Factorized Exact Inference for Discrete Probabilistic Programs

Steven Holtzen, Joe Qian, Todd Millstein, Guy Van den Broeck

UCLA

sholtzen@cs.ucla.edu, qzy@g.ucla.edu, todd@cs.ucla.edu, guyvdb@cs.ucla.edu

STAR AI RESEARCH LAB UCLA

Introduction & Motivation

- Our problem: ***exact*** probabilistic inference for ***discrete*** programs

Example program

```
x~flip(0.5);
if(x) {
  y~flip(0.4);
} else {
  y~flip(0.6);
}
```

Example inference

$$\Pr(y) = \frac{1}{2}$$

Why exact inference?

1. No error propagation
2. Core of effective approximation techniques
3. Unaffected by low-probability observations

Introduction & Motivation

- Our problem: ***exact*** probabilistic inference for ***discrete*** programs

Example program

```
x~flip(0.5);
if(x) {
  y~flip(0.4);
} else {
  y~flip(0.6);
}
```

Example inference

$$\Pr(y) = \frac{1}{2}$$

Why discrete?

1. Program constructs (e.g. `if`-statements)
2. Discrete models (graphs, topic models, …)

Existing techniques for exact inference

1. Enumerative inference

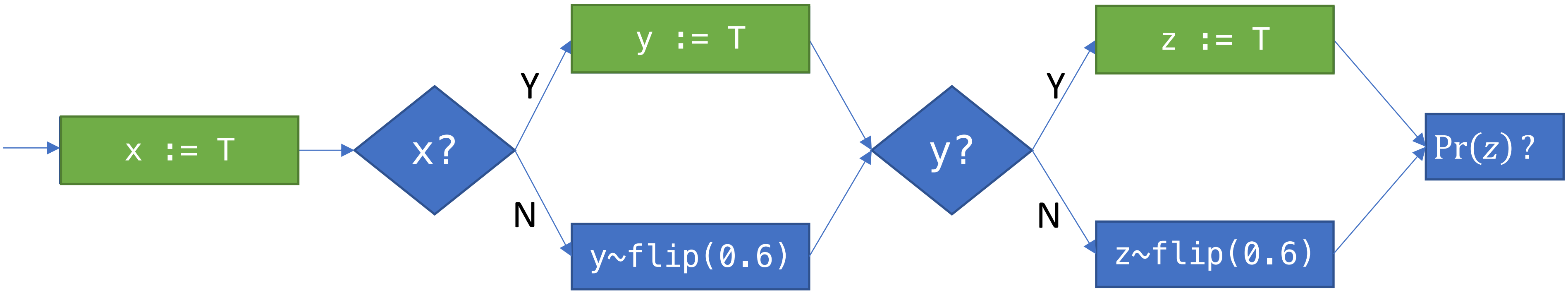
- Psi
- WebPPL
- FairSquare

2. Graphical model compilation

- Figaro
- Infer.NET
- Factorie

Enumerative inference

- Systematically explore all possible assignments to `flips` in the program

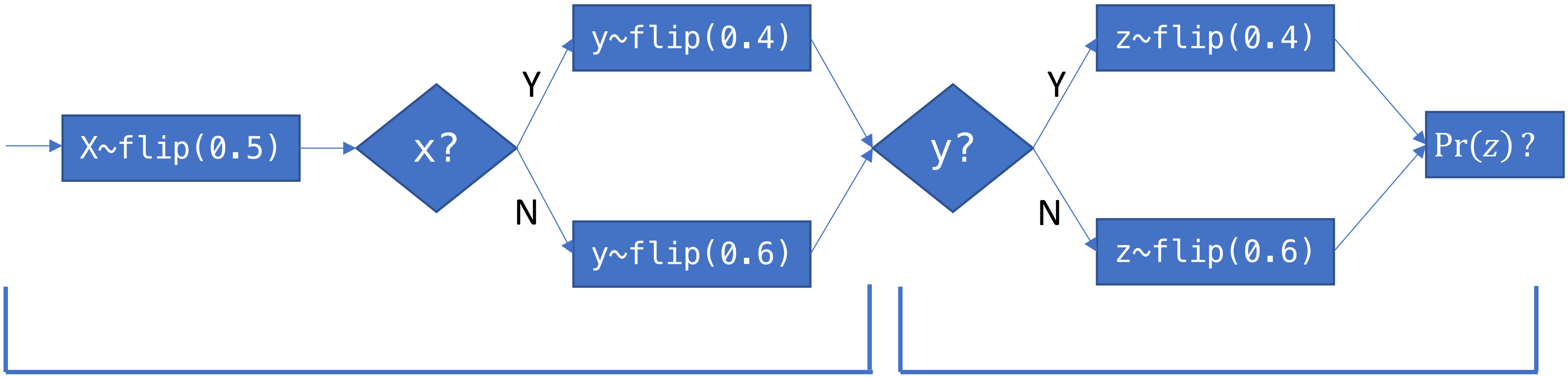


Assignment Probability:
$0.5 \times 0.4 \times 0.4$

- Scales exponentially with #`flips`

Inadequacy of enumerative inference

- Often, we can do better than enumeration

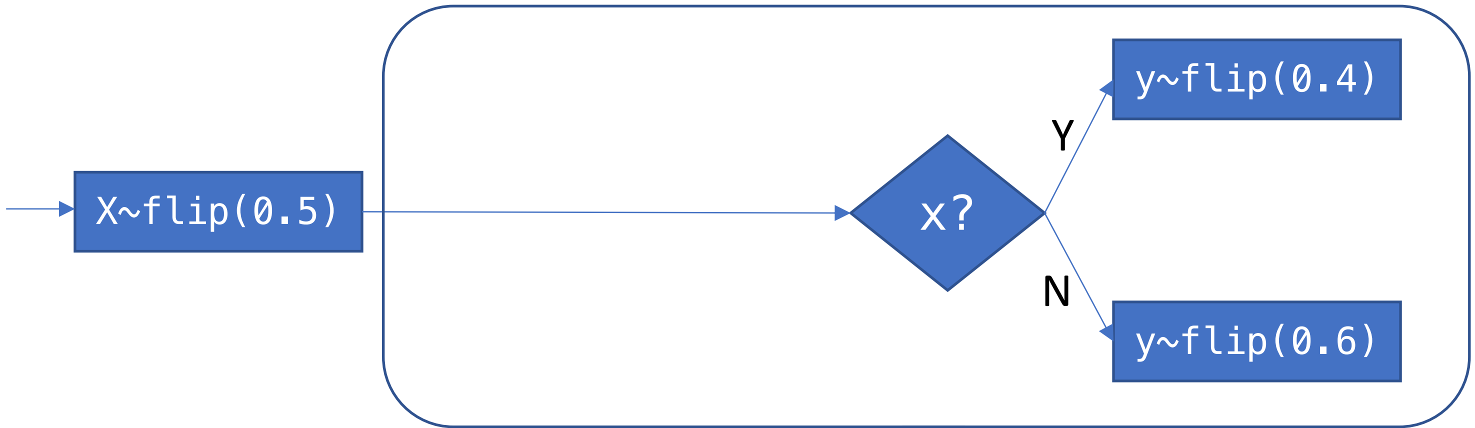


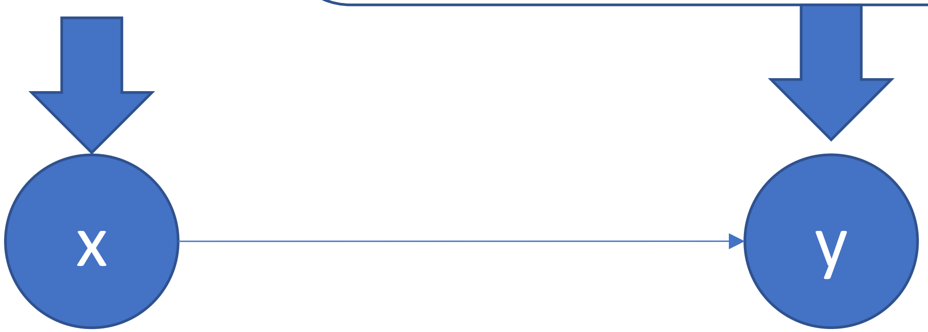
First compute $\Pr(y) = \frac{1}{2}$

Then, compute $\Pr(z)$ without looking at x

- Exploits *independence* of x and z given y
- Can we do this systematically?

Graphical model compilation



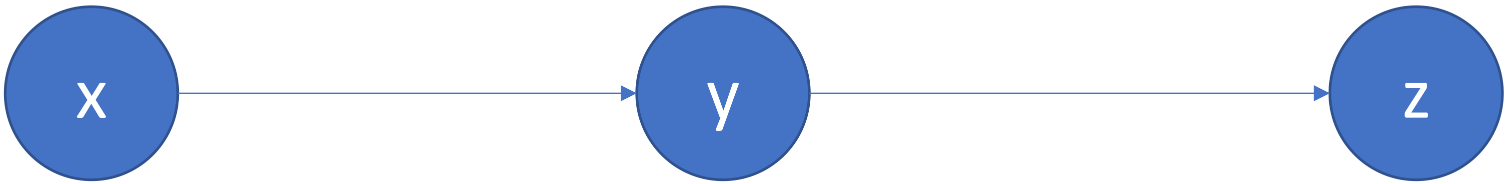


X	Pr(x)
T	0.5
F	0.5

x	y	Pr(y|x)
T	T	0.4
T	F	0.6
F	T	0.6
F	F	0.4

Graphical model compilation

- Graph makes dependencies between variables explicit



X	Pr(x)
T	0.5
F	0.5

x	y	Pr(y|x)
T	T	0.4
T	F	0.6
F	T	0.4
F	F	0.6

y	z	Pr(z|y)
T	T	0.4
T	F	0.6
F	T	0.6
F	F	0.4

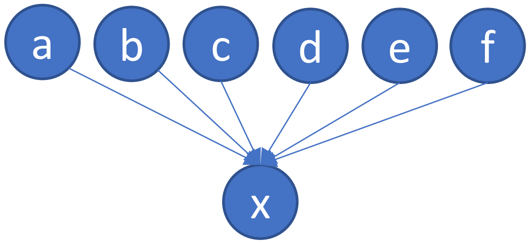
- Specialized graph-based inference methods exploit this

Coarseness of graphical models as an abstraction

- Arbitrary choice of abstraction

```
x = a || b || c || d || e || f;
```

- Tiny program, *huge conditional probability tables*



x	a	b	c	d	e	f	Pr(x|a,b,c,d,e,f)
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
1	0	1	1	1	1	1	1
…							

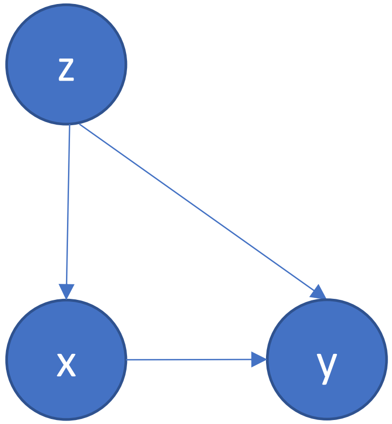
2^7 rows!

- Obfuscates useful program structure
- Easy for path-based analysis: just run the program!

Coarseness of graphical models as an abstraction

- Graph is *coarse-grained*: if a dependency *can* exist between two variables, they *must* have an edge in the graph

```
z ~flip₁(0.5);
if(z) {
  x ~flip₂(0.6);
  y ~flip₃(0.7)
} else {
  x ~flip₄(0.4);
  y := x
}
```



- Graph says there are *no independences*
  - However, program says x and y are indep. *given z = T*
  - Challenging for both graph-based and enumeration inference

Techniques for exact inference

Exploits independence to decompose inference? \ Keeps program structure?	No	Yes
Yes	**Graphical Model Compilation**	**Symbolic compilation (This work)**
No		**Enumeration**

Our contribution

- Exact inference for a Boolean-valued loop-free PPL with arbitrary observations
    - Exploits independence, is competitive with graphical model compilation
    - Retains nuanced program structure
- Give semantics for our language, prove our inference correct

Symbolic compilation

Background: Symbolic model checking

- Non-probabilistic programs can be interpreted as *logical formulae* which relate input and output states

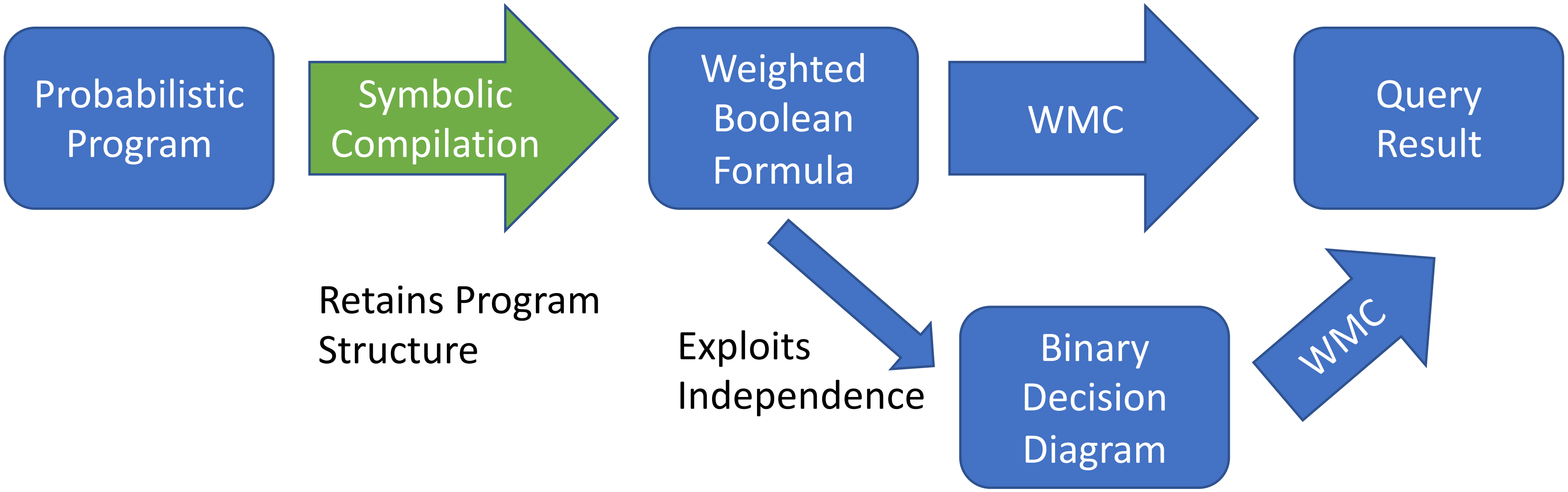

```
x := y;
```

$$\varphi = (x' \Leftrightarrow y) \land (y' \Leftrightarrow y)$$

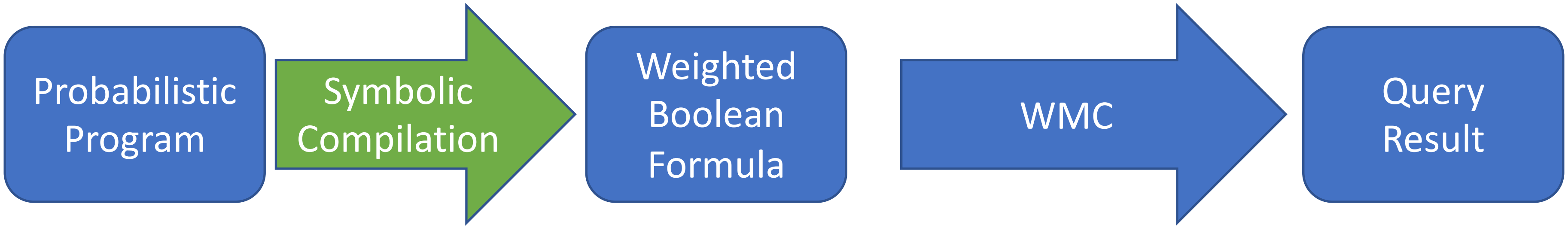
$$SAT(\varphi \land x' \land y) = T$$

$$SAT(\varphi \land x' \land \bar{y}) = \mathrm{F}$$

Inference via Weighted Model Counting



Inference via Weighted Model Counting




```
x := flip(0.5);
```

l	$w(l)$
f_1	0.4
$\overline{f_1}$	0.6

$(x' \Leftrightarrow f_1)$

$$\mathrm{WMC}(\varphi, w) = \sum_{m \models \varphi} \prod_{l \in m} w(l).$$

$\mathrm{WMC}\big((x' \Leftrightarrow f_1) \wedge x \wedge x', w\big)$?

- A single model: $\mathrm{m} = x' \wedge x \wedge f_1$
- $w(x') * w(x) * w(f_1) = 0.4$

Symbolic compilation: Flip

- Compositional process $\texttt{s} \rightsquigarrow (\varphi, w)$

$$\frac{\text{fresh } f}{x \sim \texttt{flip}(\theta) \rightsquigarrow \Big((x' \Leftrightarrow f) \wedge (\text{rest unchanged}), w\Big)}$$

All variables in the program except for x are not changed by this statement

Symbolic compilation: Assignment

- Compositional process $\texttt{s} \rightsquigarrow (\varphi, w)$

$$\frac{}{x := \texttt{e} \rightsquigarrow \Big((x' \Leftrightarrow \texttt{e}) \wedge (\text{rest unchanged}), w \Big)}$$

- Captures program structure in the logical expression

```
x := a || b || c || d || e || f
```

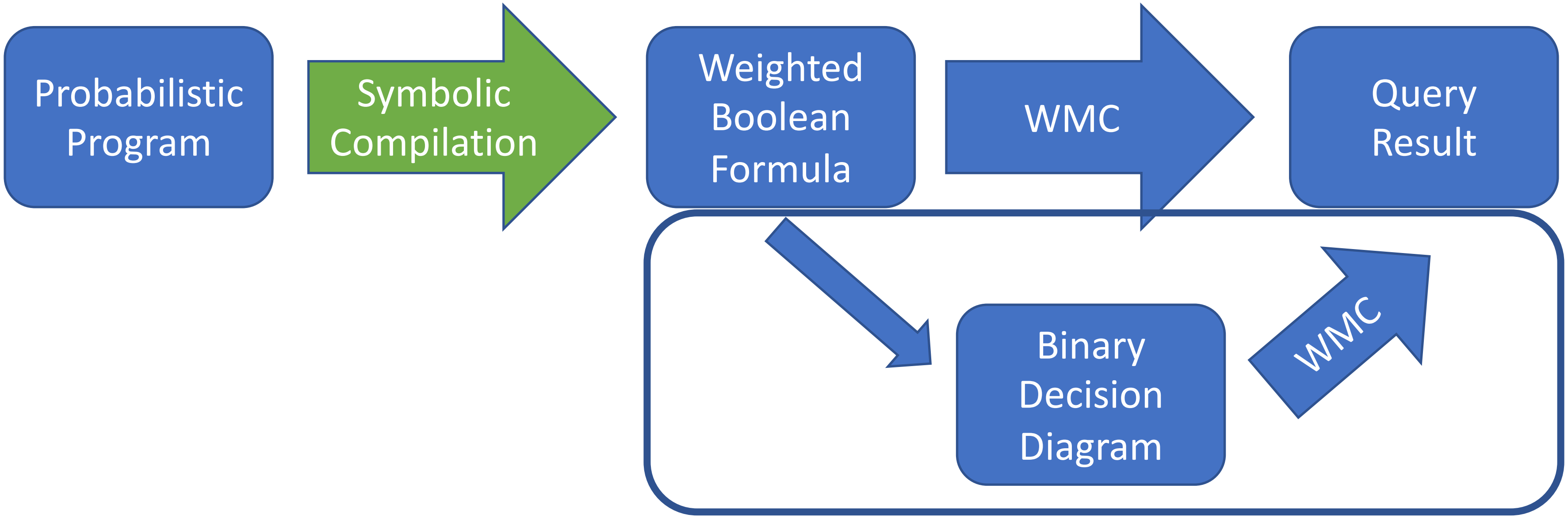
Symbolic compilation: Sequencing

- Compositional process $\mathtt{s} \rightsquigarrow (\varphi, w)$

$$\frac{s_1 \rightsquigarrow (\varphi_1, w_1) \qquad s_2 \rightsquigarrow (\varphi_2, w_2) \\ \varphi_2' = \varphi_2[x_i \mapsto x_i', x_i' \mapsto x_i'']}{\mathtt{s}_1; \mathtt{s}_2 \rightsquigarrow ((\exists x_i'. \varphi_1 \wedge \varphi_2')[x_i'' \mapsto x_i'], w_1 \uplus w_2)}$$

- Compile two sub-statements, do some relabeling, then combine them to get the result

Inference via Weighted Model Counting



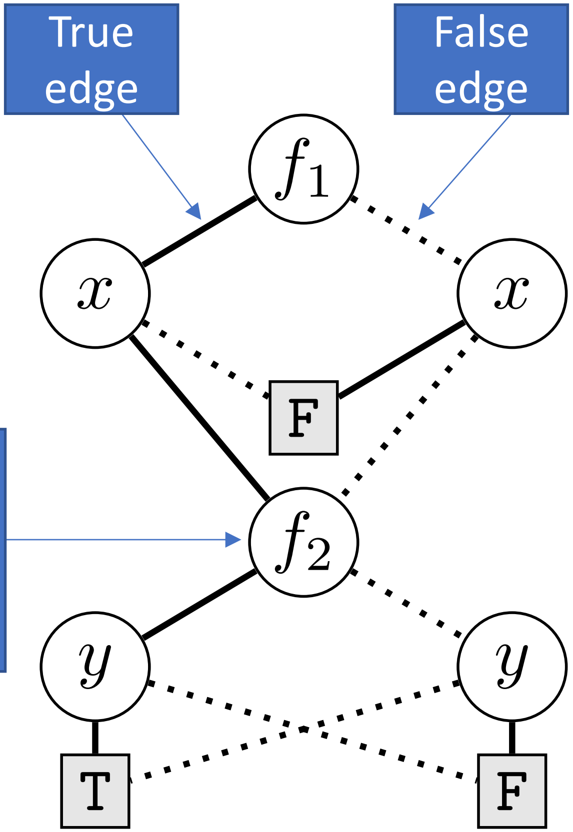
Compiling to BDDs

- Consider an example program:

```
x~flip(0.4);
y~flip(0.6)
```

$$(\mathtt{x} \iff f_1) \wedge (y \iff f_2)$$

This sub-function does not depend on x: exploits independence

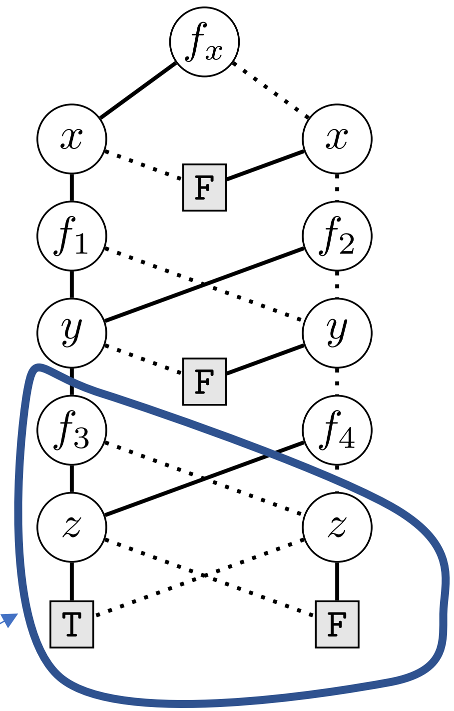


- WMC is efficient for BDDs: *time linear* in size
  - Small BDD = Fast Inference

BDDs exploit conditional independence

- Size of BDD grows linearly with length of Markov chain

```
x ~flip_x(0.5);
if(x) { y ~flip_1(0.6) }
else { y ~flip_2(0.4) };
if(y) { z ~flip_3(0.6) }
else { z ~flip_4(0.9) }
```

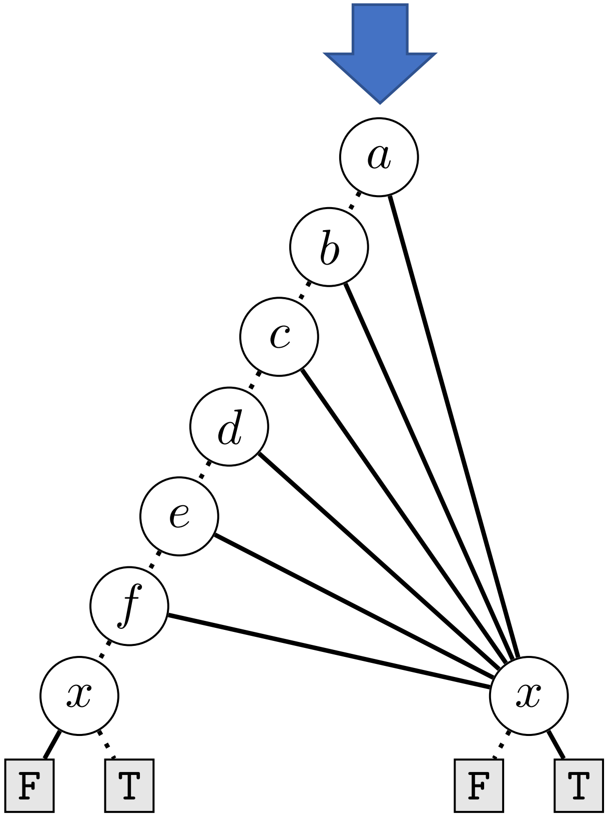


Given y=T, does not depend on the value of X: exploits conditional independence

Compiling to BDDs

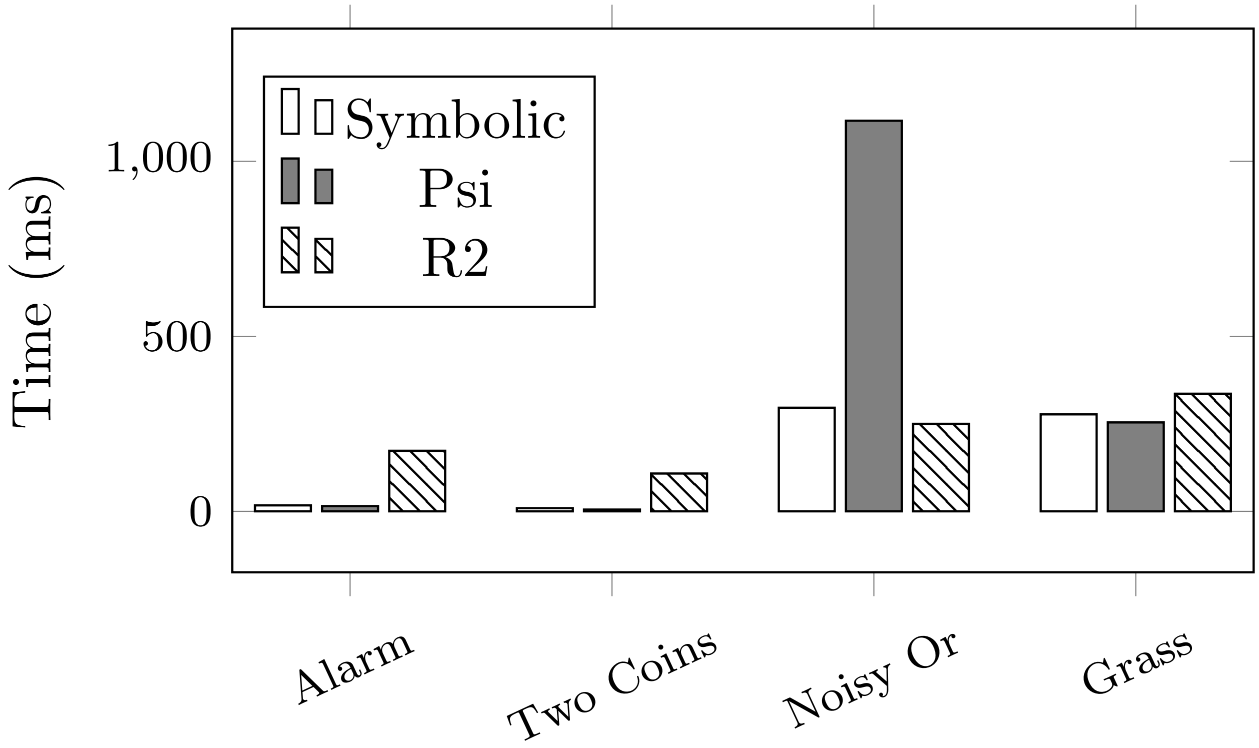
- BDDs compactly capture complex program structure

```
x = a || b || c || d || e || f;
```

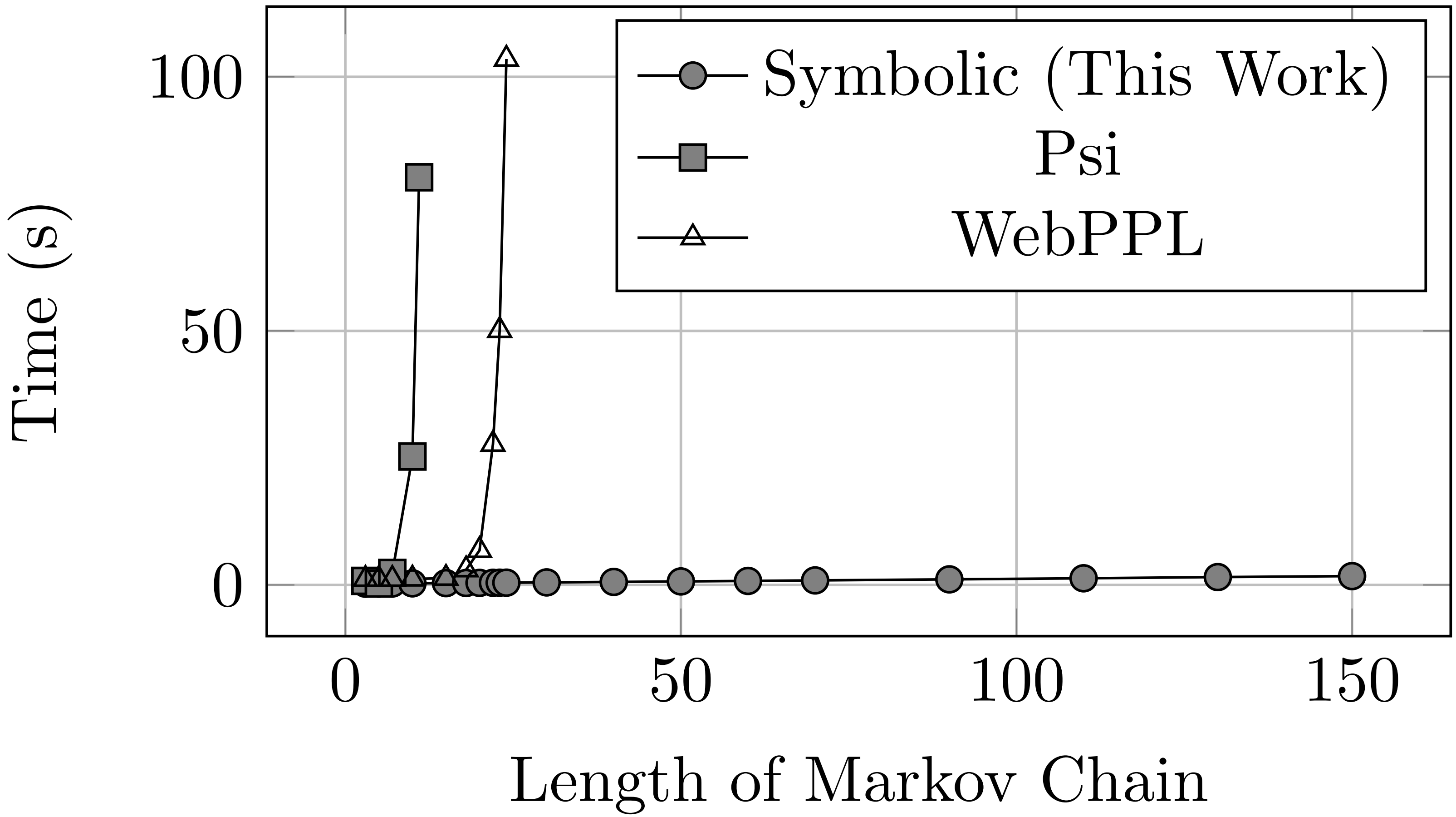


Experiments: Well-known Baselines

- Small programs (10s of lines)



Experiments: Markov Chain

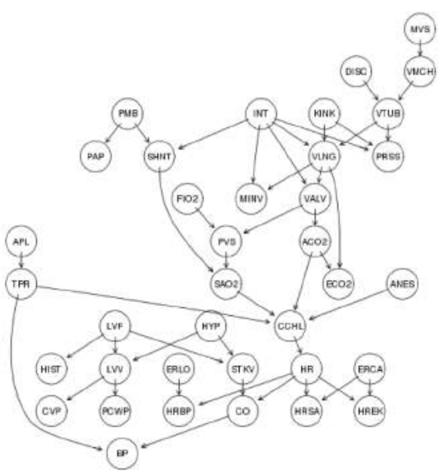


Experiment: Bayesian Network Encodings

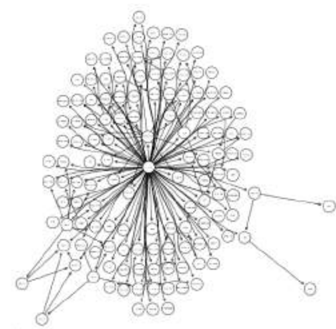
- Larger programs (thousands of lines, tens of thousands of flips)

Model	Us (s)	BN Time (s)	Size of BDD
Alarm	1.872	0.21	52k
Halfinder	12.652	1.37	157k
Hepar2	7.834	Not reported	139k
pathfinder	62.034	14.94	392k

Specialized BN inference algorithm



Alarm Network



Pathfinder Network

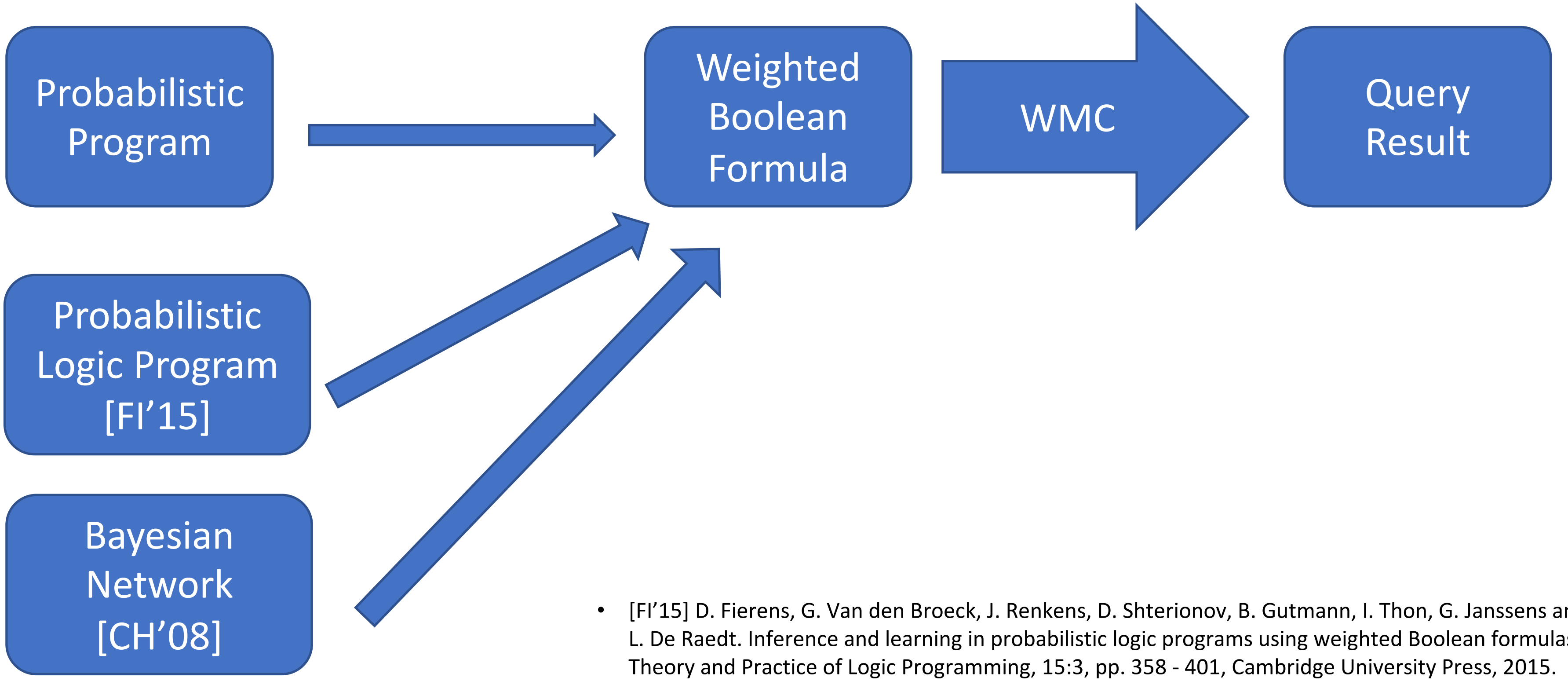
Probabilistic model checking

- Notable systems: STORM [DE'17], PRISM [KW'11]
- Different family of queries
  - Focus on finding upper/lower bounds on probabilities, not Bayesian inference
- Different symbolic representation of distribution
  - ADDs (aka. MTBDDs) instead of weighted model counting (also used by [CL'13])
  - Cannot exploit independence (but can exploit sparsity)

- [DE'17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. Proc. of CAV, Volume 10427 of LNCS, pages 592–600, Springer, 2017.
- [KW'11] Marta Kwiatkowska, Gethin Norman and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer, 2011.
- [CL'13] Claret, G., Rajamani, S. K., Nori, A. V, Gordon, A. D., & Borgström, J. (2013). Bayesian Inference Using Data Flow Analysis. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 92–102. https://doi.org/10.1145/2491411.2491423

Inference via WMC

- Has been applied to models other than discrete probabilistic programs



- [FI’15] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens and L. De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming, 15:3, pp. 358 - 401, Cambridge University Press, 2015.
- [CH’08] Chavira, M., & Darwiche, A. (2008). On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6–7), 772–799. https://doi.org/10.1016/j.artint.2007.11.002

Future Work and Conclusion

- We described a *symbolic exact* approach to inference in discrete probabilistic programs
  - Avoids combinatorial explosion of variable enumeration
  - Systematically exploits nuanced program structure like independence
  - Competitive with exact inference Bayesian network inference techniques
  - Gave a semantics, proved it corresponds with compilation

Future Work and Conclusion

- Extending to more expressive program constructs
  - Loops: symbolic fixpoint construction
  - Procedures: exploiting structure of repeated calls
  - Datatypes: categorical, algebraic types
- Theoretical analysis of inference
  - What program properties make queries harder or easier?
- Alternative symbolic representations beyond BDDs
- Integrating exact discrete inference into systems which do not currently handle it?

Thank you!

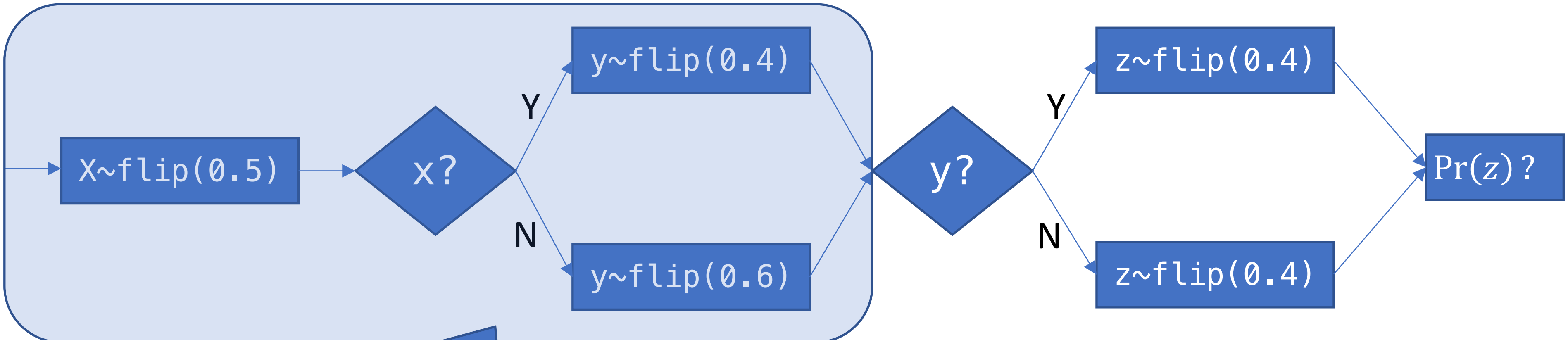
Questions?

Contact me: `sholtzen@cs.ucla.edu`

Extra Slides

Doing better than path-based inference

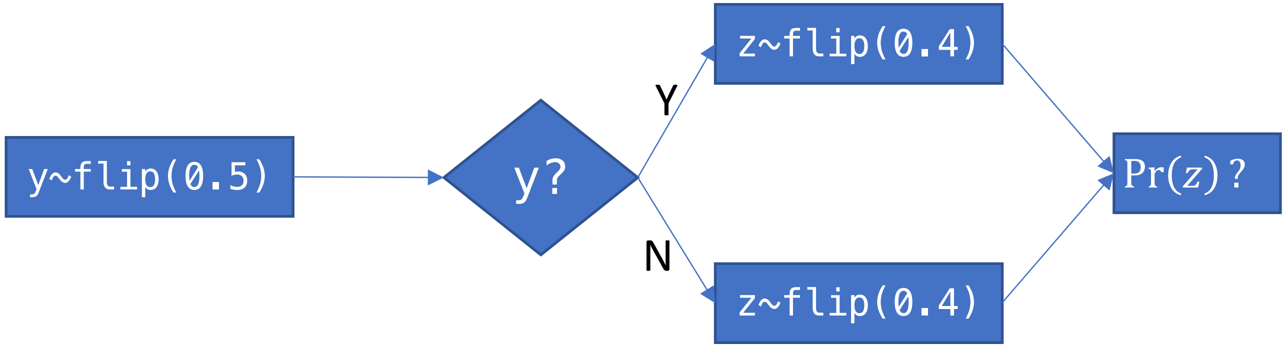
- *Observation*: z is independent of x given y



Can be summarized by computing
$\Pr(y) = 0.5\times 0.4 + 0.5\times 0.6 = 0.5$

Doing better than path-based inference

- *Observation*: z is independent of x given y



- Program now has only 2 paths

Semantics

- Goal: Prove inference correct
  - Semantics of statements naturally encoded as *conditional probabilities*

```
x ~ flip(0.4);
```

$(x' \Leftrightarrow f_1)$

x’	x	f_1	Pr?
1	1	1	0.4
1	1	0	0
1	0	1	0.4
1	0	0	0
0	1	1	0
0	1	0	0.6
0	0	1	0
0	0	0	0.6

Symbolic execution

- SAT queries tell us *reachability*

```
x := y;
```

$$\varphi = (x' \Leftrightarrow y) \wedge (y' \Leftrightarrow y)$$

x'	x	y'	y	SAT?
1	1	1	1	Y
1	1	0	1	N
1	1	0	0	N
1	0	1	1	Y
…				

"Can I start in state $(x \wedge \bar{y})$ and end in state $(x \wedge y)$"?

$$\mathrm{SAT}\big(\varphi \wedge (x \wedge \bar{y}) \wedge (x' \wedge y')\big) = F$$

Transition probability

- Assign a *probability* to transitioning between states

Problem: This table is huge!

Q: How can we compactly represent it?

x'	x	f_1	Pr?
1	1	1	0.4
1	1	0	0
1	0	1	0.4
1	0	0	0
0	1	1	0
0	1	0	0.6
0	0	1	0
0	0	0	0.6

Table shows *conditional probability* of starting in x and ending in x'

Weighted Model Counting

- Given Boolean formula φ, weight function w, $\mathrm{WMC}(\varphi, w) = \sum_{m \models \varphi} \prod_{l \in m} w(l)$.
- WMC queries tell us *transition probability*

"What is the probability of starting in state x and ending in state x'?"

$$\mathrm{WMC}\left((x' \Leftrightarrow f_1) \land x' \land x, \; w\right) = 0.4$$

l	$w(l)$
x	1
$\bar{x}$	1
f_1	0.4
$\bar{f_1}$	0.6

x'	x	f_1	Pr?
1	1	1	0.4
1	1	0	0
1	0	1	0.4
1	0	0	0
0	1	1	0
0	1	0	0.6
0	0	1	0
0	0	0	0.6

Inference via Weighted Model Counting

Probabilistic Program → Symbolic Compilation → Weighted Boolean Formula → WMC → Query Result

```
x ~ flip(0.4);
```

$$(x' \Leftrightarrow f_1)$$

l	$w(l)$
x	1
$\bar{x}$	1
f_1	0.4
$\bar{f_1}$	0.6

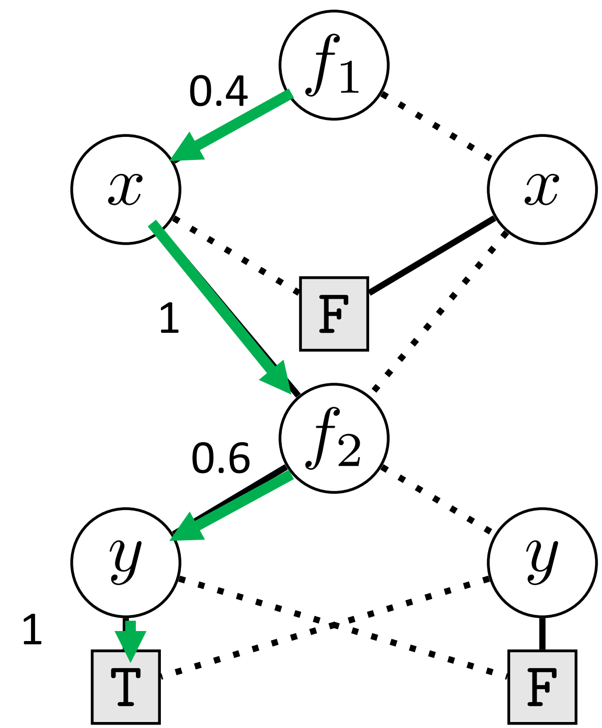
Q: How can we do this efficiently?

(i.e., without building the whole transition probability table)

Compiling to BDDs

- BDD = compact representation of transition probability table

```
x~flip(0.4);
y~flip(0.6)
```



Size linear in # variables, exploits independence

$$\Pr(x = T, y = T) = 0.4 * 0.6 * 1 * 1$$

Querying with BDDs

- Suppose we want to compute $\Pr(x)$

```
x~flip(0.4);
y~flip(0.6)
```

$$\Pr(x) = 1.0 * 0.4 + 0.6 * 0 = 0.4$$