>
<tr>
<td>munin</td>
<td>&gt;1,000,000</td>
<td>&gt;1,000,000</td>
<td>109,687</td>
<td>16,536</td>
<td>3,500</td>
</tr>
</tbody>
</table>

Inference time in milliseconds
Conclusions

- Are we already in the age of computational abstractions?
- **Probabilistic circuits** for learning deep tractable probabilistic models
- **Probabilistic programs** as the new probabilistic knowledge representation language
Thanks