UCLA Computer Science

STAR AI RESEARCH LAB UCLA

Recent Advances in Discrete Probabilistic Program Inference

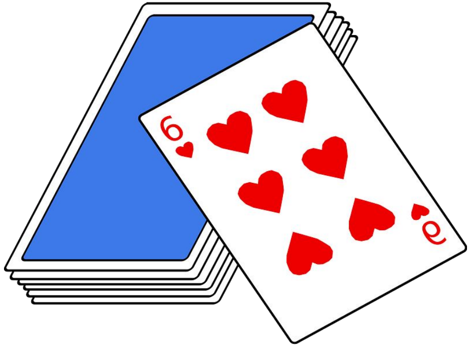
Guy Van den Broeck

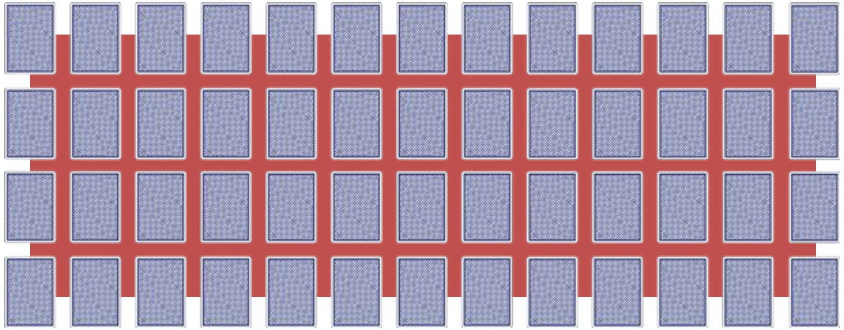
VeriProP 2021 - Jul 19, 2021

What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

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





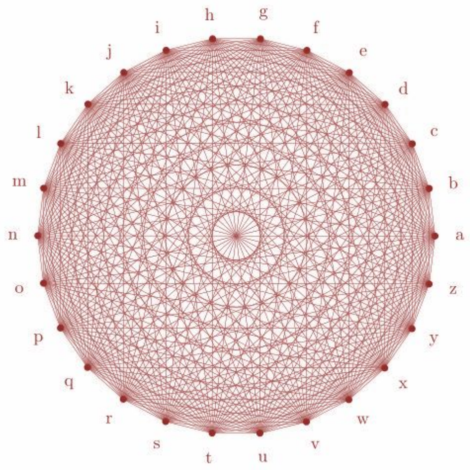
[VdB KRR15]

What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

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

3.14 Smokes(x) ∧ Friends(x,y) ⇒ Smokes(y)



[VdB KRR15]

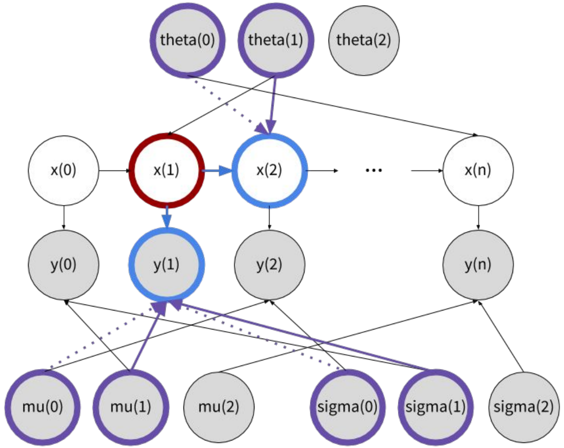
What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

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

Bean Machine

$$\mu_k \sim \mathrm{Normal}(\alpha, \beta)$$
$$\sigma_k \sim \mathrm{Gamma}(\nu, \rho)$$
$$\theta_k \sim \mathrm{Dirichlet}(\kappa)$$
$$x_i \sim \begin{cases} \mathrm{Categorical}(init) & \text{if } i = 0 \\ \mathrm{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 \end{cases}$$
$$y_i \sim \mathrm{Normal}(\mu_{x_i}, \sigma_{x_i})$$



[Tehrani et al. PGM20]

Computational Abstractions

Let us think of probability as something that is computed.

Abstraction = Structure of Computation

Two levels of abstraction:

Probabilistic Programs	"High-level code"
Probabilistic Circuits	"Machine code"

language design

compilation

program abstraction

source-to-source compilation

compiler optimization

learning/ synthesis

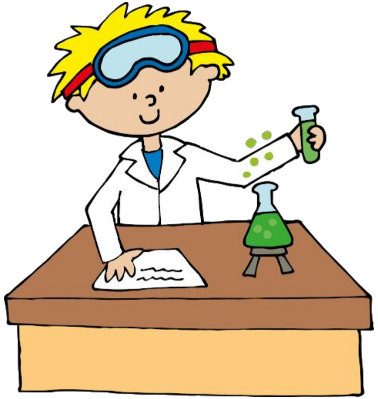
hardware mapping

...

Probabilistic Programs

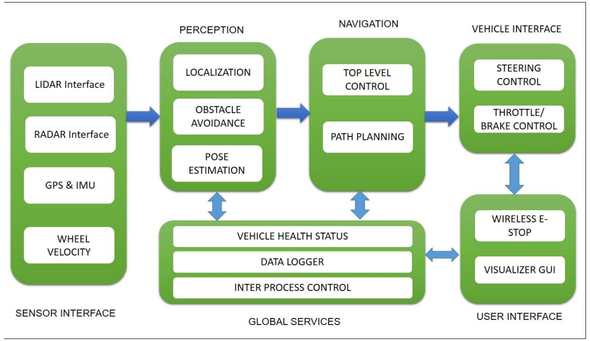
Motivation from the AI side:
Making modern AI systems is **too hard**

System Builders



Model Builders

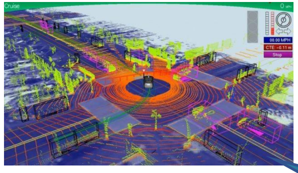
AI System Builder



Need to integrate uncertainty over the whole system

20% chance of obstacle!

94% chance of obstacle!



. . .

99% certain about current location

Inside the Self-Driving Tesla Fatal Accident

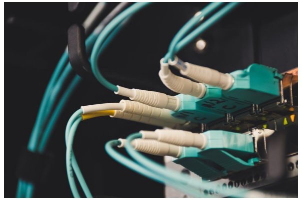
By ANJALI SINGHVI and KARL RUSSELL UPDATED July 12, 2016

The accident may have happened in part because the crash-avoidance system is designed to engage only when radar and computer vision systems agree that there is an obstacle, according to an industry executive with direct

AI Model Builder

“When you have the flu you have a cough 70% of the time”

“What is the probability that a patient with a fever has the flu?”



“Routers fail on average every 5 years”

“What is the probability that my packet will reach the target server?” [SGTVV SIGCOMM’20]

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 ½"

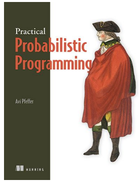
Standard (functional) programming constructs: let, if, ...

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, CPLogic, 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!
- Programs should be AI models

Focus on Discrete Models

1. Real programs have inherently discrete structure (e.g. if-statements)
2. Discrete structure is inherent in many domains (graphs, text, ranking, etc.)
3. Many existing PPLs assume smooth and differentiable densities and do not handle discreteness well.

Stan

Does not support if-statements!

WebPPL

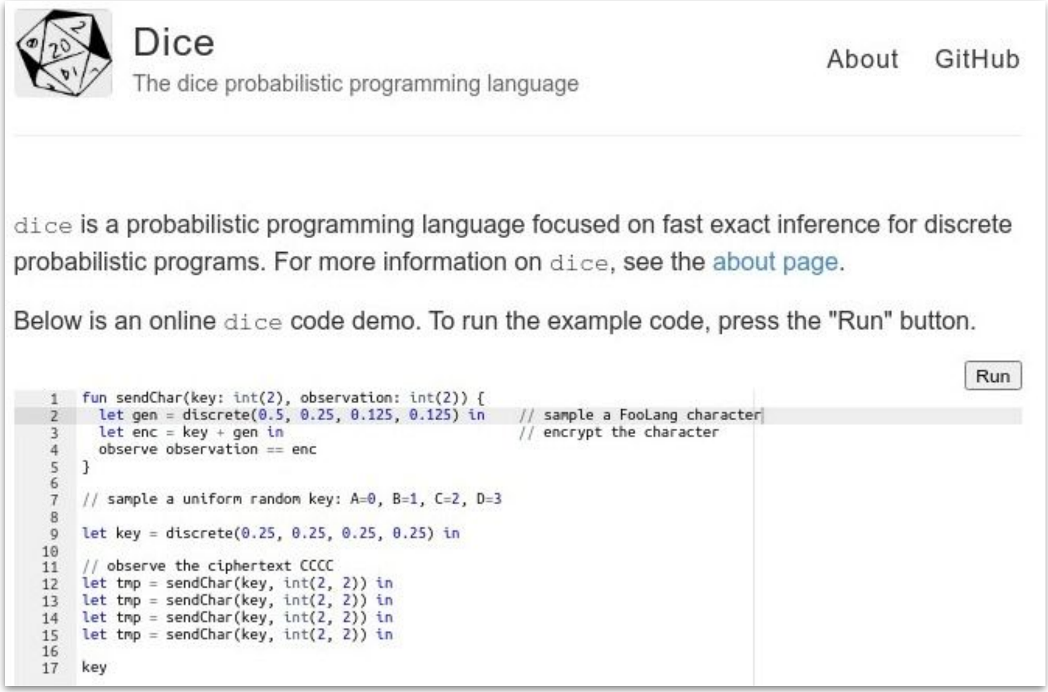
coroutines. Whenever a discrete variable is encountered in a program's execution, the program is suspended and resumed multiple times with all possible values in the support of that distribution. Listing 10, which impl[...]nts a simple finite

[AADB+'19]

Discrete probabilistic programming is the important unsolved open problem!

Dice language for discrete probabilistic programs

http://dicelang.cs.ucla.edu/ [Holtzen et al. OOPSLA20]



Dice

The dice probabilistic programming language

About GitHub

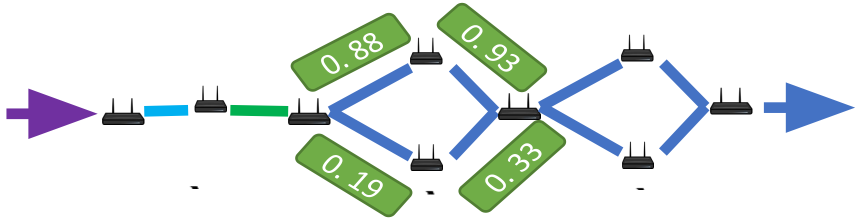
`dice` is a probabilistic programming language focused on fast exact inference for discrete probabilistic programs. For more information on `dice`, see the about page.

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

Run

```
fun sendChar(key: int(2), observation: int(2)) {
  let gen = discrete(0.5, 0.25, 0.125, 0.125) in    // sample a FooLang character
  let enc = key + gen in                            // encrypt the character
  observe observation == enc
}

// sample a uniform random key: A=0, B=1, C=2, D=3

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

// observe the ciphertext CCCC
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in
let tmp = sendChar(key, int(2, 2)) in

key
```

Network Verification in Dice




```
fun n1(init: bool) {
   let l1succeed = flip 0.99 in
   let l2succeed = flip 0.91 in
   init && l1succeed && l2succeed
}
```

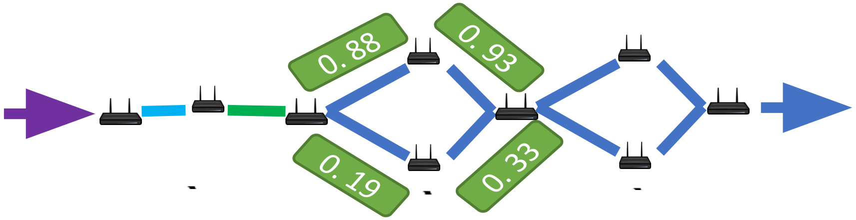
```
fun n2(init: bool) {
   let routeChoice = flip 0.5 in
   if routeChoice then
      init && flip 0.88 && flip 0.93
   else
      init && flip 0.19 && flip 0.33
}
```

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks

```
n2(n2(n1(true)))
```

Network Verification i




```
fun n1(
  let l1
  let l2
  init &
}
```

This doesn't show all the language features of dice:

- Integers
- Tuples
- Bounded recursion
- Bayesian conditioning
- …

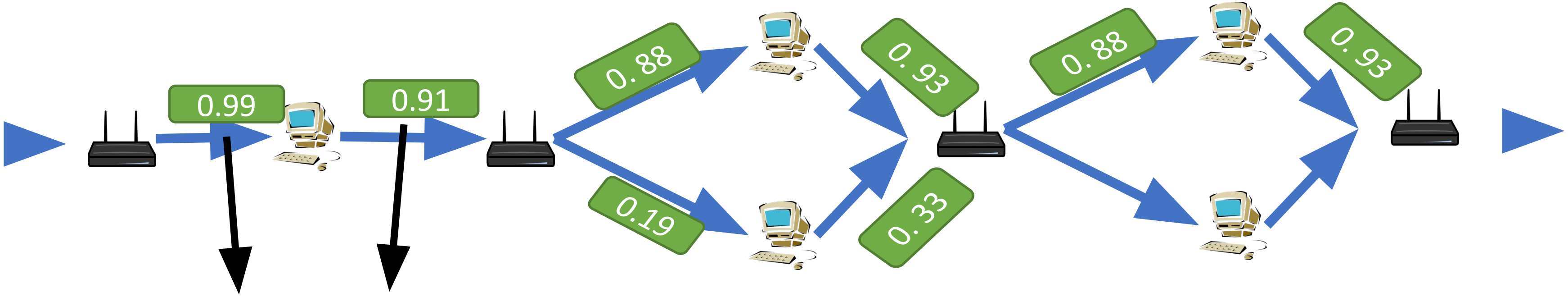
```
fun n2(init: bool) {
  let routeChoice = flip 0.5 in
  if routeChoice then
    init && flip 0.88 && flip 0.93
  else
    init && flip 0.19 && flip 0.33
}
```

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks

```
n2(n2(n1(true)))
```

Probabilistic Program Inference



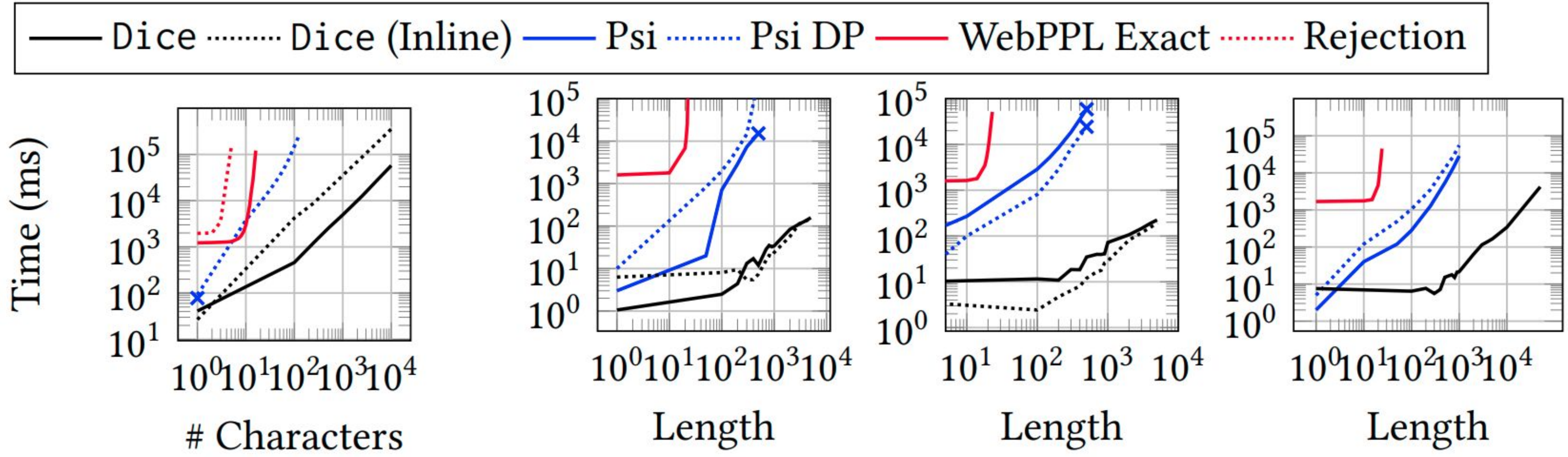
0.99 x 0.91 x 0.5 x 0.88 x 0.93 x 0.5 x 0.88 x 0.93

\+ 0.99 x 0.91 x 0.5 x 0.19 x 0.33 x 0.5 x 0.88 x 0.93

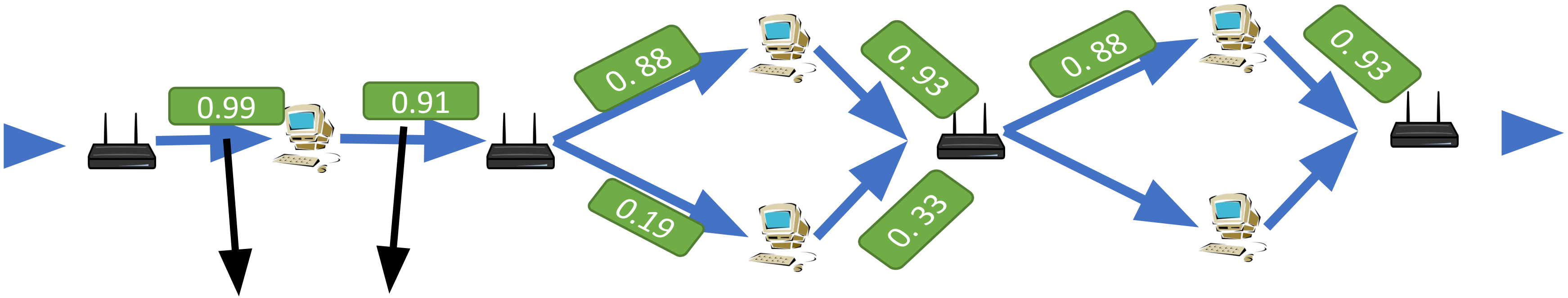
\+ ...

Probabilistic Program Inference

Path enumeration: find all of them!



Key to Fast Inference: **Factorization** (product nodes)



$$
\begin{aligned}
&\boxed{0.99 \times 0.91} \times 0.5 \times 0.88 \times 0.93 \times 0.5 \times 0.88 \times 0.93 \\
+\ &0.99 \times 0.91 \times 0.5 \times 0.19 \times 0.33 \times 0.5 \times 0.88 \times 0.93 \\
+\ &\ldots
\end{aligned}
$$

Easy to see on the graph structure …
how about on the program?

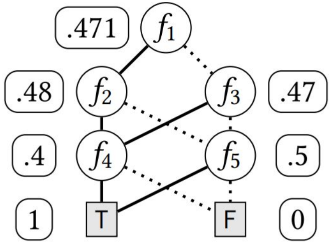
Symbolic Compilation in Dice

- Construct Boolean formula
- Satisfying assignments ≈ paths
- Variables are flips
- Associate weights with flips
- Compile factorized circuit

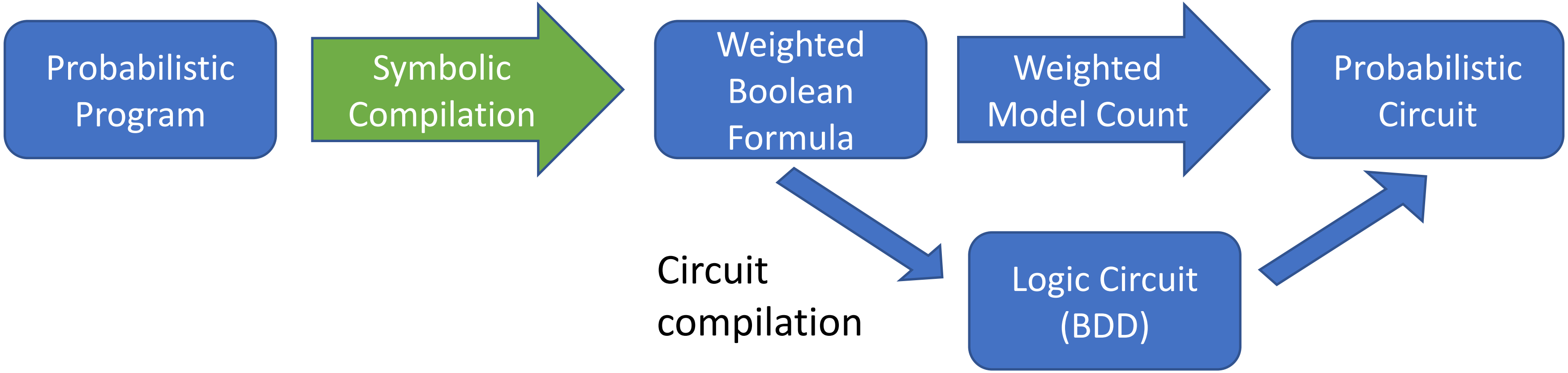
```
let x = flip₁ 0.1 in
let y = if x then flip₂ 0.2 else
    flip₃ 0.3 in
let z = if y then flip₄ 0.4 else
    flip₅ 0.5 in z
```

$$\underbrace{0.1}_{x=T} \cdot \underbrace{0.2}_{y=T} \cdot \underbrace{0.4}_{z=T} + \underbrace{0.1}_{x=T} \cdot \underbrace{0.8}_{y=F} \cdot \underbrace{0.5}_{z=T} + \underbrace{0.9}_{x=F} \cdot \underbrace{0.3}_{y=T} \cdot \underbrace{0.4}_{z=T} + \underbrace{0.9}_{x=F} \cdot \underbrace{0.7}_{y=F} \cdot \underbrace{0.5}_{z=T}$$

$$\Rightarrow f_1 f_2 f_4 \vee f_1 \bar{f}_2 f_5 \vee \bar{f}_1 f_3 f_4 \vee \bar{f}_1 \bar{f}_3 f_5 \Rightarrow$$



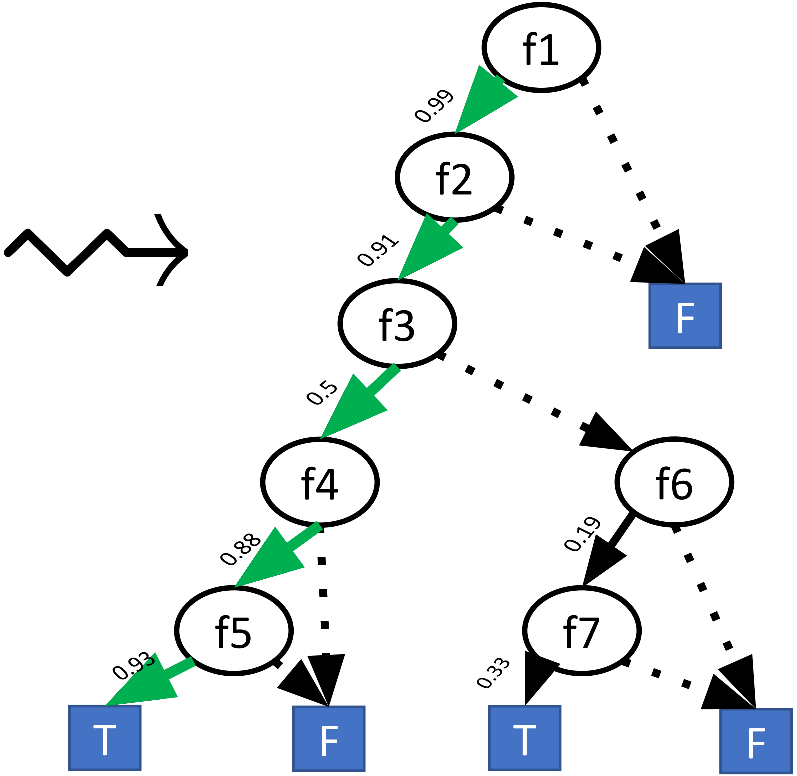
Symbolic Compilation in Dice



An *Equivalent* BDD to this Program

```
fun n1(init: bool) {
  let l1succeed = True
  let l2succeed = True
  init && l1succeed && l2succeed
}
```

```
fun n2(init: bool) {
  let rC = True
  if rC then
    init && True True
  else
    init && flip6 0.19 && flip7 0.33
}
```

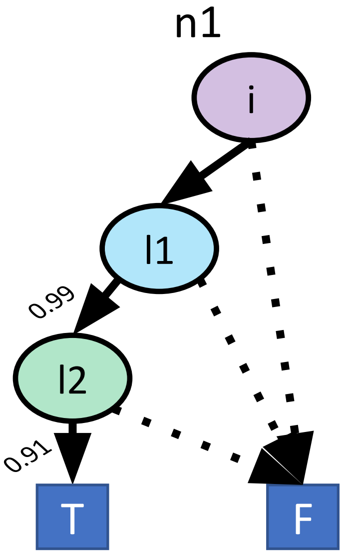


Now, how do we compile this?

Compiling the BDD Modularly

```
fun n1(init: bool) {
  let l1succeed = flip 0.99 in
  let l2succeed = flip 0.91 in
  init && l1succeed && l2succeed
}
```

$\rightsquigarrow$

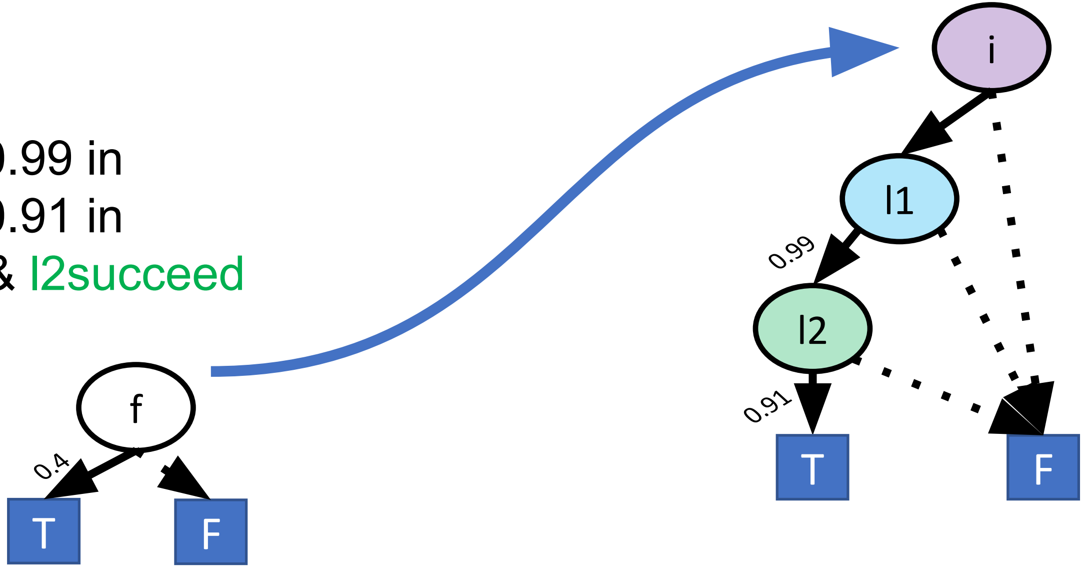


First, compile the function n1

Compiling the BDD Modularly

```
fun n1(init: bool) {
   let l1succeed = flip 0.99 in
   let l2succeed = flip 0.91 in
   init && l1succeed && l2succeed
}
n1(flip 0.4)
```

flip 0.4 $\rightsquigarrow$

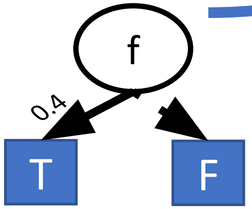


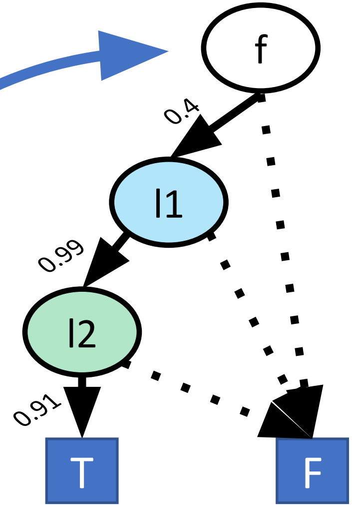
Then, to *call* n1, substitute for i

Compiling the BDD Modularly

```
fun n1(init: bool) {
  let l1succeed = flip 0.99 in
  let l2succeed = flip 0.91 in
  init && l1succeed && l2succeed
}
n1(flip 0.4)
```

flip 0.4 $\rightsquigarrow$

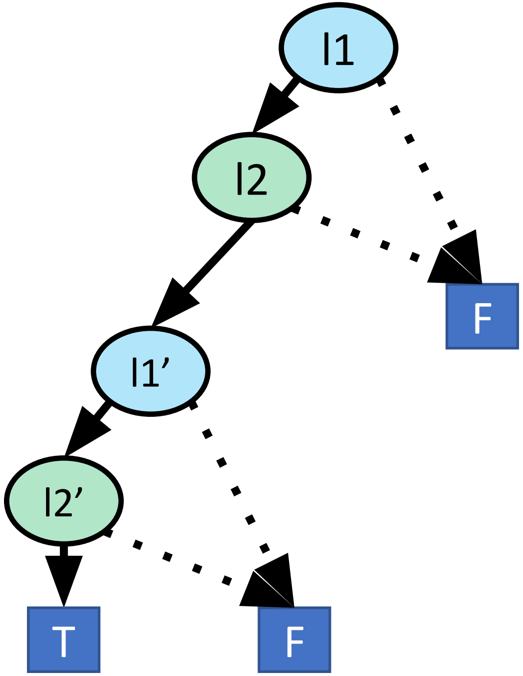




Then, to *call* n1, substitute for i

Compiling the BDD Modularly

```
fun n1(init: bool) {
   let l1succeed = flip 0.99 in
   let l2succeed = flip 0.91 in
   init && l1succeed && l2succeed
}
n1(n1(true))
```



- Calling itself? Size (and therefore inference cost) grows *linearly*
- Build BDD for whole program by combining sub-programs **modularly**

Denotational Semantics
+ Formal Inference Rules

$$[\![v_1]\!](v) \triangleq \big(\delta(v_1)\big)(v) \qquad [\![\texttt{fst}\ (v_1, v_2)]\!](v) \triangleq \big(\delta(v_1)\big)(v) \qquad [\![\texttt{snd}\ (v_1, v_2)]\!](v) \triangleq \big(\delta(v_2)\big)(v)$$

$$[\![\texttt{if}\ v_g\ \texttt{then}\ \mathsf{e}_1\ \texttt{else}\ \mathsf{e}_2]\!](v) \triangleq \begin{cases} [\![\mathsf{e}_1]\!](v) & \text{if } v_g = \mathsf{T} \\ [\![\mathsf{e}_2]\!](v) & \text{if } v_g = \mathsf{F} \\ 0 & \text{otherwise} \end{cases} \qquad [\![\texttt{flip}\ \theta]\!](v) \triangleq \begin{cases} \theta & \text{if } v = \mathsf{T} \\ 1-\theta & \text{if } v = \mathsf{F} \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\texttt{observe}\ v_1]\!](v) \triangleq \begin{cases} 1 & \text{if } v_1 = \mathsf{T} \text{ and } v = \mathsf{T}, \\ 0 & \text{otherwise} \end{cases} \qquad [\![f(v_1)]\!](v) \triangleq \Big(\big(T(f)\big)(v_1)\Big)(v)$$

$$[\![\texttt{let}\ x = \mathsf{e}_1\ \texttt{in}\ \mathsf{e}_2]\!](v) \triangleq \sum_{v'} [\![\mathsf{e}_1]\!](v') \times [\![\mathsf{e}_2[x \mapsto v']]\!](v)$$

$$\frac{}{\mathsf{T} \rightsquigarrow (\mathsf{T}, \mathsf{T}, \emptyset)}\ (\text{C-True}) \qquad \frac{}{\mathsf{F} \rightsquigarrow (\mathsf{F}, \mathsf{T}, \emptyset)}\ (\text{C-False}) \qquad \frac{}{x \rightsquigarrow (\mathbf{x}, \mathsf{T}, \emptyset)}\ (\text{C-Ident})$$

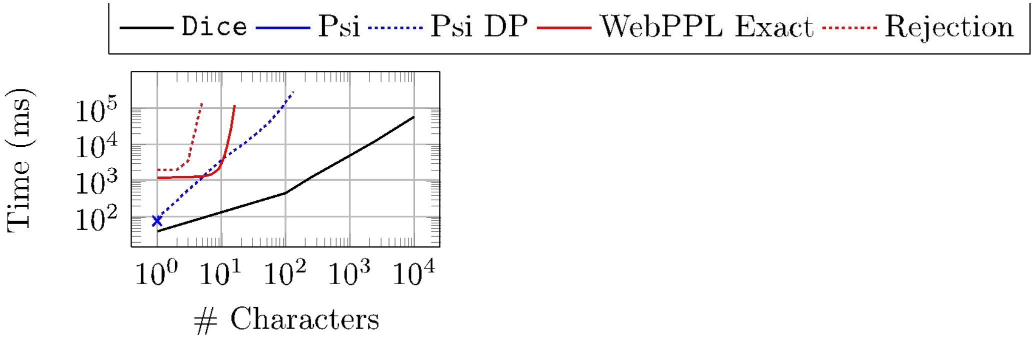
$$\frac{\text{fresh } \mathbf{f}}{\texttt{flip}\ \theta \rightsquigarrow \big(\mathbf{f}, \mathsf{T}, (\mathbf{f} \mapsto \theta, \mathsf{T}, \overline{\mathbf{f}} \mapsto 1-\theta)\big)}\ (\text{C-Flip}) \qquad \frac{\mathsf{aexp} \rightsquigarrow (\varphi, \mathsf{T}, \emptyset)}{\texttt{observe}\ \mathsf{aexp} \rightsquigarrow (\mathsf{T}, \varphi, \emptyset)}\ (\text{C-Obs})$$

$$\frac{\mathsf{aexp} \rightsquigarrow (\varphi_g, \mathsf{T}, \emptyset) \qquad \mathsf{e}_T \rightsquigarrow (\varphi_T, \gamma_T, w_T) \qquad \mathsf{e}_E \rightsquigarrow (\varphi_E, \gamma_E, w_E)}{\texttt{if}\ \mathsf{aexp}\ \texttt{then}\ \mathsf{e}_\mathsf{T}\ \texttt{else}\ \mathsf{e}_\mathsf{E} \rightsquigarrow \big((\varphi_g \wedge \varphi_T) \vee (\overline{\varphi}_g \wedge \varphi_E), (\varphi_g \wedge \gamma_T) \vee (\overline{\varphi}_g \wedge \gamma_E), w_T \cup w_E\big)}\ (\text{C-Ite})$$

$$\frac{\mathsf{e}_1 \rightsquigarrow (\varphi_1, \gamma_1, w_1) \qquad \mathsf{e}_2 \rightsquigarrow (\varphi_2, \gamma_2, w_2)}{\texttt{let}\ x = \mathsf{e}_1\ \texttt{in}\ \mathsf{e}_2 \rightsquigarrow \big(\varphi_2[\mathbf{x} \mapsto \varphi_1], \gamma_1 \wedge \gamma_2[\mathbf{x} \mapsto \varphi_1], w_1 \cup w_2\big)}\ (\text{C-Let})$$

Experimental Evaluation

- Example from text analysis: breaking a Caesar cipher

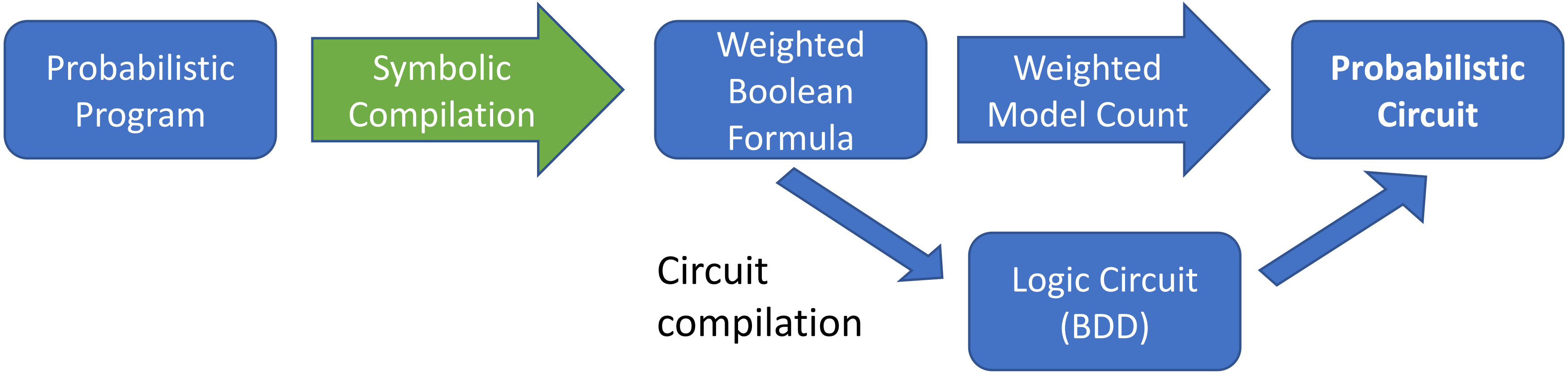


- Competitive with specialized Bayesian network solvers

More program paths than atoms in the universe

Benchmark	Psi (ms)	DP (ms)	Dice (ms)	# Parameters	# Paths	BDD Size
Cancer	772	46	**13**	10	1.1×10^3	28
Survey	2477	152	**13**	21	1.3×10^4	73
Alarm	✗	✗	**25**	509	1.0×10^{36}	1.3×10^3
Insurance	✗	✗	**212**	984	1.2×10^{40}	1.0×10^5
Hepar2	✗	✗	**54**	48	2.9×10^{69}	1.3×10^3
Hailfinder	✗	✗	**618**	2656	2.0×10^{76}	6.5×10^4
Pigs	✗	✗	**72**	5618	7.3×10^{492}	35
Water	✗	✗	**2590**	1.0×10^4	3.2×10^{54}	5.1×10^4
Munin	✗	✗	**1866**	8.1×10^5	2.1×10^{1622}	1.1×10^4

Symbolic Compilation in Dice to **Probabilistic Circuits**



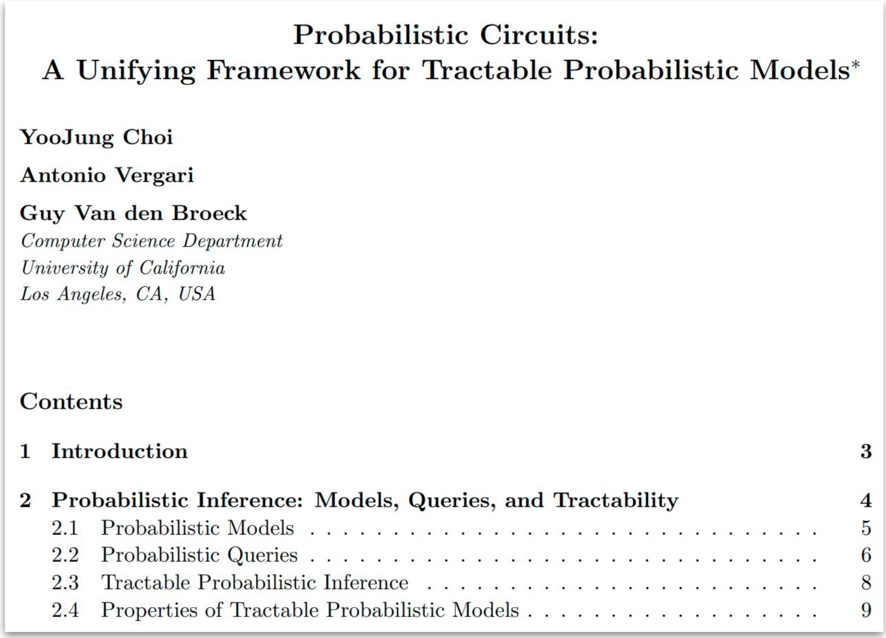
Tractable representations of probability distributions, learnable from data, mapped to GPU/hardware, with many interesting properties!

Learn more about probabilistic circuits?

Tutorial (3h)

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

Overview Paper (80p)



Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models*

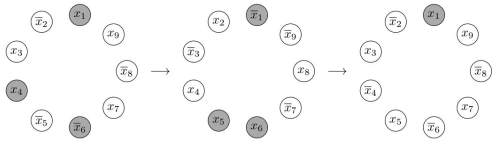
YooJung Choi
Antonio Vergari
Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

Contents

http://starai.cs.ucla.edu/papers/ProbCirc20.pdf

If you build it they will come

As soon as ***dice*** was put online people started using it in surprising ways we had not foreseen

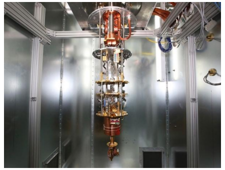


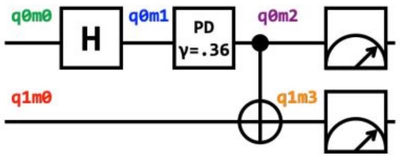
Probabilistic Model Checking

Prism

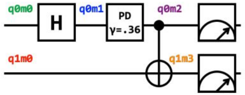
Rubicon

dice





Quantum Simulation



quantum circuit

probabilistic circuit

If you build it they will come

In both cases, ***dice*** outperforms existing specialized methods on important examples!

Probabilistic Model Checking

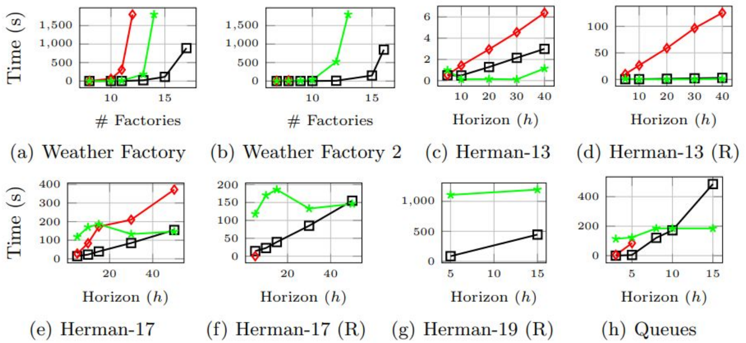
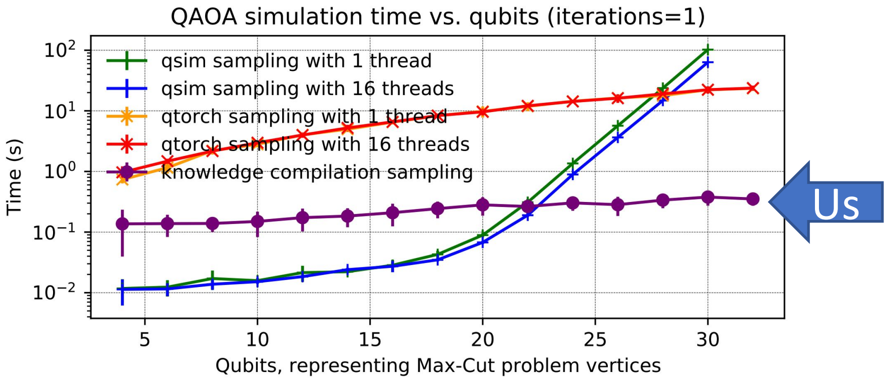


Fig. 9. Scaling plots comparing RUBICON (—□—), STORM's symbolic engine (—◇—), and STORM's explicit engine (—★—). An "(R)" in the caption denotes random parameters.

Check out CAV talk video or ask Steven Holtzen, Sebastian Junges, or Marcell Vazquez-Chanlatte

Quantum Simulation



Competitive with well-known simulators like Google qsim and qtorch [FSC+ PloS one '18] !

Better Inference. How?

Exploit modularity - program structure

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:

3. Flip hoisting optimization
4. Determinism, optimize integer representation, etc.

Flip Hoisting

```
let x = flip 0.1 in
let z = flip 0.2 in
let y = if x && z then flip 0.3
    else if x && !z then flip 0.4
    else flip 0.3
in y
```

$\equiv$

```
let x = flip 0.1 in let z = flip 0.2 in
let tmp = flip 0.3 in
let y = if x && z then tmp
    else if x && !z then flip 0.4
    else tmp
in y
```

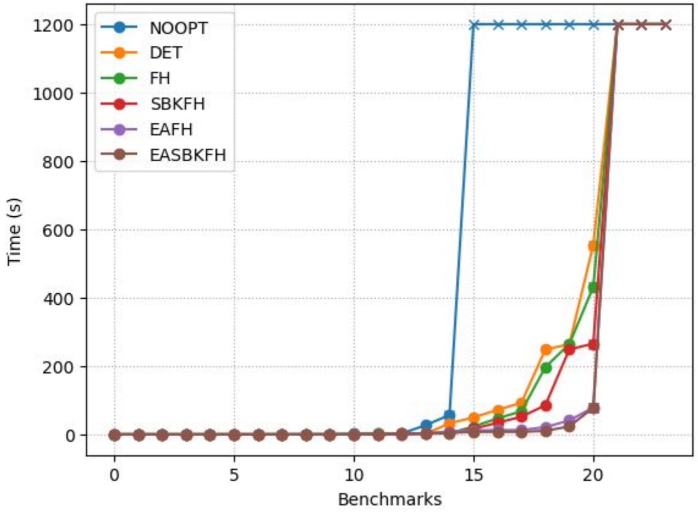
- Fewer flips = smaller compiled circuits = faster
- But, be careful with soundness:

```
flip 0.3 && flip 0.3
```

$\not\equiv$

```
let tmp = flip 0.3 in tmp && tmp;
```

Compiler Optimization Experiments



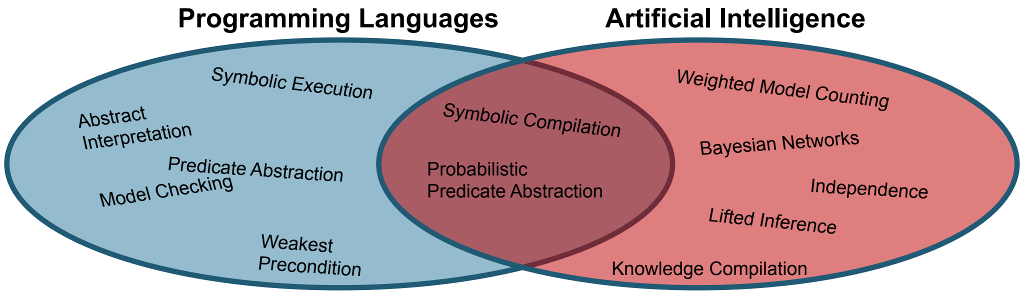
Benchmarks	No Opt	Det	FH	SBK+FH	Ea+FH	Ea+SBK+FH
ALARM	0.15	0.12	0.07	0.10	**0.06**	0.12
ANDES	56.73	32.48	3.44	**3.42**	12.57	7.47
ASIA	0.06	0.03	0.03	0.03	**0.02**	**0.02**
BARLEY	-	-	-	-	-	-
CANCER	0.04	0.03	0.03	0.03	**0.02**	**0.02**
CHILD	0.05	0.04	0.04	**0.03**	**0.03**	**0.03**
DIABETES	-	-	-	-	-	-
EARTHQUAKE	0.03	0.03	0.03	0.03	**0.02**	**0.02**
HAILFINDER	1.67	0.45	0.53	0.49	0.49	**0.41**
HEPAR2	0.13	**0.07**	**0.07**	0.12	0.08	0.10
INSURANCE	0.17	0.08	**0.07**	0.14	0.16	0.13
LINK	-	263.38	264.32	265.53	78.75	**78.10**
MILDEW	-	-	-	-	-	-
MUNIN	-	71.80	47.01	34.19	11.86	**7.52**
MUNIN1	-	49.73	22.95	16.34	8.23	**3.67**
MUNIN2	-	248.78	196.96	85.39	21.26	**10.45**
MUNIN3	-	554.62	431.62	248.68	40.87	**23.37**
MUNIN4	-	92.44	67.73	51.72	12.70	**7.66**
PATHFINDER	2.28	**1.73**	2.51	5.20	1.96	4.64
PIGS	2.33	1.87	1.58	1.54	0.20	**0.14**
SACHS	0.04	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
SURVEY	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
WATER	27.04	1.46	1.90	0.87	1.49	**0.61**
WIN95PTS	0.09	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**

Conclusions

- Are we already in the age of computational abstractions?

- Probabilistic programs as the new probabilistic knowledge representation language

- Fruitful synthesis of AI and PL/FM



Thanks

This was the work of many wonderful students & collaborators!

References: http://starai.cs.ucla.edu/publications/