A program analysis perspective

Probabilistic Programming

Aditya V. Nori   Sriram K. Rajamani
Microsoft Research India

Collaborators

- Johannes Borgstorm (Uppsala University)
- Arun Chaganty (Stanford University)
- Guillaume Claret (ENS, Paris)
- Andy Gordon (MSR Cambridge)
- Akash Lal (MSR India)
- Selva Samuel (MSR India)

Background

"Usual" programs in "usual" languages, such as **C**, **Java**, **C#**, **LISP** or **Scheme** with two added features:

1. *The ability to sample from a distribution*
2. *The ability to condition values of variables through observations*

Goal of a probabilistic program: succinctly specify a probability distribution

Goal of inference: infer the distribution specified by a probabilistic program

Simple probabilistic program

```
bool c1, c2;
c1 = Bernoulli(0.5);
c2 = Bernoulli(0.5);
```

$c1$	$c2$	$P(c1, c2)$
0	0	$1/4$
0	1	$1/4$
1	0	$1/4$
1	1	$1/4$

Probabilistic program with conditioning

```
bool c1, c2;
c1 = Bernoulli(0.5);
c2 = Bernoulli(0.5);
observe(c1 || c2);
```

$c1$	$c2$	$P(c1, c2)$
0	0	0
0	1	$1/3$
1	0	$1/3$
1	1	$1/3$

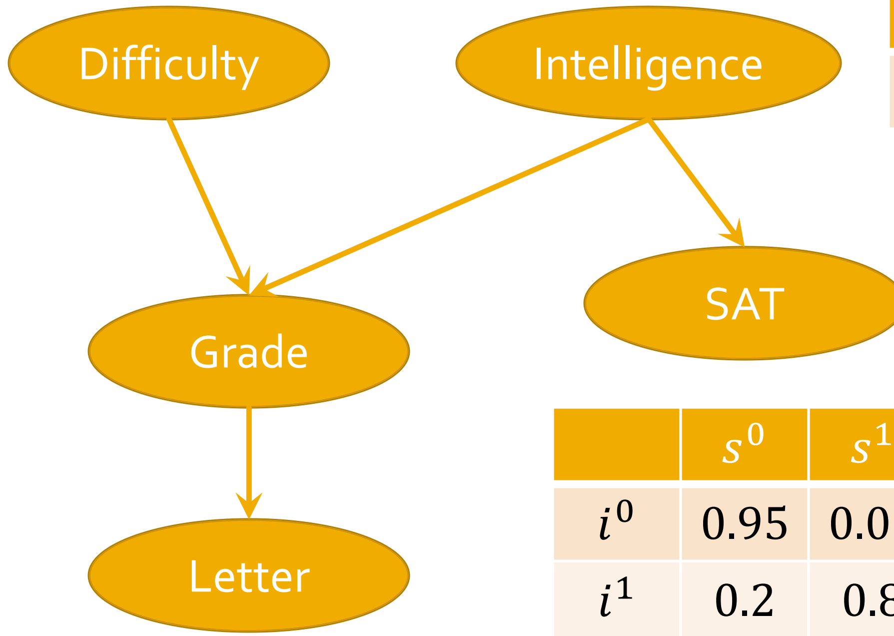
Bayesian networks (BNs)

- A Bayesian network is a DAG in which every node is a conditional probability distribution (CPD) …

Example BN (from PGM book)

d^0	d^1
0.6	0.4

i^0	i^1
0.6	0.4



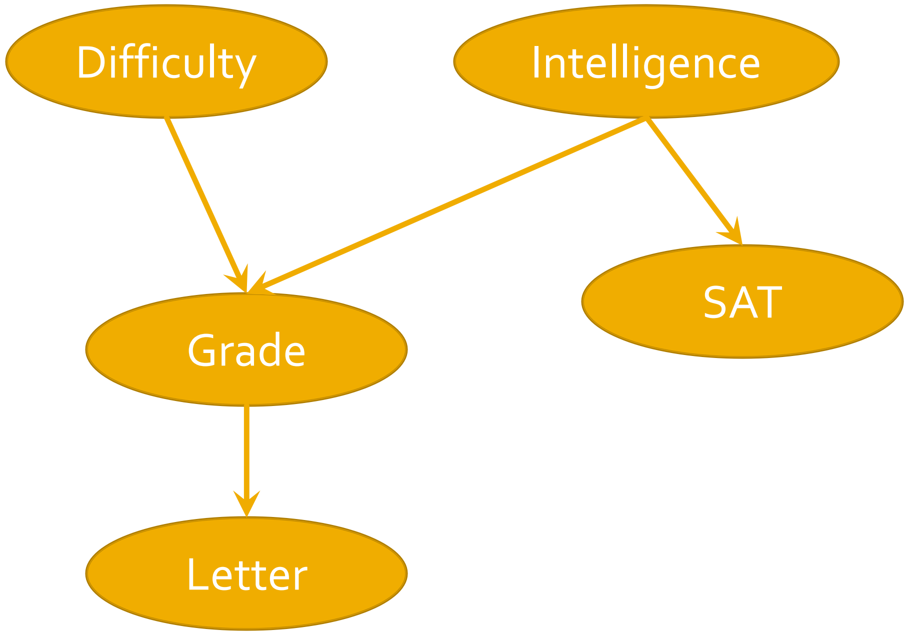
	g^1	g^2	g^3
i^0, d^0	0.3	04.	0.3
i^1, d^1	0.05	0.25	0.7
i^1, d^0	0.9	0.08	0.02
i^1, d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

$$P(d^0, i^1, g^3, s^1, l^1) = 0.6 \times 0.3 \times 0.02 \times 0.8 \times 0.01$$

BNs as PPs




```
d = Discrete({0.6, 0.4});
i = Discrete({0.7, 0.3});
//grade
if(i==0 && d==0)
  g = Discrete({0.3, 0.4, 0.3});
else if(i==0 && d==1)
  g = Discrete({0.05, 0.25, 0.7});
else if(i==1 && d==0)
  g = Discrete({0.9, 0.08, 0.02});
else g = Discrete({0.5, 0.3, 0.2});
//SAT
if (i==0) s = Discrete({0.95, 0.05})
else s = Discrete({0.2, 0.8})
//Letter
if(g==1) l = Discrete({0.1, 0.9})
else if (g==2) l = Discrete({0.4, 0.6})
else if (g==3) l = Discrete({0.99, 0.01})
```

Markov chains

$D = (S, s_{init}, P, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$

AP$= (try, fail, succ)$

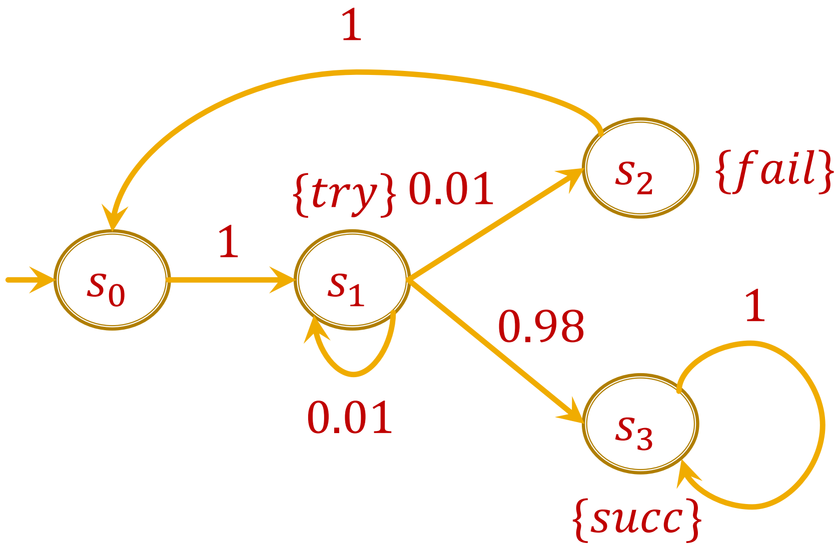
$L(s_0) = \emptyset,$

$L(s_1) = \{try\},$

$L(s_2) = \{fail\},$

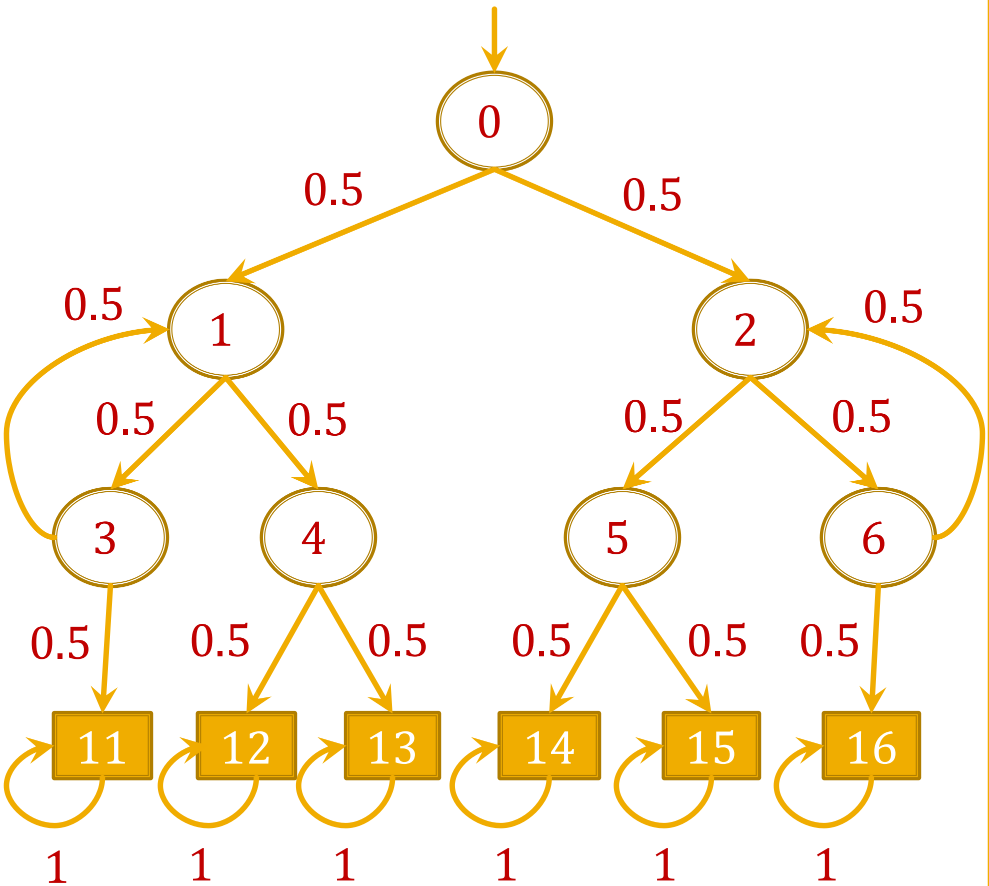
$L(s_3) = \{succ\}$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(example: courtesy David Parker’s course)

MCs as PPs



Knuth–Yao's technique to get a fair die from fair coin tosses

```
int nextState( int curState) {
  bool coin = Bernoulii(0.5);
  switch (curState){
    case(0):
      if (coin) return 1 else return 2;
    case(1):
      if (coin) return 3 else return 4;
    case(2):
      if (coin) return 5 else return 6;
    case(3):
      if (coin) return 1 else return 11;
    case(4):
      if (coin) return 12 else return 13;
    case(5):
      if (coin) return 14 else return 15;
    case(6):
      if (coin) return 15 else return 16;
  }
}
main() {
  int x = 0;
  while (x < 11) {  x = nextState(x); }
  return (x);
}
```

Halo multiplayer

- How are skills modeled?
- Player A beats Player B
  - $skillA > skillB$?

TrueSkill

```
float skillA, skillB, skillC;
float perfA1, perfB1, perfB2,
      perfC2, perfA3, perfC3;
skillA = Gaussian(100, 10);
skillB = Gaussian(100, 10);
skillC = Gaussian(100, 10);

// first game: A vs B, A won
perfA1 = Gaussian(skillA, 15);
perfB1 = Gaussian(skillB, 15);
observe(perfA1 > perfB1);

// second game: B vs C, B won
perfB2 = Gaussian(skillA, 15);
perfC2 = Gaussian(skillB, 15);
observe(perfB2 > perfC2);

// third game: A vs C, A won
perfA3 = Gaussian(skillA, 15);
perfC3 = Gaussian(skillB, 15);
observe(perfA3 > perfC3);
```

- Sample $perfA$ from a noisy $skillA$ distribution
- Sample $perfB$ from a noisy $skillB$ distribution
- if $perfA > perfB$ then A wins else B wins

$skillA = Gaussian(102.1, 7.8)$
$skillB = Gaussian(100.0, 7.6)$
$skillC = Gaussian(97.9, 7.8)$

Goal of inference

- *Infer the distribution specified by a probabilistic program*
- What does this formally mean?
- Explore techniques to perform inference

Restricted model: Boolean Probabilistic Programs (BPP)

$r \in \mathbb{R}$

$x \in Vars$

$\mathrm{T} ::= bool$	types
$uop ::= not$	unary operators
$bop ::= and \mid or$	binary operators
$D ::= \mid \mathrm{T} x_1, \mathrm{x}_2, \ldots, x_n$	declaration
$\mathrm{E} ::=$	expressions
$\mid x$	variable
$\mid \mathrm{c}$	constant
$\mid \mathrm{E}_1\, bop\ E_2$	binary operation
$\mid uop\ \mathrm{E}$	unary operation
$S ::=$	statements
$\mid x := E$	deterministic assignment
$\mid x := Bernoulli(r)$	Bernoulli assignment
$\mid observe(E)$	observe
$\mid skip$	skip
$\mid S_1; S_2$	sequential composition
$\mid if\ E\ then\ S_1 else\ S_2$	conditional composition
$\mid while\ E\ do\ S$	loop
$P ::= \mathrm{D\ S}$	programs

Operational semantics of BPPs

- States: σ, valuation to all variables $x_1, x_2, \ldots, x_n$
- Set of all states: Γ
- Configuration: $\langle \sigma, statement \rangle$

Operational semantics of BPPs …

$\langle \sigma, x \coloneqq E\rangle \rightarrow^{1} \langle \sigma[x \leftarrow \sigma(E)], skip\rangle$
$\langle \sigma, x \coloneqq Bernoulli(r)\rangle \rightarrow^{r} \langle \sigma[x \leftarrow true], skip\rangle$
$\langle \sigma, x \coloneqq Bernoulli(r)\rangle \rightarrow^{1-r} \langle \sigma[x \leftarrow false], skip\rangle$
$\langle \sigma, x \coloneqq observe(E)\rangle \rightarrow^{1} \langle \sigma, skip\rangle, if\ \sigma(E) = true$
$\langle \sigma, skip; S\rangle \rightarrow^{1} \langle \sigma, S\rangle$
$\langle \sigma, S_1; S_2\rangle \rightarrow^{p} \langle \sigma', S'; S_2\rangle, if\ \langle \sigma, S_1\rangle \rightarrow^{p} \langle \sigma', S'\rangle$
$\langle \sigma, if\ E\ then\ S_1 else\ S_2\rangle \rightarrow^{1} \langle \sigma, S_1\rangle, if\ \sigma(E) = true$
$\langle \sigma, if\ E\ then\ S_1 else\ S_2\rangle \rightarrow^{1} \langle \sigma, S_2\rangle, if\ \sigma(E) = false$
$\langle \sigma, while\ \ E\ do\ S\rangle \rightarrow^{1} \langle \sigma, skip\rangle, if\ \sigma(E) = false$
$\langle \sigma, while\ \ E\ do\ S\rangle \rightarrow^{1} \langle \sigma, S; while\ E\ do\ S\rangle, if\ \sigma(E) = true$

Run $\omega = \langle \sigma_0, \mathrm{S}_0\rangle \rightarrow^{p_1} \langle \sigma_1, \mathrm{S}_1\rangle \rightarrow^{p_2} \cdots \rightarrow^{p_n} \langle \sigma_n, skip\rangle$
$Prob(\omega) = p_1 p_2 \cdots p_n$

$$Prob(\sigma) = \sum_{\{\omega | \omega\ is\ a\ run\ that\ ends\ in\ state\ \sigma\}} Prob(\omega)$$

Semantics for general probabilistic programs

- Easy to add more general types (int, real) and continuous distributions
- Semantics can still be defined using measure theory
  - Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, Jurgen Van Gael. Measure Transformer Semantics for Bayesian Machine Learning. *ESOP 2011*

Inference inspired by program analysis

- Static analysis techniques, inspired by data flow analysis
- Dynamic analysis techniques, inspired by symbolic execution (and weakest preconditions)
- Iterative refinement techniques, inspired by CEGAR verification framework

Bayesian inference using data flow analysis

```
bool c1, c2;
c1 = Bernoulli(0.5);
c2 = Bernoulli(0.5);
observe(c1 || c2);
return (c1, c2);
```

$c1 = Bernoulli(0.5)$

c1	
t	1/2
f	1/2

$c2 = Bernoulli(0.5)$

c1	c2	
t	t	1/4
t	f	1/4
f	t	1/4
f	f	1/4

$observe(c1 \,||\, c2)$

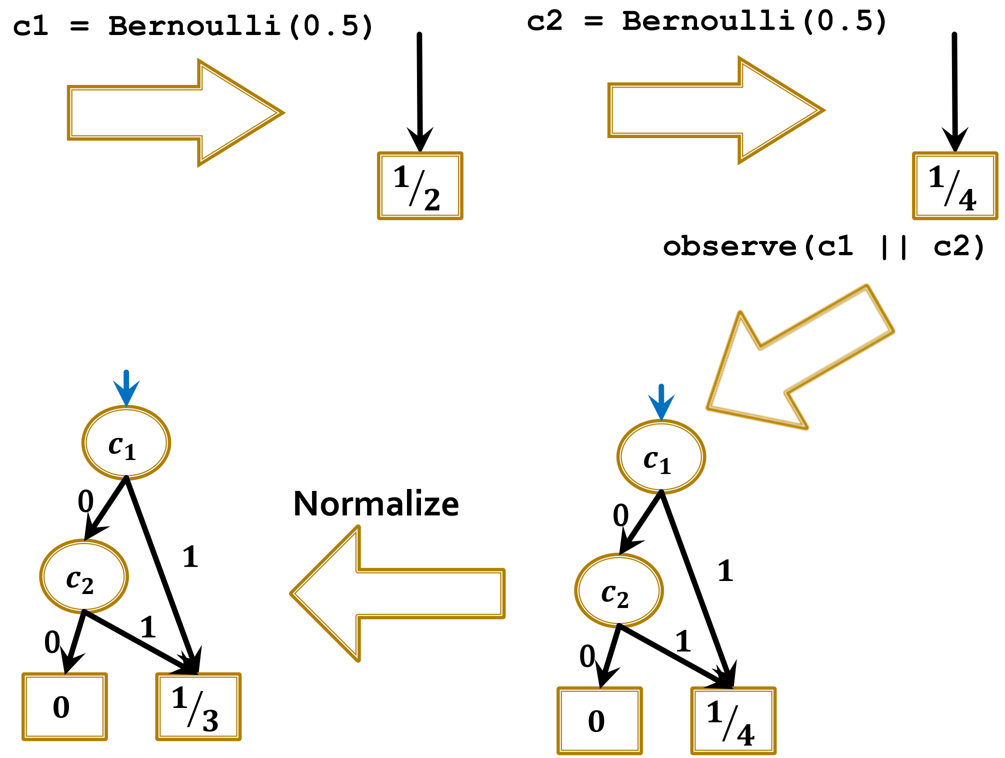
c1	c2	
t	t	1/4
t	f	1/4
f	t	1/4

Normalize

c1	c2	
t	t	1/3
t	f	1/3
f	t	1/3

Data flow analysis with ADDs

```
bool c1, c2;
c1 = Bernoulli(0.5);
c2 = Bernoulli(0.5);
observe(c1 || c2);
return (c1, c2);
```



Inference using data flow analysis

$Algorithm\ Post(\rho, S)$
$Input$: An input distribution ρ over the states of the program P, and a statement S
$Output$: Output distribution over the states of the program P

$switch(S)$
$\quad case\ x \coloneqq E$:
$\qquad return\ \lambda\sigma. \sum_{\{\sigma' | \sigma'[x \leftarrow \sigma'(E)] = \sigma\}} \rho(\sigma')$
$\quad case\ x \coloneqq Bernoulli(r)$:
$\qquad return\ \lambda\sigma. (r \times \sum_{\{\sigma' | \sigma'[x \leftarrow true] = \sigma\}} \rho(\sigma') + (1-r) \times \sum_{\{\sigma' | \sigma'[x \leftarrow false] = \sigma\}} \rho(\sigma')$
$\quad case\ observe(E)$:
$\qquad return\ \lambda\sigma. ite(\sigma(E), \rho(\sigma), 0)$
$\quad case\ skip$:
$\qquad return\ \rho$
$\quad case\ S_1; S_2$:
$\qquad \rho' = Post(\rho, S_1)$;
$\qquad return\ Post(\rho, S')$
$\quad case\ if\ E\ then\ S_1 else\ S_2$:
$\qquad \rho_t = \lambda\sigma. ite(\sigma(E), \rho(\sigma), 0)$;
$\qquad \rho_f = \lambda\sigma. ite(\sigma(E), 0, \rho(\sigma))$;
$\qquad return\ \lambda\sigma. (Post(\rho_t, S_1)(\sigma) + Post(\rho_f, S_2)(\sigma)$
$\quad case\ while\ E\ do\ S$:
$\qquad \rho_p = \perp; \rho_c = \rho$;
$\qquad while\ (\rho_p \neq \rho_c) do$
$\qquad\quad \rho_p = \rho_c$;
$\qquad\quad \rho_c = Post(\rho, if\ E\ then\ S\ else\ skip)$
$\qquad end\ while$
$\qquad return\ \rho_c$
$end\ switch$

- Can “merge” at join points without losing precision
- Loops can be handled using fixpoints
- Theorem: If the $Post$ algorithm terminates, then it is guaranteed to compute the exact distribution specified by the probabilistic program

Empirical results

Benchmark	Parameters	SHENOY-SHAFER (seconds)	HUGIN (seconds)	ZC-HUGIN (seconds)	REC-COND (seconds)	OpenBugs (seconds)	GS (seconds)	EP (seconds)	ADD (seconds)
Students	s=10, c=10, t=4	0.38	0.40	0.41	0.53	$\perp$	0.88	1.57	0.11
Friends	p=4	0.40	0.41	0.41	0.50	4	7.7	4.09	0.19
	p=5	2.75	2.66	3.37	9.62	18	$\perp$	12.06	0.42
	p =6	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	27.3	4.78
Compare	n=10	0.28	0.26	0.29	0.33	3	$\perp$	1.58	0.15
	n=20	0.33	0.31	0.30	0.37	2	$\perp$	2.34	0.16
	n=100	0.53	0.55	0.52	0.92	6	$\perp$	12.58	2.15

Efficiently sampling probabilistic programs via program analysis

- Run the program multiple times and compute statistics over resulting samples (BLOG, Church)
- Efficiently run the program without rejecting samples (based on importance sampling)

Dynamic analysis

I. Decompose program into "simple" straight-line programs

II. Efficiently *sample* from these straight-line programs by *executing* them

III. Combine results in order to compute expectations

The key equation

$$E_{\Pi}[x] = \sum_i P(\pi_i) E_{\pi_i}[x]$$

- Estimate $E_{\pi_i}[x]$ using importance sampling and Djikstra's weakest preconditions
- Estimate $P(\pi_i)$ using importance sampling
- Able to prove convergence (even for programs with unbounded recursion!)

The Qi algorithm

$Qi(\pi, \kappa_1, \kappa_2)$

$\Pi := Explore(\pi, \kappa_1)$
$\Omega := \emptyset$
$\boldsymbol{for}\ \pi_i \in \Pi\ \boldsymbol{do}$
  $(\theta, y) := Estimate(\pi_i, \kappa_2)$
  $\Omega := \Omega \cup \{(\theta, y)\}$
$\boldsymbol{end\ for}$
$\boldsymbol{return}\ \overline{\Omega}$

$Explore(\pi_i, \kappa)$

$\pi_i^* := pp\_to\_np(\pi_i)$
$\Pi := \{\}$
$F := \{\sigma_0\}$
$d := d_0$
$\boldsymbol{loop}$
  $(C, F) := Execute(\pi_i^*, F, d)$
  $\Pi := \Pi \cup C$
  $\boldsymbol{if}\ |\Pi| \geq \kappa\ \boldsymbol{then}$
    $\boldsymbol{break}$
  $\boldsymbol{else}$
    $d := d + \delta$
  $\boldsymbol{endif}$
$\boldsymbol{end\ loop}$
$\boldsymbol{return}\ \Pi$

Pearl's burglar alarm example

```
int alarm() {
  char earthquake = Bernoulli(0.001);
  char burglary = Bernoulli(0.01);
  char alarm = earthquake || burglary;
  char phoneWorking =
    (earthquake)? Bernoulli(0.6) : Bernoulli(0.99);
  char maryWakes;
  if (alarm && earthquake)
    maryWakes = Bernoulli(0.8);
  else if (alarm)
    maryWakes = Bernoulli(0.6);
  else maryWakes = Bernoulli(0.2);
  char called = maryWakes && phoneWorking;
  observe(called);
  return burglary;
}
```

Explore ≡ Symbolic execution

```
int alarm() {
  char earthquake = Bernoulli(0.001);
  char burglary = Bernoulli(0.01);
  char alarm = earthquake || burglary;
  char phoneWorking =
    (earthquake)? Bernoulli(0.6) : Bernoulli(0.99);
  char maryWakes;
  if (alarm && earthquake)
    maryWakes = Bernoulli(0.8);
  else if (alarm)
    maryWakes = Bernoulli(0.6);
  else maryWakes = Bernoulli(0.2);
  char called = maryWakes && phoneWorking;
  observe(called);
  return burglary;
```

```
int alarm() {
  char earthquake = Bernoulli(0.001);
  char burglary = Bernoulli(0.01);
  char alarm = earthquake || burglary;
  observe(earthquake);
  char phoneWorking = Bernoulli(0.6);
  observe(alarm && earthquake);
  char maryWakes = Bernoulli(0.8);
  char called = maryWakes && phoneWorking;
  observe(called);
  return burglary;
}
```

Sampling from straight-line programs

- Need to ensure that samples drawn from primitive distributions satisfy observations
- “Hoist” conditions to the primitive distributions and sample from resulting conditional distributions

Djikstra's weakest precondition in action

Statement	WP
`earthquake = Bernoulli(0.001)`	$earthquake$
`burglary = Bernoulli(0.001)`	$earthquake$
`alarm = earthquake || burglary`	$alarm \wedge earthquake$
`observe(earthquake)`	$alarm \wedge earthquake$
`phoneWorking = Bernoulli(0.6);`	$alarm \wedge earthquake \wedge phoneWorking$
`observe(alarm && earthquake);`	$phoneWorking$
`maryWakes = Bernoulli(0.8);`	$maryWakes \wedge phoneWorking$
`called = maryWakes && phoneWorking`	$called$
`observe(called);`	$true$
`return burglary;`	$true$

Evaluation: benchmarks

Name	Description
Grass Model	Small model relating the probability of rain, having observed a wet lawn
Burglar Alarm	Described earlier
Noisy OR	Given a DAG, each node is a noisy-or of its parents. Find the posterior marginal probability of a node, given observations
Red Light Game	Planning-as-inference example in which the probability of winning the game given the action is modeled. Notably, this program exhibits unbounded recursion

Evaluation: results

Name	Algorithm	Samples (Rej.)	Estimated value	Time (sec)
Grass Model	Exact Qi Church	 600 600 (940)	0.7079 0.70107±1e-4 0.70391±1e-4	 1.1 4.9
Burglar Alarm	Exact Qi Church	 30 200 (1925)	0.0743 0.0743±0 0.0675±3e-4	 1.0 12.7
Noisy OR	Exact Qi Church	 2000 5000 (16573)	0.4626 0.465±1e-4 0.463±3e-4	 1.9 84.3
Red Light Game	Exact Qi Church	 200 200 (24732)	0.75 0.7683±0 0.5985±7e-4	 7.1 163.1

Relational learning

Inferring relationships from a data corpus using *probabilistic formulas* as specifications

Examples

Advisor-Advisee inference: Academic department data, papers coauthored by faculty and students, courses taught, teaching assistants

Bibliography inference:

Noisy bibliographic data from internet, different abbreviations of author names, conference names and paper titles , spelling errors and other variations in various words

Probabilistic formulas

Probabilistic formula is of the form

$$w : \varphi$$

Real number $0 \le w \le 1$

Formula in FOL

- Logic + Probability provides the tools to express specifications for inference
- Logic used to capture intuitions about how new relationships can be derived from existing relationships
- Probability used to model uncertainty and incompleteness (in our understanding), and presence of noise (in data)

More on probabilistic formulas

$$w : \neg\varphi \;=\; (1 - w) : \varphi$$

Formulas of the form $0{:}\,\varphi$ and $1{:}\,\varphi$ are called *axioms*

Example: De-duplicating citation data

[Sin

Dom
Rela
Sam .
Goa

Axioms

1.0 : $(\forall b\_0 \Rightarrow SameBib(b\_0, b\_0\ ))$
1.0 : $(\forall b\_0 b\_1\ .SameBib(b\_0, b\_1\ ) \Rightarrow SameBib(b\_1, b\_0\ ))$
1.0 : $(\forall b_0 b_1 b_2 . SameBib(b_0, b_1\ ) \wedge SameBib(b_1, b_2\ ) \Rightarrow SameBib(b_0, b_2))$

1.0: $(\forall b_0 b_1\ .SameBib(b_0, b_1\ )$
$\Rightarrow SameAuthor\big(BibAuthor(b_0), BibAuthor(b_1)\big)$
1.0: $(\forall b_0 b_1\ .SameBib(b_0, b_1\ ) \Rightarrow SameTitle\big(BibTitle(b_0), BibTitle(b_1)\big)$

Probabilistic formula (part of spec):
0.9: $(\forall b_0 b_1\ .SameAuthor\big(BibAuthor(b_0), BibAuthor(b_1)\big)$
$\wedge SameTitle\big(BibTitle(b_0), BibTitle(b_1)\big) \Rightarrow SameBib(b_0, b_1))$

Example: De-duplicating citation data

[Sing

Dom
Rela
Sam
Goa

```
for b0 in Bibs {
  for b1 in Bibs {
    if (b0 == b1) {
      observe(BibAuthor(b0) == BibAuthor(b1));
      observe(BibTitle(b0) == BibTitle(b1));
    }

    if (Bernoulli(0.9))
      if (BibAuthor(b0) == BibAuthor(b1) &&
          BibTitle(b0) == BibTitle(b1))
        observe(b0 == b1);
   }
}
```

Markov Logic Network [Domingos et al]

MLN is a triple of the form

$$L = \langle D, R, F \rangle$$

Where

- D is a set of domains
- R is a set of relations
- F is a set of probabilistic formulas

A world ω is a valuation to all the relations R

An evidence ε is a valuation to some of the relations in R

The formulas F define a probability distribution over the likely worlds

Goal of MLN inference is to compute the most likely world (one with maximum probability), given evidence ε

MAP inference, formally defined

- Given a world ω, and a formula $\mathrm{w}\!: f$, let $\Phi(\omega, f)$ be w if ... otherwise
- The weight or lik... is: $\prod_{f \in F} \Phi(\omega, f)$
- MAP solution: world with maximum weight

This problem is NP-hard.
Stochastic search algorithms such as WalkSAT work well in practice.

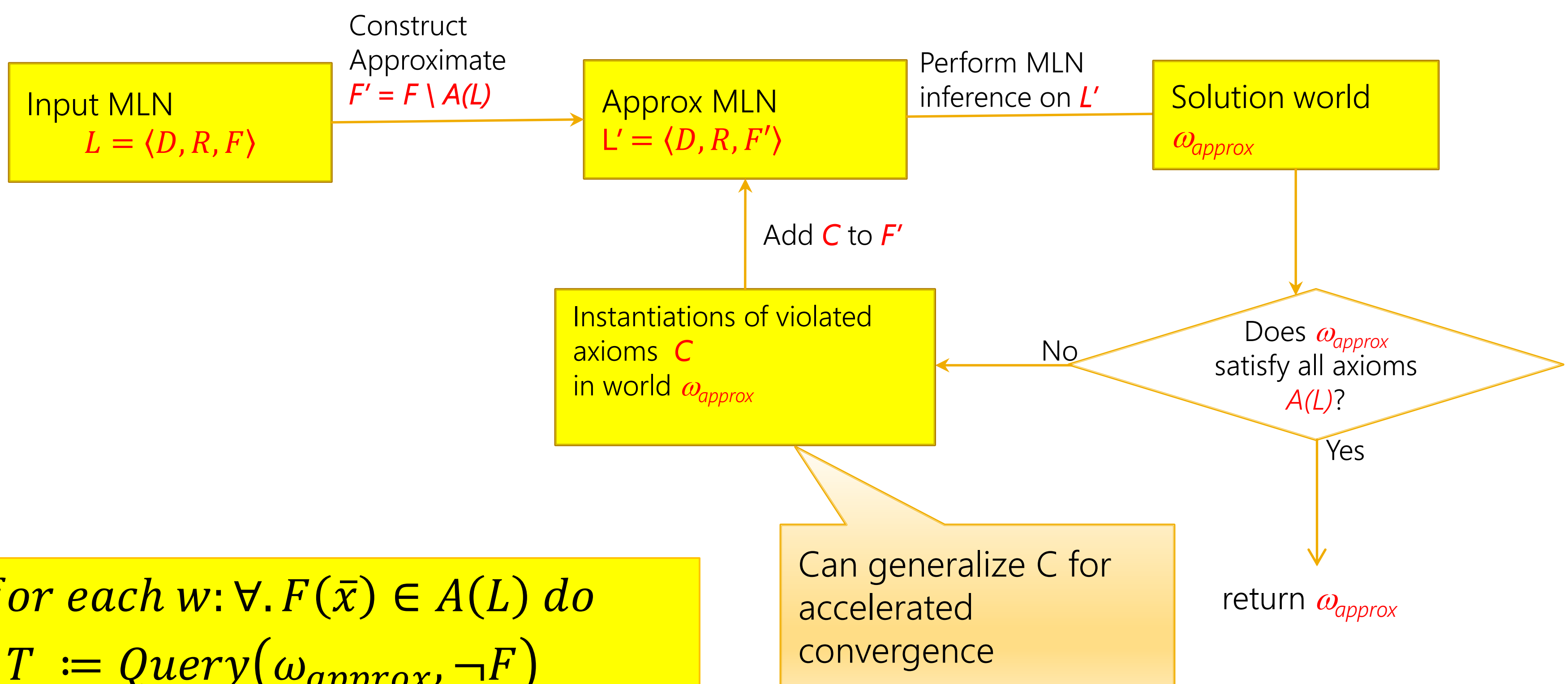
From WalkSAT to Quantifiers

- Quantified formulas are handled by instantiating them over the domain (this is called "grounding")
- Grounding is very expensive and slows down WalkSAT considerably
- Axioms (e.g. equivalence, congruence etc.) usually have lots of quantifiers

Key idea:

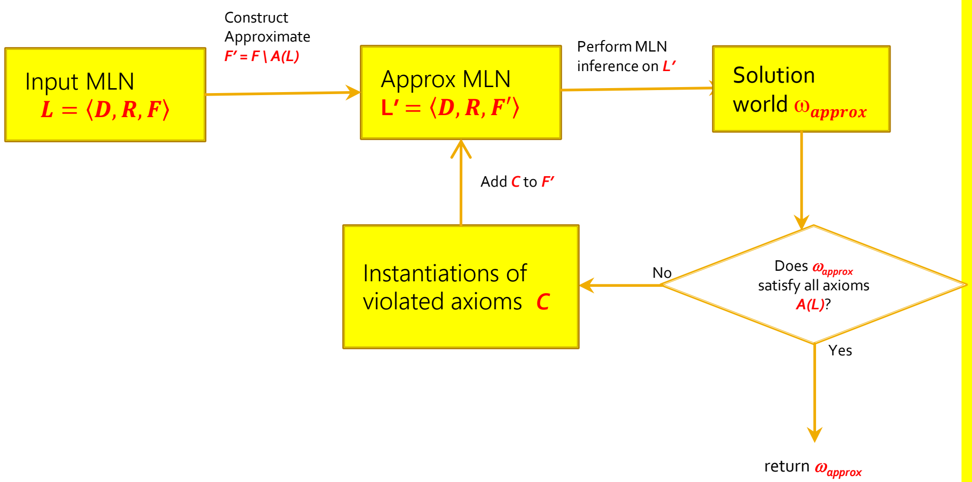
Can use CEGAR to lazily instantiate axioms

CEGAR for MAP



$$\begin{aligned}
&for\ each\ w{:}\ \forall.F(\bar{x}) \in A(L)\ do\\
&\quad T \ \coloneqq Query(\omega_{approx}, \neg F)\\
&\quad C' \coloneqq C' \cup \{1.0 : F(\bar{c}) \mid \bar{c} \in T\}
\end{aligned}$$

Why does this produce the correct MAP solution?



$$L_1 = \langle D, R, F \cup C_1 \rangle$$
$$L_2 = \langle D, R, F \cup C_2 \rangle$$

Suppose C_1 and C_2 contain only axioms, and $C_1 \subseteq C_2$

If a world w has a ***weight*** p in L_2, and satisfies all the axioms in $C_2 \setminus C_1$, then it has the same ***weight*** in L_1 as well.

weight(MAP(L_1)) $\geq$ ***weight***(MAP(L_2))

If world w is an MAP solution for L_1 and it satisfies all axioms in $C_2 \setminus C_1$, then w is an MAP solution for L_2

Evaluation

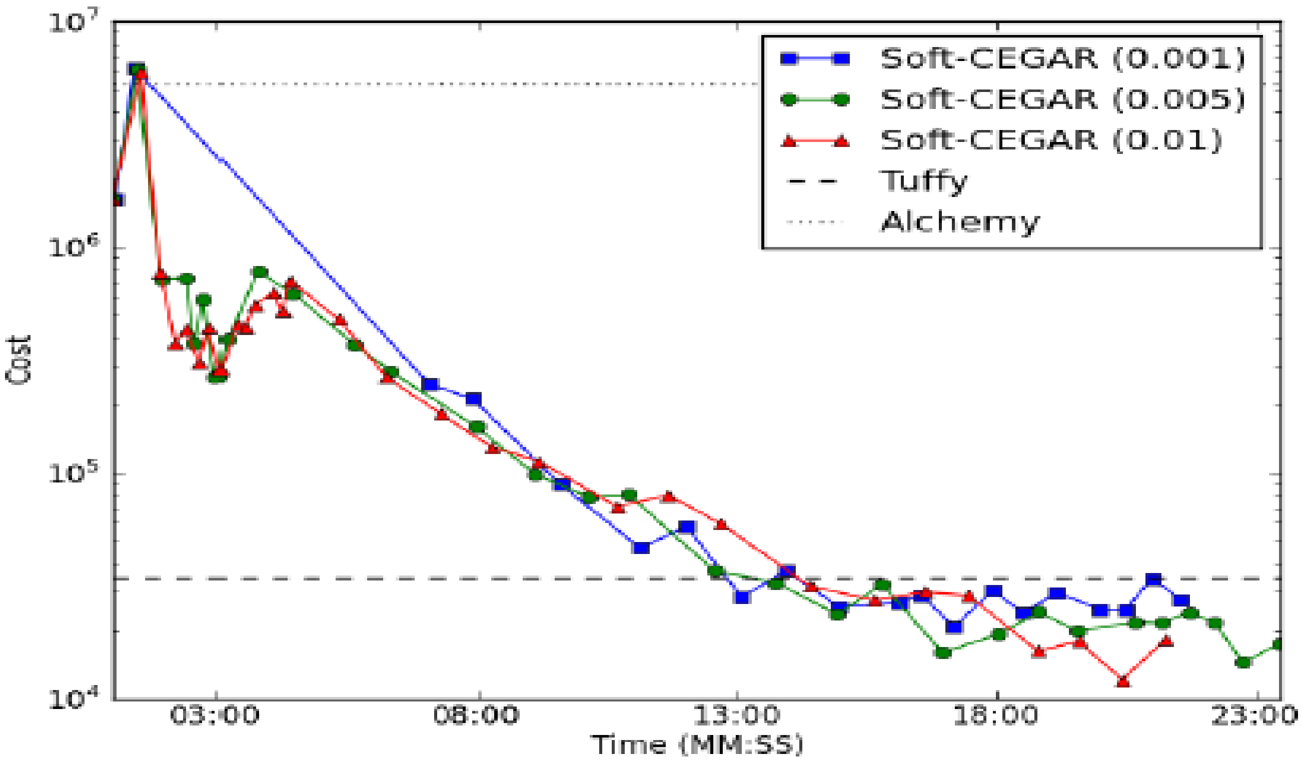
	AR	ER	IE	RC
#relations	14	14	19	5
#formula	24	3.8K	1.1K	32
#axioms	6	7	3	2
#atoms	88K	20K	81K	9860
#evidence-atoms	65K	676	613K	430K
#query-atoms	188	400	400	400

Application MLN and dataset statistics

Method	Iterations	Time	Solution Cost
Advisor Recommendation			
SOFT-CEGAR	18	06:44	3669.50
TUFFY	1	-	*
ALCHEMY	-	-	*
Entity Resolution			
SOFT-CEGAR	8	13:06	28112.24
TUFFY	1	15:13	34416.97
ALCHEMY	1	16:17	5287838.62
Information Extraction			
SOFT-CEGAR	3	17:46	109.40
TUFFY	1	55:49	3944.29
ALCHEMY	-	-	*
Relational Classification			
SOFT-CEGAR	2	05:00	870.37
TUFFY	1	05:42	874.63
ALCHEMY	-	-	*

Empirical evaluation of SOFT-CEGAR

Evaluation: Entity Resolution (ER)



- Cora dataset: 1295 citations and 132 distinct research papers

Summary

- **Probabilistic programs:** Succinct ways of specifying probabilistic models
- **Probabilistic inference using program analysis:**
    - Aditya V. Nori, Gil Hur, Sriram K. Rajamani, Selva Samuel. Semantics Sensitive Sampling. *Draft under review*
    - Gil Hur, Aditya V. Nori, Sriram K. Rajamani. Program Transformations for Probabilistic Inference. *Draft under review*
    - Arun T. Chaganty, Aditya V. Nori, and Sriram K. Rajamani. Efficiently Sampling Probabilistic Programs via Program Analysis. In *AISTATS '13: Artificial Intelligence and Statistics*, April 2013
    - Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon and Johannes Borgström. Bayesian Inference Using Data Flow Analysis. In *ESEC-FSE '13: Foundations of Software Engineering*, August 2013
    - Arun T. Chaganty, Akash Lal, Aditya V. Nori, and Sriram K. Rajamani. Combining Relational Learning with SMT Solvers using CEGAR. In *CAV '13: Computer Aided Verification*, July 2013
    - Andrew D. Gordon, Mihhail Aizatulin, Johannes Borgström, Guillaume Claret, Thore Graepel, Aditya V. Nori, Sriram K. Rajamani, and Claudio Russo. A Model-Learner Pattern for Bayesian Reasoning. In *POPL '13: Principles of Programming Languages*, January 2013