PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V8: Probabilistic Programming I

Özgür L. Özçep

Universität zu Lübeck

Institut für Informationssysteme

Today‘s Agenda (in classical linear form)

1. Premotivation: Probabilities
2. Motivation: Probabilistic Programming
3. Running Example
4. Semantics of Probabilistic Programs
5. Nonparametrics
6. Landscape of Probabilistic Programming Languages

PREMOTIVATION: PROBABILITIES

Remember: Problems with deep neural networks

- Very data hungry (e.g. often millions of examples)
- Very compute-intensive to train and deploy
- Poor at representing uncertainty
- Easily fooled by adversarial examples
- Finicky to optimise: non-convex + choice of architecture, learning procedure, expertise required
- Uninterpretable black-boxes, lacking in trasparency, difficult to trust

- => Amongst others, these problems lead to developments towards generative models (lecture V6)

Bayes‘ rule to rule them all ...

- If we use the mathematics of probability theory to express all forms of uncertainty and noise associated with our model ...
- ... then inverse probability (-> Bayes rule) allows us to infer unknown quantities, adapt our models, make predictions, and learn from data.

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

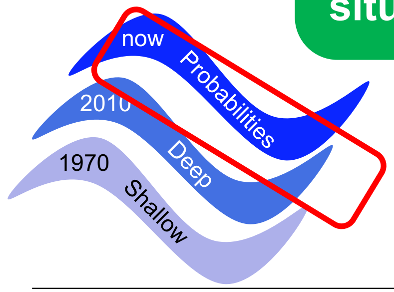
H = hypothesis, model

D = data, observation

Bayes‘ Rule

„ **The third wave of differentiable programming**

Getting deep systems that know when they do not know and, hence, recognise new situations and adapt to them



“ 1)

Kristian Kersting - Sum-Product Networks: The Third Wave of Differentiable Programming

---

1) Yes, a slide, quoting a slide

Reminder on basics of w.r.t. Bayes‘ Rule

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

- If $H \cup D$ is the set of all RVs, then $\boldsymbol{P}(H, D)$ is called the full joint distribution, which is all you need for inference tasks
- Bayes′rule relies on conditional probability
  - $P(H \mid D) = P(H, D) \,/\, P(D) \; if \;\; P(D) > 0$
- The step in the second equation relies on marginalization
  - $\boldsymbol{P}(D) = \sum_{h \in H} \boldsymbol{P}(D, h)$
- With conditional probabilities this gives conditioning (on H):
  - $\boldsymbol{P}(D) = \sum_{h \in H} \boldsymbol{P}(D \mid h) P(h)$

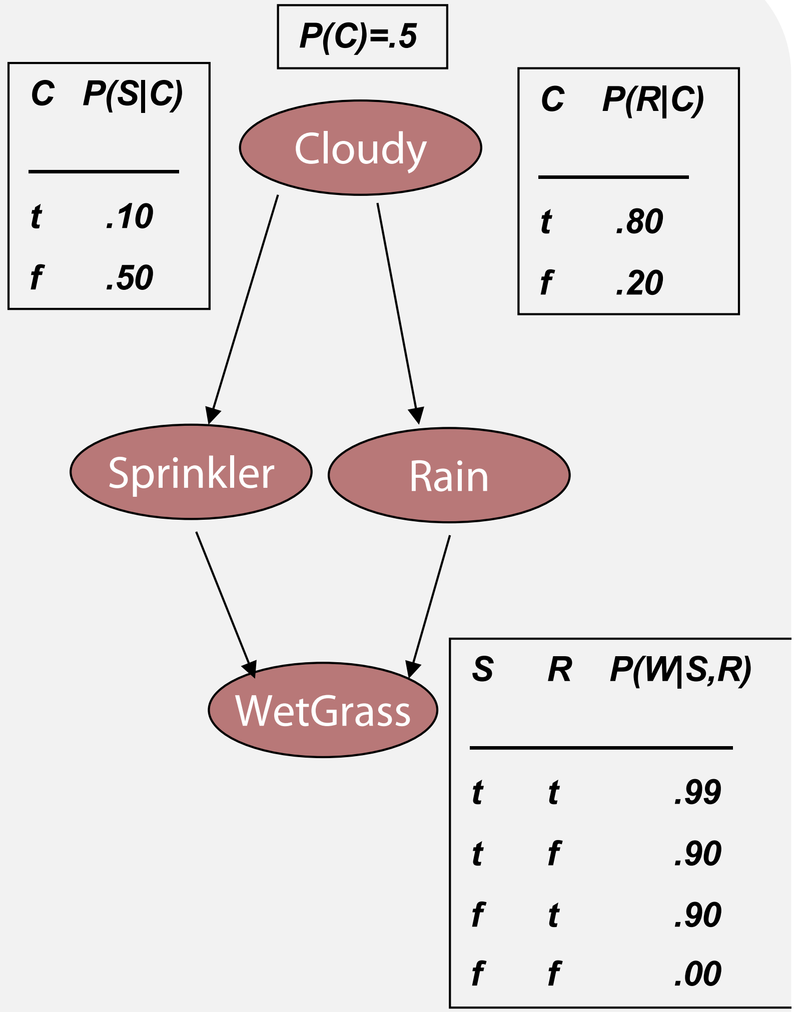
Reminder: Bayes Net/Probabilistic Graphical Model (PGM)

Idea: Encode efficiently (factorize) full joint probabilities

- Defines full joint distribution
  - $\boldsymbol{P}\,(X_1, \ldots, X_n) = \prod_{i=1}^{n} \boldsymbol{P}\,(X_i \mid Parents(X_i))$
  - Here:
    $\boldsymbol{P}(S, C, R, W)$
    $= \boldsymbol{P}(C)\boldsymbol{P}(S|C)\boldsymbol{P}(R|C)\boldsymbol{P}(W|S, R)$

    Gives, e.g.,
    $P(c, s, \neg r, w) = 0.5 \cdot 0.1 \cdot 0.2 \cdot 0.9$
- Graph topology encodes independencies of variables (efficiency!); e.g.
- S independent of R given C:

$$P(S|C) = P(S|C, R)$$



For in-depth treatment of (other) PGMs see (Koller/Friedman 2019)

Why then not stick to probabilities & PGMs

- Problem 1: Probabilistic model development and the derivation of inference algorithms is time-consuming and error-prone.
- Problem 2: Exact (and approximate inference) hard due to normalization)
- Solution to 1
  - Develop Probabilistic Programming (PP) Languages for expressing probabilistic models as computer programs that generate data (i.e. simulators).
  - Derive Universal Inference Engines for these languages that do inference over program traces given observed data (Bayes rule on computer programs).

MOTIVATION: PROBABILISTIC PROGRAMMING

A „Vennified“ Overview on Role of PP

Bayesian/Probabilistic Machine Learning
IFIS Course Intelligent Agents; V8

Deep Learning
V3-V6

Profits from

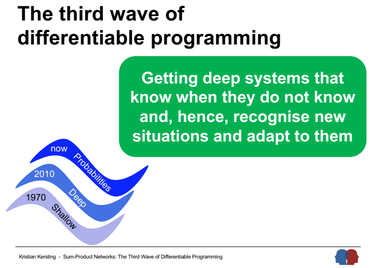
Generative DL V6

exemplifies

Profits from

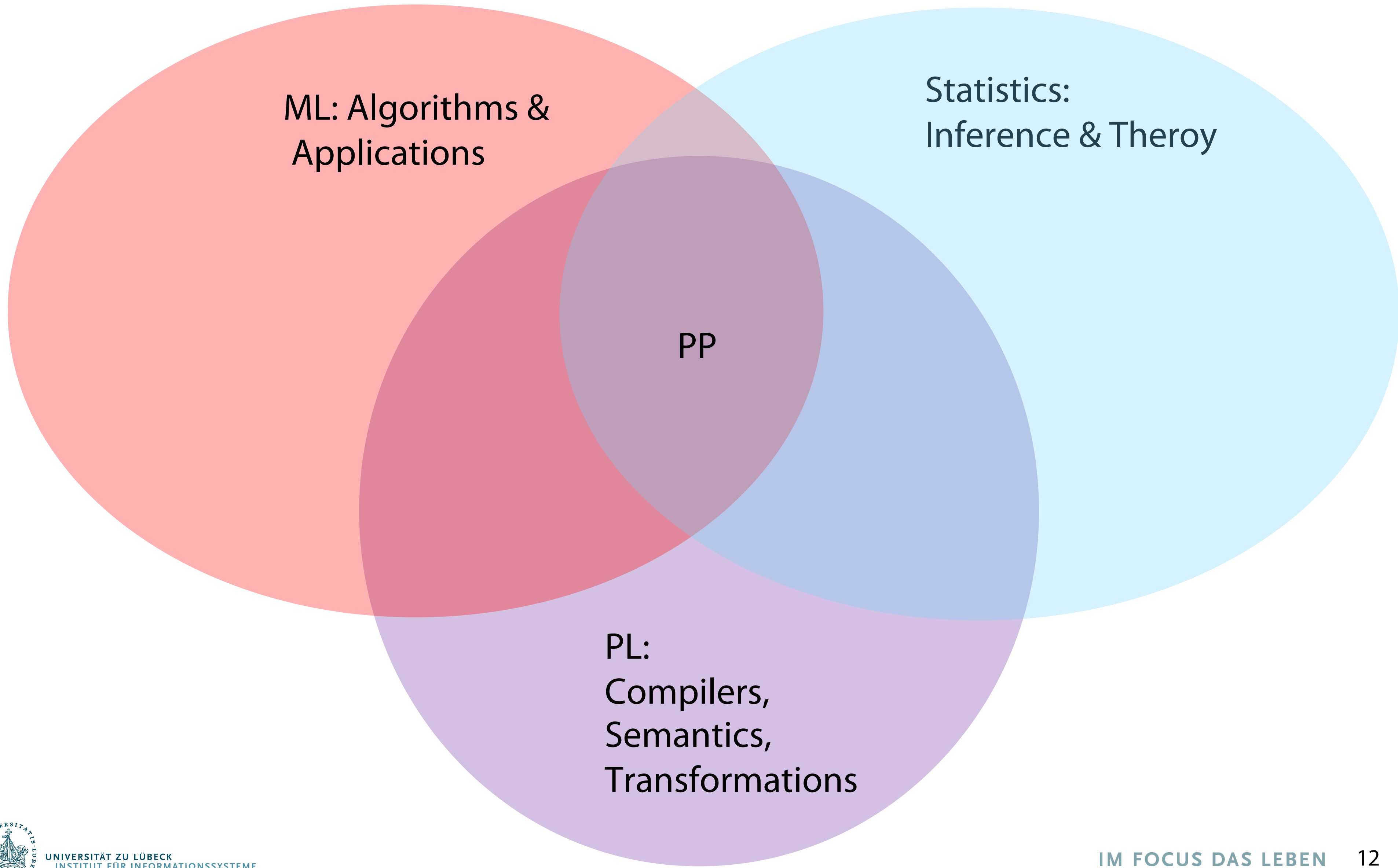
Probabilistic Programming
V8, V9

Gradient Descent/
Automatic Differentiation V3,V7



Efficient representation of probabilities
V10- V12

(Even more Vennification)

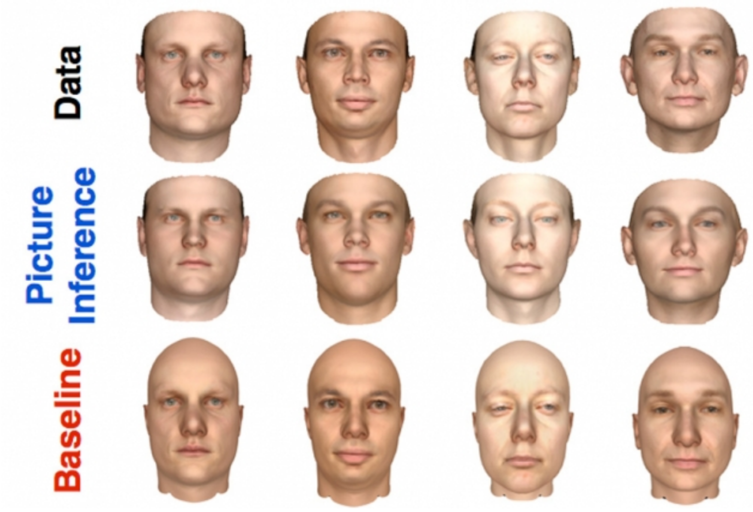


Of course this is also a reason ...

PHYS.ORG

Probabilistic programming does in 50 lines of code what used to take thousands

13 April 2015, by Larry Hardesty



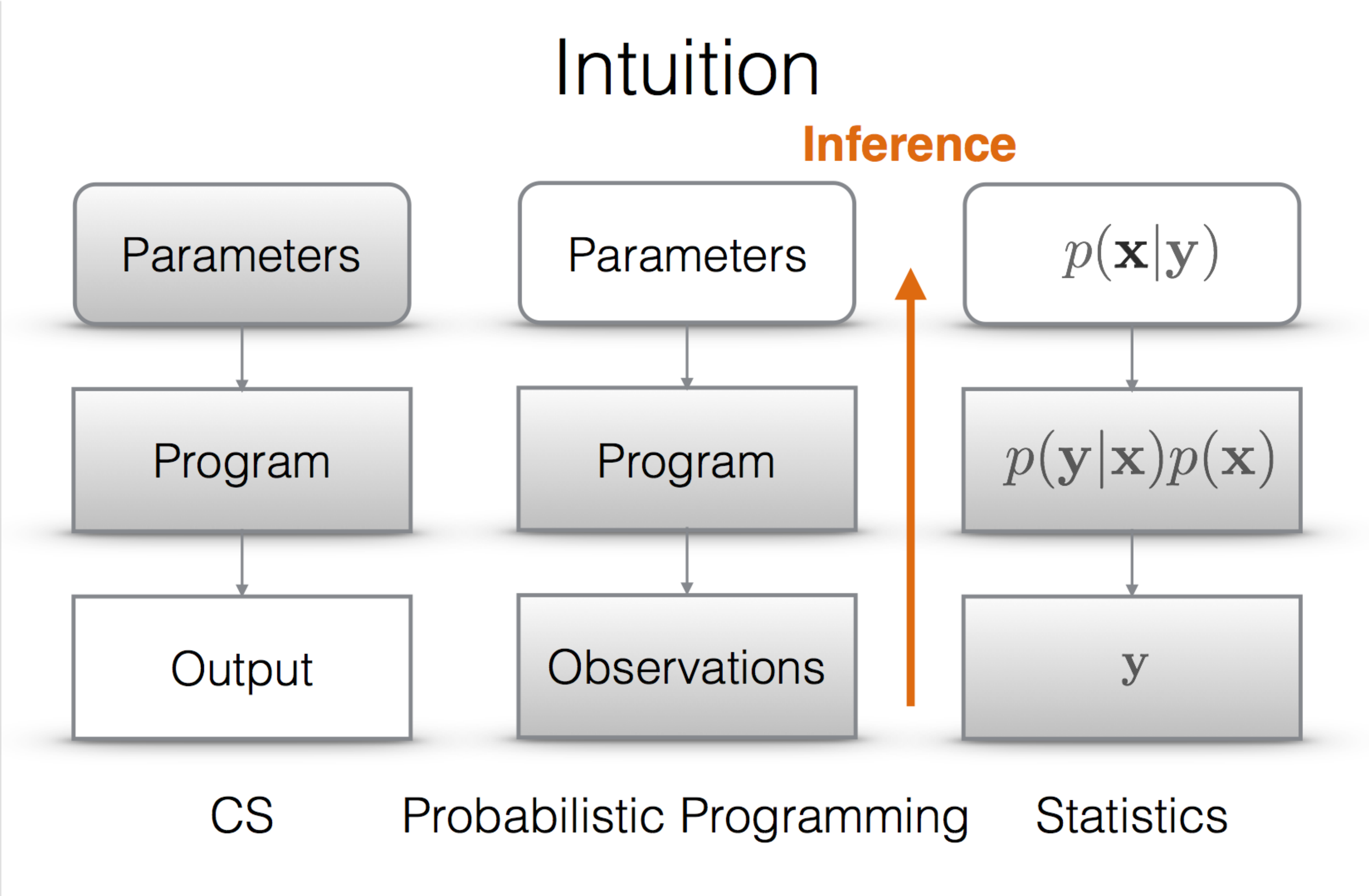
systems with thousands of lines of code.

"This is the first time that we're introducing probabilistic programming in the vision area," says Tejas Kulkarni, an MIT graduate student in brain and cognitive sciences and first author on the new paper. "The whole hope is to write very flexible models, both generative and discriminative models, as short probabilistic code, and then not do anything else. General-purpose inference schemes solve the problems."

By the standards of conventional computer

Comparison

OeOe: Note the „inverted“ use of variables x and y



Ex: F. Wood: Probabilistic Programming, PPAML Summer School, Portland 2016

RUNNING EXAMPLE

A probabilistic program (PP) is any program that can depend on random choices.

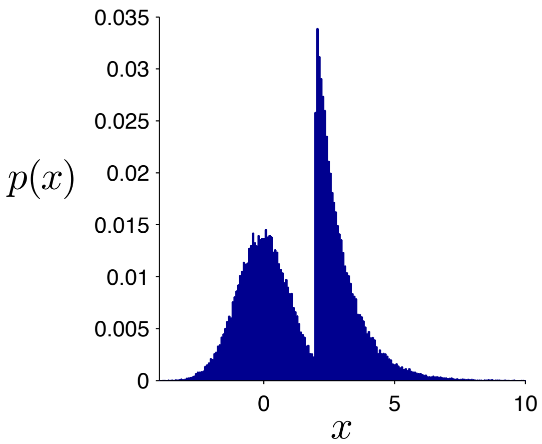
- Can be written in any language that has a random number generator.
- You can specify any computable prior by simply writing down a PP that generates samples.
- A probabilistic program implicitly defines a distribution over its output.

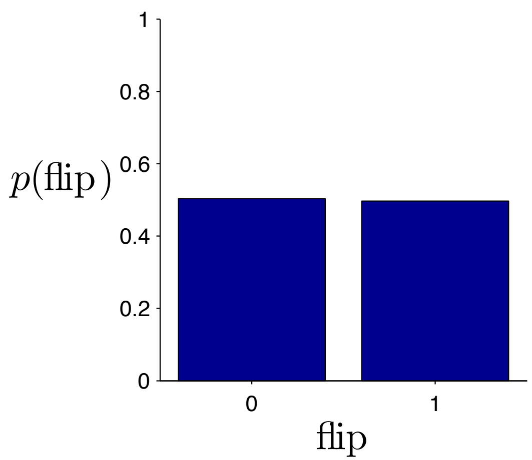
- There are many different PP languages based on different paradigms: imperative, functional, and logical
- Here we illustrate PPs with a lightweight approach for imperative programming based on MATLAB

An Example Probabilistic Program

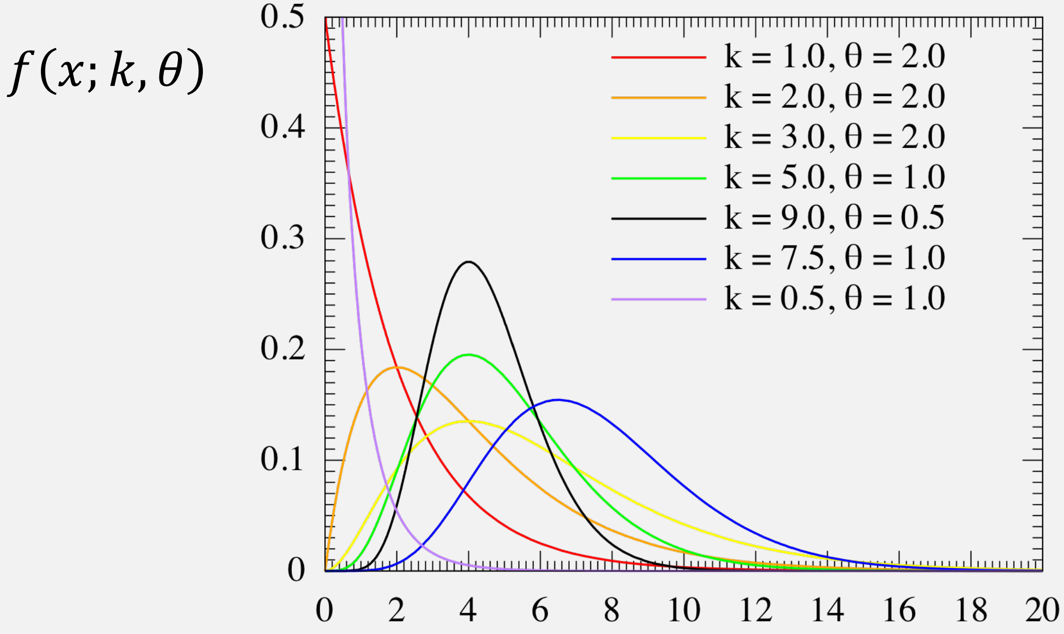
```
flip = rand < 0.5
       % flip is 1 if random number from [0,1] smaller 0,5
 if flip
      x = randg + 2  % Random draw from Gamma(1,1)
else
   x = randn        % Random draw from standard Normal
end
```

Implied distributions over variables





Reminder: Gamma distribution $\Gamma(k, \theta)$



Probability density function of Γ

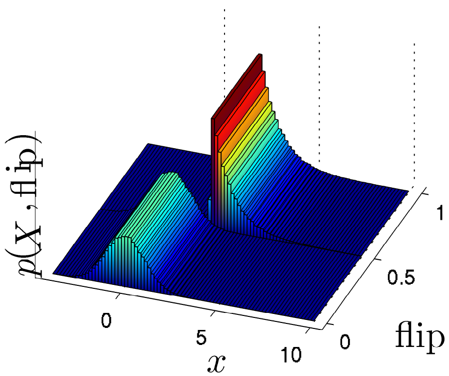
$$f(x; k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}$$

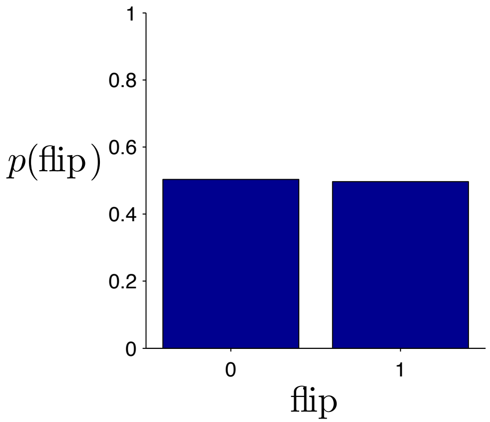
where $\Gamma(k) = \int_0^\infty x^{k-1} e^{-x}\, dx$ is gamma-function (generelization of factorial to complex numbers)

An Example Probabilistic Program

```
flip = rand < 0.5
      % flip is 1 if random number from [0,1] smaller 0,5
 if flip
      x = randg + 2    % Random draw from Gamma(1,1)
else
   x = randn  % Random draw from standard Normal
end
```

Implied distributions over variables





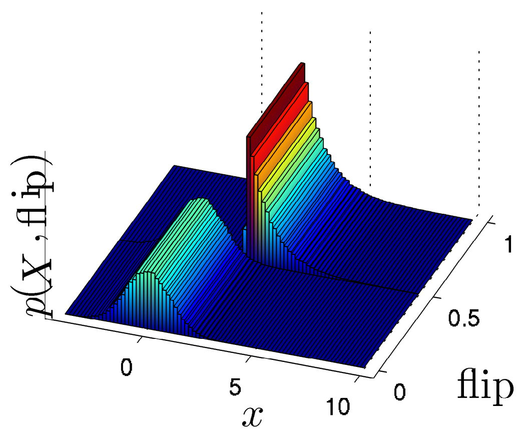
Conditioning

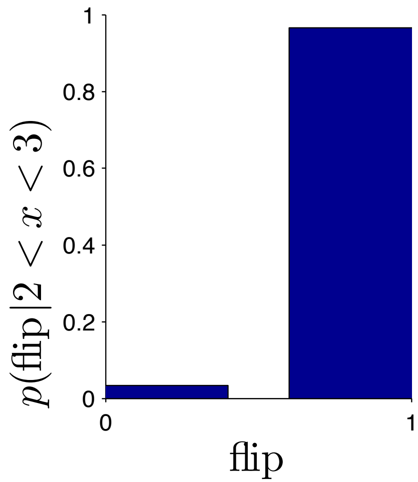
- Once we've defined a prior, what can we do with it?
- PP defines joint distribution $P(D, N, H)$
  - D to be the subset of variables we observe (condition on)
  - H the set of variables we're interested in
  - N the set of variables that we're not interested in, (so we'll marginalize them out).
- We want to know about $P(H|D)$
- Probabilistic Programming
  - Usually refers to doing conditional inference when a probabilistic program specifies your prior.
  - Could also be described as automated inference given a model specified by a generative procedure.

Conditioning with Probabilistic Program

```
flip = rand < 0.5
        % flip is 1 if random number from [0,1] smaller 0,5
 if flip
        x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn  % Random draw from standard Normal
end
```

Implied distributions over variables





Condition/Evidence
$D = 2 < x < 3$

SEMANTICS OF PROBABILISTIC PROGRAMS

Can we develop generic inference for all PPs?

- Rejection sampling
  1. Run the program with fresh source of random numbers
  2. If condition D is true, record H as a sample; else ignore the sample
  3. Repeat

```
flip = rand < 0.5          >> True
if flip
    x = randg + 2
else                        >> 2.7
x = randn
end
```

In case of our example (with $D = 2 < x < 3$) this produces samples over the execution trace, e.g., `(True, 2.7)`

Can we develop generic inference for all PPs?

- Rejection sampling
  1. Run the program with fresh source of random numbers
  2. If condition D is true, record H as a sample; else ignore the sample
  3. Repeat

```
flip = rand < 0.5
if flip          %
    x = randg + 2
else             %
x = randn
end
```

```
>> True
>> 3.2
```

Sample
`(True, 3.2)`
rejected

In case of our example (with $D = 2 < x < 3$) this produces samples over the execution trace, e.g., `(True, 2.7)`

Can we develop generic inference for all PPs?

- Rejection sampling
  1. Run the program with fresh source of random numbers
  2. If condition D is true, record H as a sample; else ignore the sample
  3. Repeat

```
flip = rand < 0.5
if flip
    x = randg + 2
else
x = randn
end
```

```
>> True

>> 2.1
```

In case of our example (with $D = 2 < x < 3$) this produces samples over the execution trace, e.g.,
`(True, 2.7) (True, 2.1)`

Can we develop generic inference for all PPs?

- Rejection sampling
  1. Run the program with fresh source of random numbers
  2. If condition D is true, record H as a sample; else ignore the sample
  3. Repeat

```
flip = rand < 0.5          >> False
if flip
    x = randg + 2
else
x = randn
end                        >> -1.3
```

Sample
`(False, -1.3)`
rejected

In case of our example (with $D = 2 < x < 3$) this produces samples over the execution trace, e.g.,
`(True, 2.7) (True, 2.1)`

Can we develop generic inference for all PPs?

- Rejection sampling
  1. Run the program with fresh source of random numbers
  2. If condition D is true, record H as a sample; else ignore the sample
  3. Repeat

```
flip = rand < 0.5
if flip
    x = randg + 2
else
x = randn
end
```

```
>> False


>> 2.3
```

In case of our example (with $D = 2 < x < 3$) this produces samples over the execution trace, e.g.,
`(True, 2.7) (True, 2.1) (False, 2.3),...`

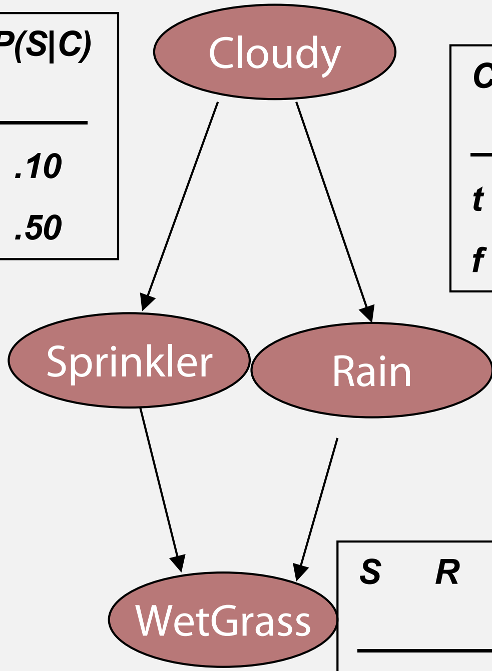
Of course we can do better

- Rejection sampling (as the simplest form of stochastic simulation) is inefficient
  - Rejects to many samples because
  - Probability $P(D)$ is small (drops exponentially with increasing numbers of evidence variables)
- Better is likelihood weighting:
  - produce only samples consistent with evidence, the probabilities of which are incorporated as weights
- Another well-known stochastic simulation is an instance of Markov-Chain-Monte-Carlo (MCMC) simulation:

  Metropolis-Hastings (MH)

Reminder: Likelihood Weighting for Bayes Nets

- P(Rain|Sprinkler=true, WetGrass = true) = ?
- Sampling
  - w = 1.0 (weight initialized)
  - Sample $\boldsymbol{P}(Cloudy) = (0.5{,}0.5) \Rightarrow true$
  - Sprinkler is an evidence variable with value true
    w ← w * P(Sprinkler=true | Cloudy = true) = 0.1
  - Sample P(Rain|Cloudy=true)=(0.8,0.2) => true
  - WetGrass is an evidence variable with value true
    w ←w * P(WetGrass=true |Sprinkler=true, Rain = true) = 0.099
  - [true, true, true, true] with weight 0.099
- Estimating
  - Accumulating weights to either Rain=true or Rain=false
  - Normalize (= divide by sum of weights)



P(C)=.5

C	P(S|C)
t	.10
f	.50

C	P(R|C)
t	.80
f	.20

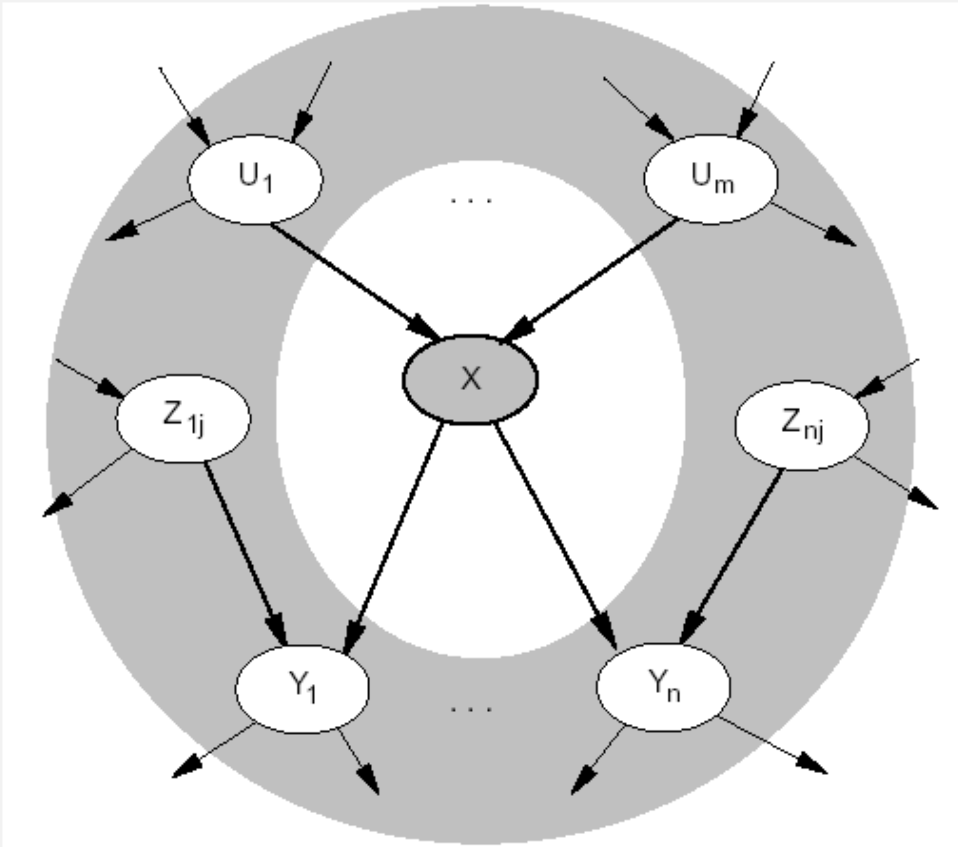
S	R	P(W|S,R)
t	t	.99
t	f	.90
f	t	.90
f	f	.00

Reminder: Markov Chain Monte Carlo (MCMC)

- Let's think of the network as being in a particular current state specifying a value for every variable
- MCMC generates each event by making a random change to the preceding event
- The next state is generated by randomly sampling a value for one of the non-evidence variables X_i, conditioned on the current values of the variables in the Markov blanket of X_i
- Note: Likelihood Weighting only takes into account the evidences of the parents. (Problematic if evidence on leaves).

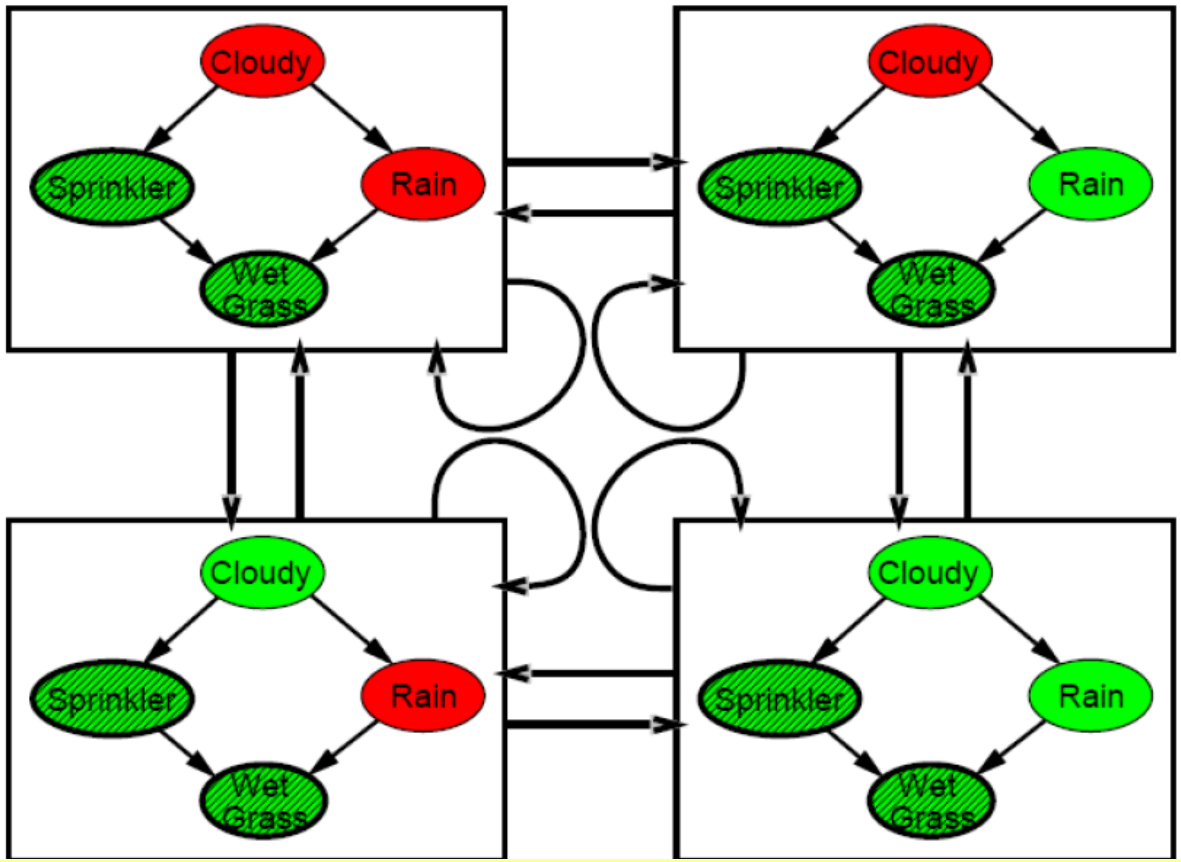
Reminder: Markov Blanket

- Markov blanket: Parents + children + children's parents
- Node is conditionally independent of all other nodes in network, given its Markov Blanket



Reminder: MCMC

With $Sprinkler = true, WetGrass = true$, there are four states:

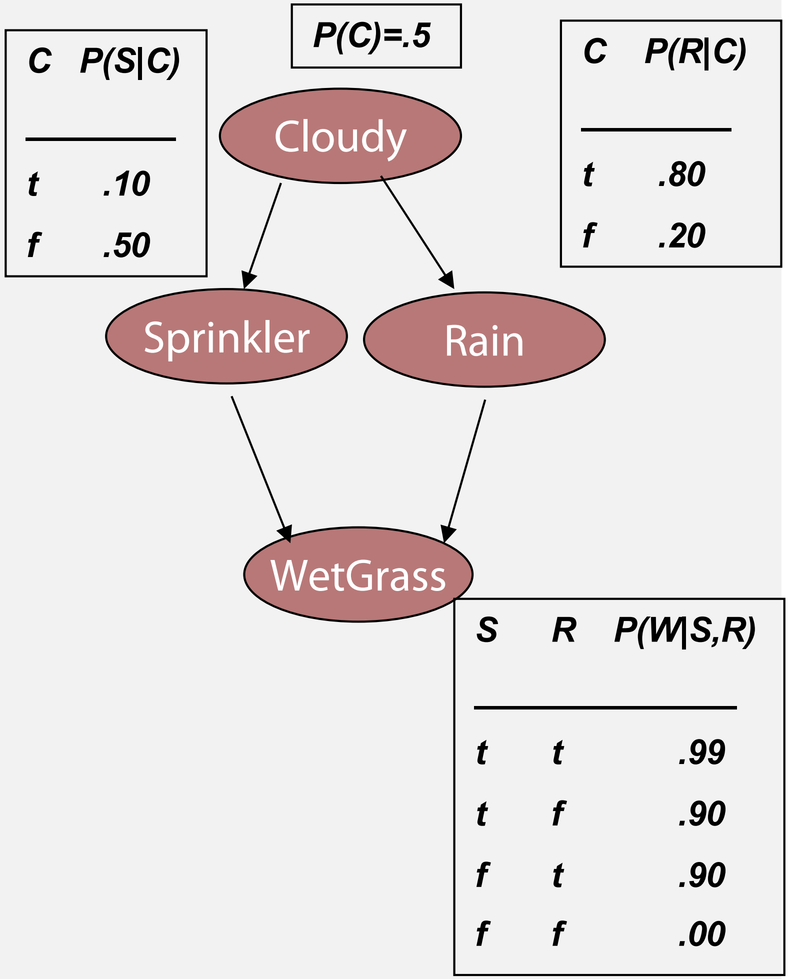


Arrows describe transition probabilities; leads to a (the Markov) chain of states

Wander about for a while, average what you see

Reminder: Markov Chain Monte Carlo: Example

- $P(Rain|Sprinkler = true, WetGrass = true) = ?$
- States [Cloudy,Sprinkler,Rain,WetGrass]
- Initial state is [true, true, false, true]
- The following steps are executed repeatedly:
  - Cloudy ~ P(Cloudy|Sprinkler= true, Rain=false) => Cloudy = false
    State update: [false, true, false, true]
  - Rain ~ P(Rain|Cloudy=false,Sprinkler=true, WetGrass=true) => Rain = true
    State update: [false, true, true, true]
- After all the iterations, let's say the process visited 20 states where Rain is true and 60 states where Rain is false then the answer of the query is NORMALIZE((20,60))=(0.25,0.75)



Example: Metropolis-Hastings

1. Start with a trace
2. Change one random decision, discarding subsequent decisions
3. Sample subsequent decisions
4. Accept with appropriate MCMC acceptance probability

1. (True, 2.3)
2. (False,)
3. (False, -0,9)
4. Reject, does not satisfy observation

Example: Metropolis-Hastings

1. Start with at race
2. Change one random decision, discarding subsequent decisions
3. Sample subsequent decisions
4. Accept with appropriate MCMC acceptance probability

1. (True, 2.3)
2. (True, 2.9)
3. Nothing to do
4. Accept, maybe

Semantics of PP via MH - Notation

- Evaluating a program results in a sequence of random choices
  - $x_1 \sim p_{t_1}(x_1)$
  - $x_2 \sim p_{t_2}(\, x_2 \mid x_1 \,)$
  - $x_3 \sim p_{t_3}(\, x_3 \mid x_2, x_1 \,)$
  - …
  - $k \sim p_{t_k}(\, x_k \mid x_{k-1}, \dots, x_1 \,)$
    (for execution trace $\mathrm{x} = \; x_1, \dots x_{k-1}$)
- Density/Probability of a particular evaluation is then
  - $p(x_1, \dots, x_n) = \; \prod_{k=1}^{K} p_{t_k}(x_k \mid x_{k-1}, \dots, x_1 \,)$
- Then perform MH over the execution traces x

MH over traces

- Select a random decision in the execution trace x
  - E.g. x_k
- Propose a new value
  - E.g. $x'_k \sim K_{t_k}(x'_k \mid x_k)$ (K_{t_k} is called proposal distribution)
- Run the program to determine all subsequent choices $(x'_l: l > k)$, reusing current choices where possible
- Propose moving from the state $x_1, \ldots, x_K$ to $(\underbrace{x_1, \ldots, x_{k-1}}_{\text{old choices}}, \underbrace{x'_k, \ldots, x'_{K'}}_{\text{new choices}})$
- Accept the change with the appropriate MH acceptance probability $(= \min\{\alpha, 1\})$
  - $\alpha = \dfrac{K_{t_k}\left(x_k | x'_k\right) \prod_{i=k}^{K'} p_{t'_i}(x'_i | x_1, \ldots, x_{k-1}, x'_k, \ldots, x'_{i-1})}{K_{t_k}\left(x'_k | x_k\right) \prod_{i=k}^{K} p_{t_i}(x_i | x_1, \ldots, x_{k-1}, x_k, \ldots, x_{i-1})}$

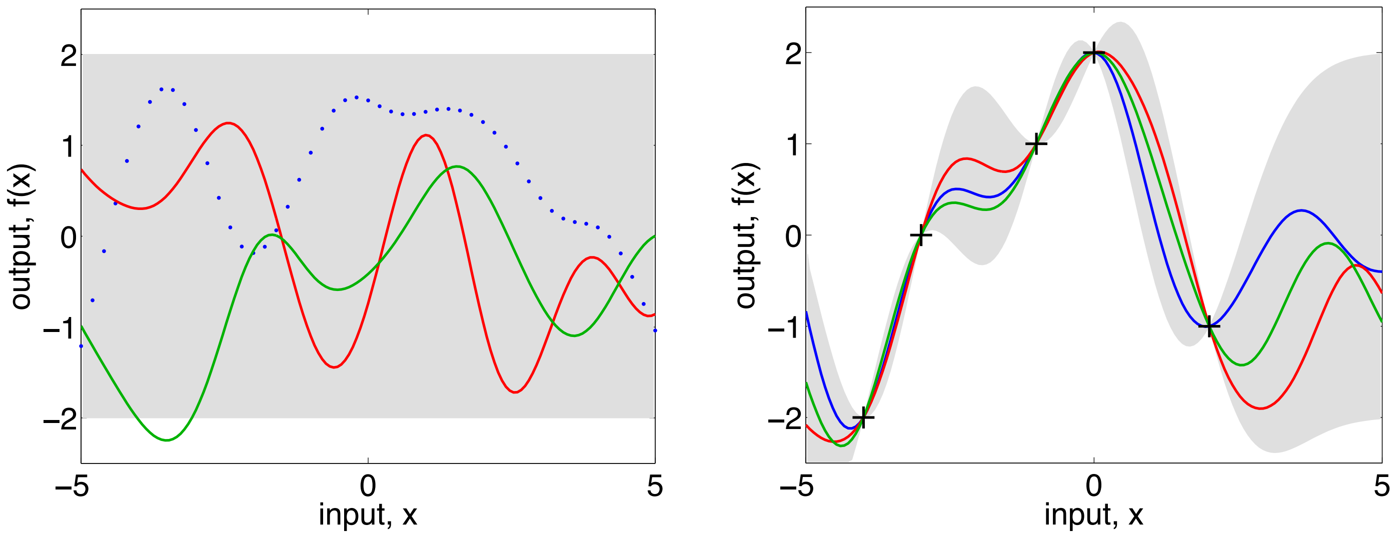
NONPARAMETRICS

Works also for non-parametric models

- If we can sample from the prior of a nonparametric model using finite resources with probability 1, then we can perform inference automatically using the techniques described thus far
- We can sample from a number of nonparametric processes/models with finite resources (with probability 1) using a variety of techniques
  - Gaussian processes via marginalisation <=
  - Dirichlet processes via stick breaking
  - Indian Buffet processes via urn schemes

Tackling non-parametric models

- Non-parametric models: Allow distributions over arbitrary functions to learn a target function



Prior

Posterior

- Typical Example: Gaussian Process (GP)

Reminder: Multivariate Gaussians

Write r.v. $\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{pmatrix}$ Then define $X \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to mean

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{m/2} \| \boldsymbol{\Sigma} \|^{1/2}} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \, \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Co-variance matrix

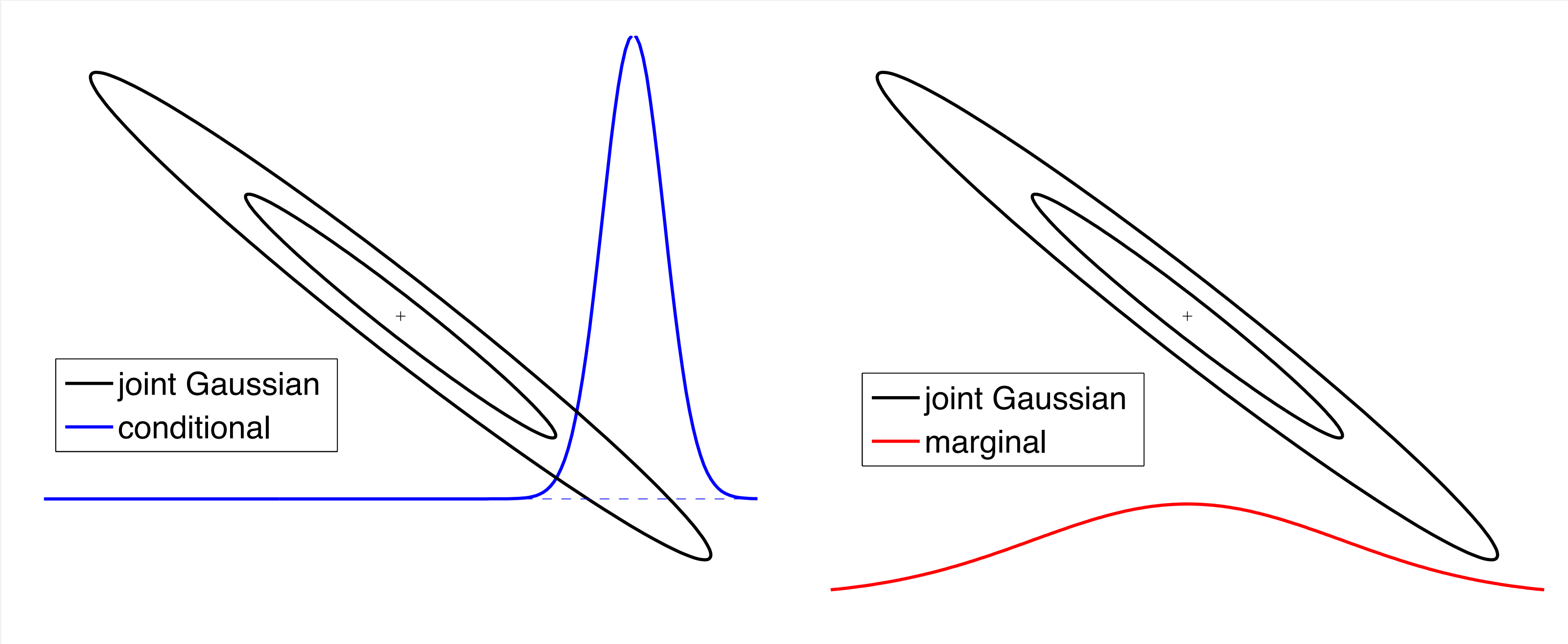
where the Gaussian's parameters have…

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_m \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma^2{}_1 & \sigma_{12} & \cdots & \sigma_{1m} \\ \sigma_{12} & \sigma^2{}_2 & \cdots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1m} & \sigma_{2m} & \cdots & \sigma^2{}_m \end{pmatrix}$$

where $\sigma_{ij} = Cov(X_i, X_j) = E[(X_i - E(X_i)) \cdot (X_j - E(X_j))]$

One can show: E[X] = μ and Cov[X] = Σ.

Reminder: Gaussians



The class of Gaussians is invariant both under conditionalizing and marginalizing

Tackling non-parametric models

- Gaussian Process (GPs)
  - A Gaussian process is a collection of random variables, any finite number of which have (consistent) Gaussian distributions
  - GPs generalize of multivariate Gaussians to infinitely many variables (and infinitely long vector = function)
- A Gaussian distribution $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is specified by mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
- A GP is fully specified by a mean function $\mu(x)$ and a covariance function $k(x, x')$

Doing the sampling finitely

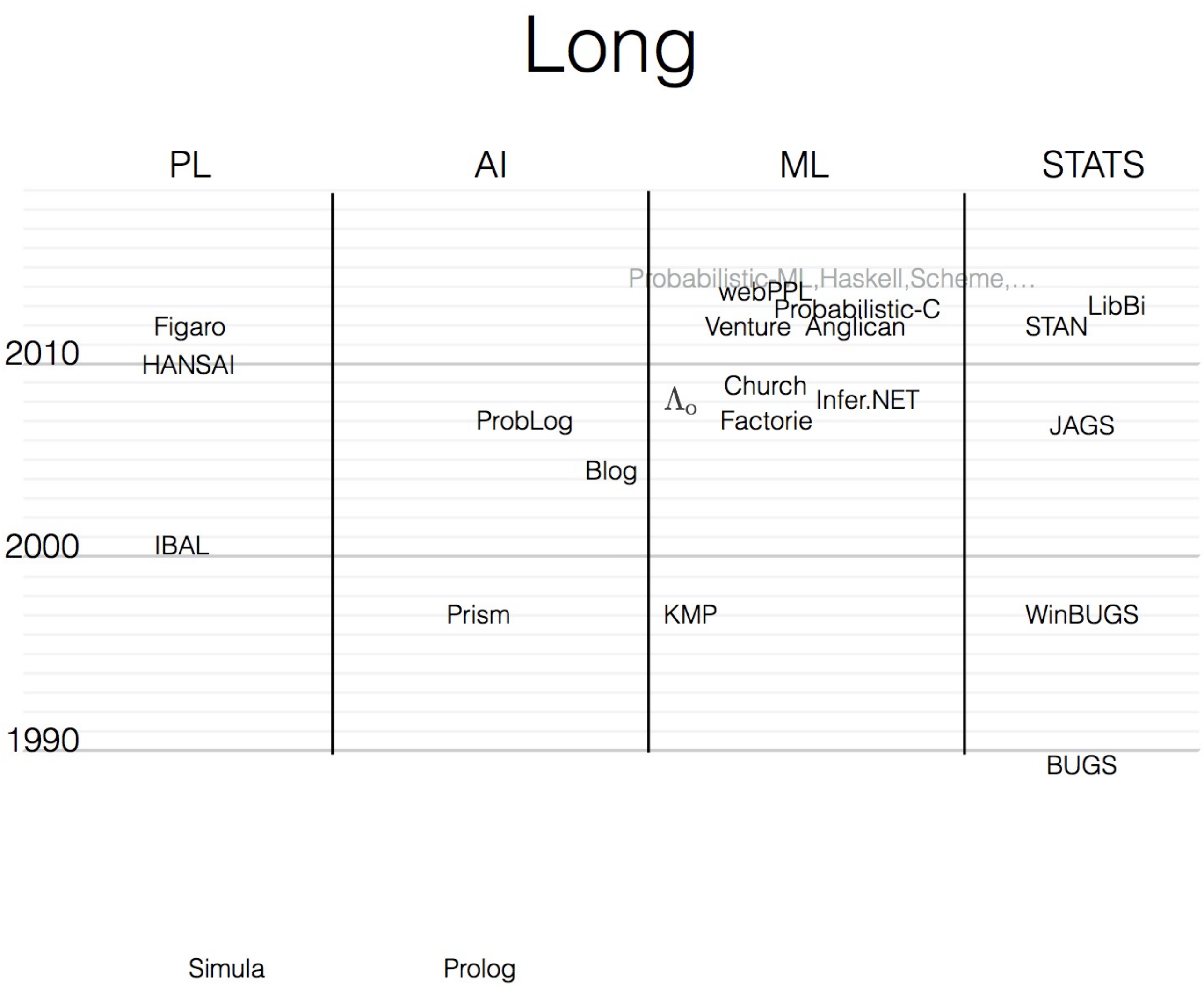
- Marginalization works here too, so can marginalize on all variables except for finite vector of RVs $\boldsymbol{x}$
  - $p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y})\mathrm{d}\boldsymbol{y}$ (for arbitrary continous variables)
  - For Gaussians
  - If $p(\boldsymbol{x}, \boldsymbol{y}) = N\left(\begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{pmatrix} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^\top & \boldsymbol{C} \end{pmatrix}\right)$, then $p(\boldsymbol{x}) = N(\boldsymbol{a}, \boldsymbol{A})$

Advanced Automatic Inference

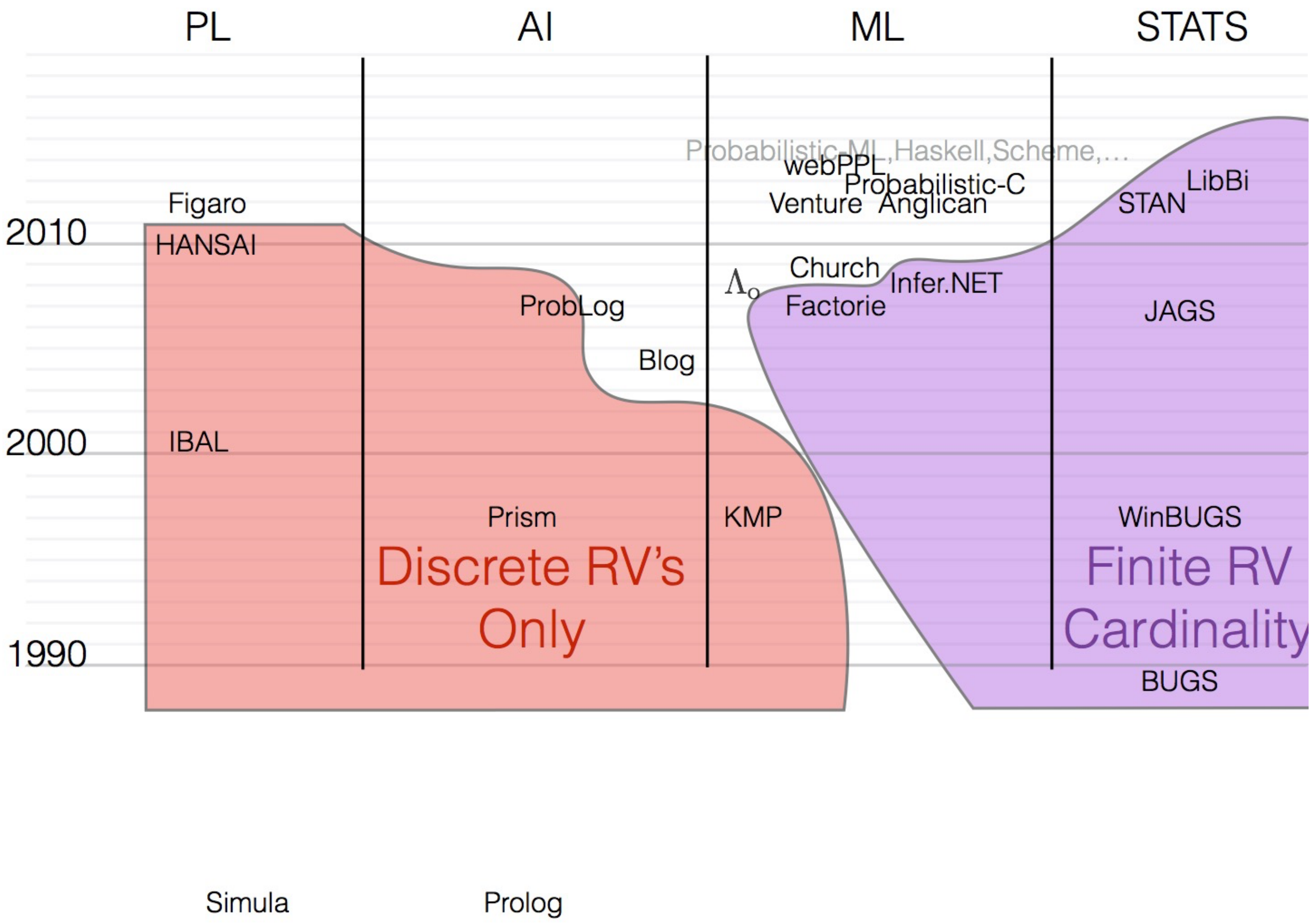
- Now that we have separated inference and model design, can use any inference algorithm.
- Free to develop inference algorithms independently of specific models.
- Once graphical models identified as a general class, many model-agnostic inference methods:
  - Belief Propagation
  - Pseudo-likelihood
  - Mean-field Variational
  - MCMC
- What generic inference algorithms can we implement for more expressive generative models?

LANDSCAPE OF PROBABILISTIC PROGRAMMING LANGUAGES

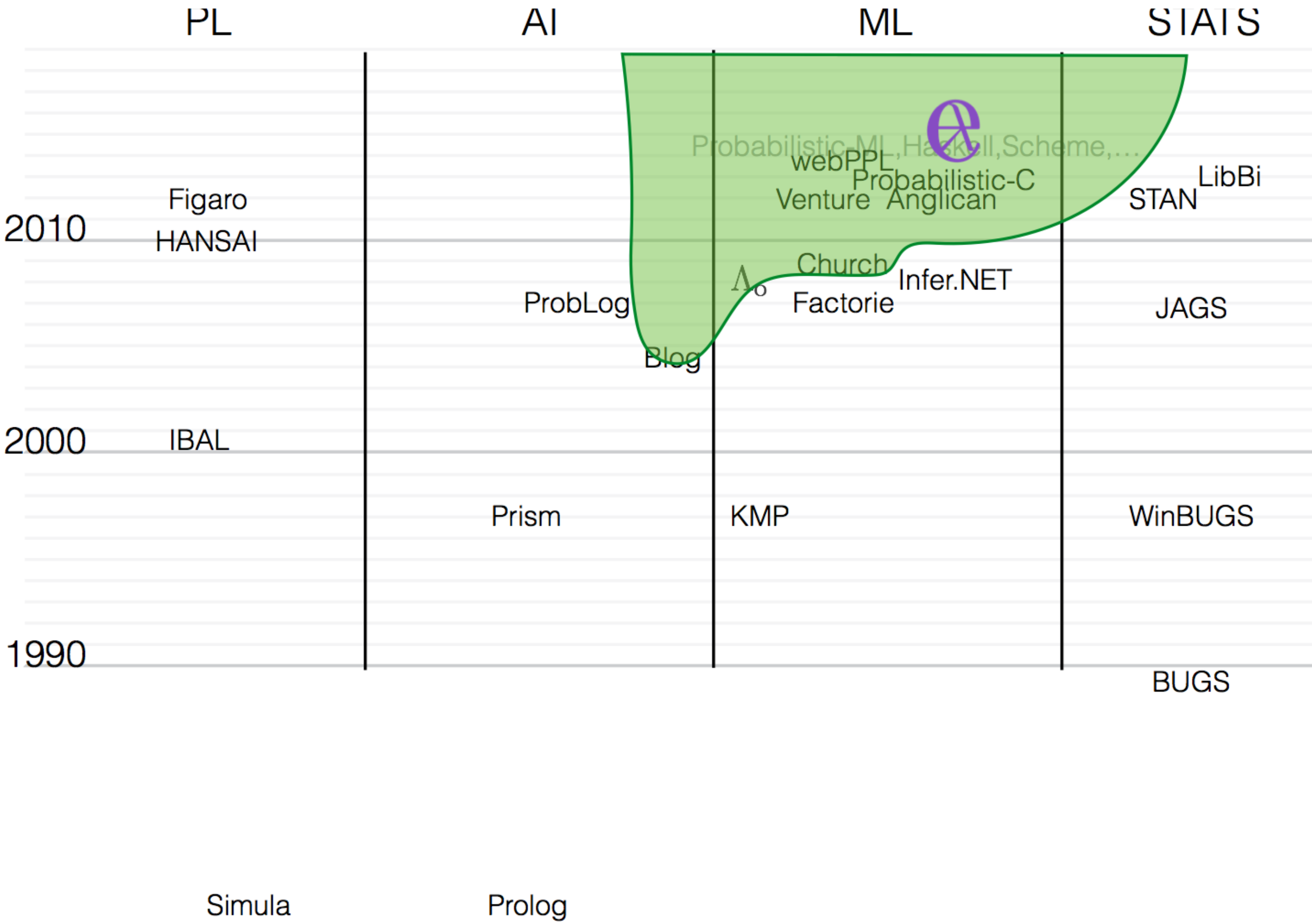
History of PP with Programming Languages



First-Order PP languages



Higher-Order PP Languages

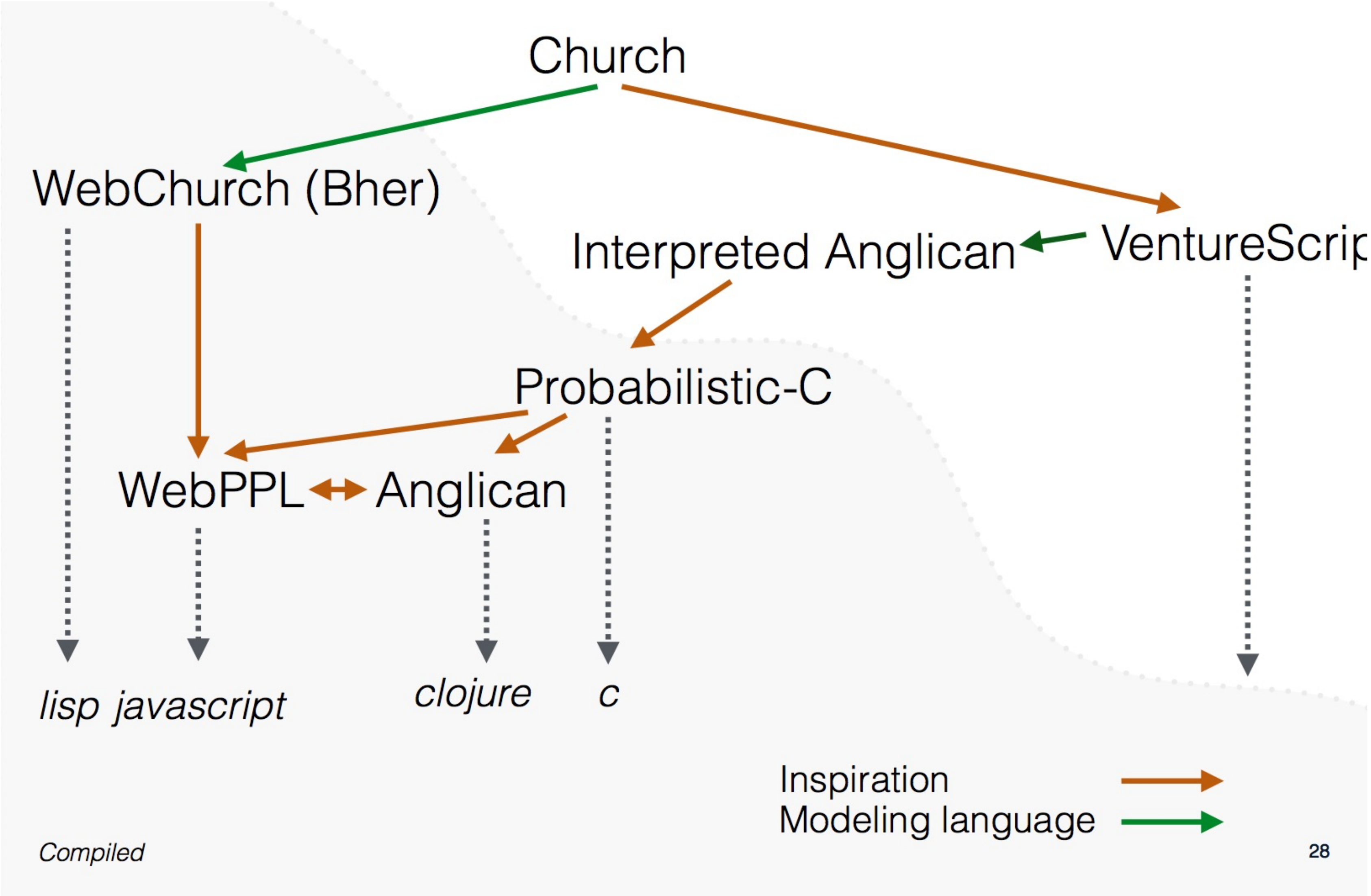


The Church Family

- Lisp like constructs extended with two main functions
  - Sample
  - Observe

- For a book-lengthy treatment see (Van de Ment et al 2018)
  - In particular, describes a formal grammar, astonishingly simple grammar

The church family

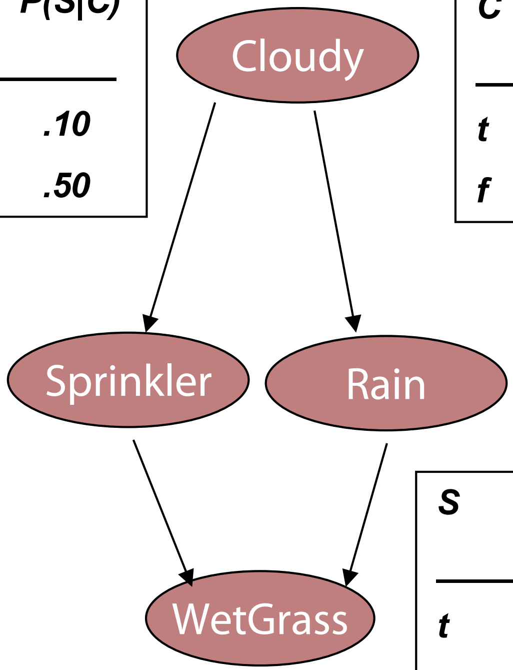


Example: Bayes Net in Anglican

P(C)=.5

C	P(S|C)
t	.10
f	.50

C	P(R|C)
t	.80
f	.20



S	R	P(W|S,R)
t	t	.99
t	f	.90
f	t	.90
f	f	.00

```
(defquery sprinkler-bayes-net [sprinkler wet-grass]
  (let [is-cloudy (sample (flip 0.5))

        is-raining (cond (= is-cloudy true )
                         (sample (flip 0.8))
                         (= is-cloudy false)
                         (sample (flip 0.2)))
        sprinkler-dist (cond (= is-cloudy true)
                             (flip 0.1)
                             (= is-cloudy false)
                             (flip 0.5))
        wet-grass-dist (cond
                         (and (= sprinkler true)
                              (= is-raining true))
                         (flip 0.99)
                         (and (= sprinkler false)
                              (= is-raining false))
                         (flip 0.0)
                         (or  (= sprinkler true)
                              (= is-raining true))
                         (flip 0.9))]
    (observe sprinkler-dist sprinkler)
    (observe wet-grass-dist wet-grass)

    is-raining))
```

Example Application: CAPTCHA Breaking

Observation

Posterior Samples

Generative Model

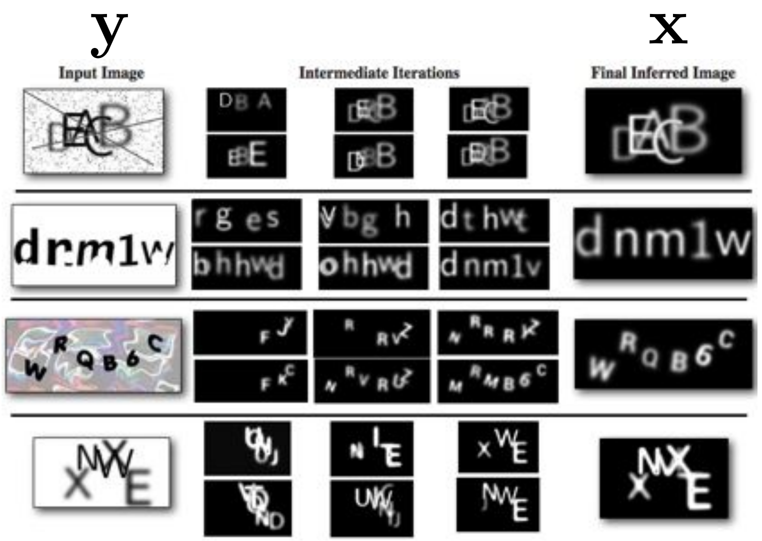
```
(defquery captcha
 [image num-chars tol]
 (let [[w h] (size image)
       ;; sample random characters
       num-chars (sample
                  (poisson num-chars))
       chars (repeatedly
               num-chars sample-char)]
  ;; compare rendering to true image
  (map (fn [y z]
         (observe (normal z tol) y))
       (reduce-dim image)
       (reduce-dim (render chars w h)))
  ;; predict captcha text
  {:text
   (map :symbol (sort-by :x chars))}))
```

$\mathbf{x}$	$\mathbf{y}$
text	image

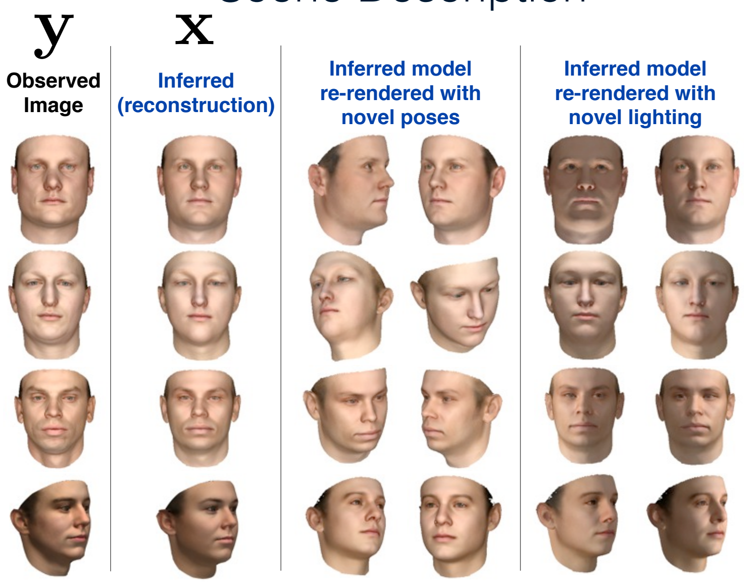
OeOe: Note the „inverted“ use of variables x and y

Examle Application: Scene interpretation

Captcha Solving



Scene Description



$\mathbf{x}$	$\mathbf{y}$
scene description	image

(Masinghka et al 2013)

(Kulkarni et al. 2015) et al 2013)

Next week

- „Probabilistic Programming“ is sometimes used in narrow sense for probabilistically enhanced imperative or functional languages (Gordon et al. 14)

- We use it in a broader sense to include also probabilistic logic programs – the topic of next week

Uhhh, a lecture with a hopefully useful

APPENDIX

Probability theory basics reminder

Random variable (RV)

- possible worlds defined by assignment of values to random variables.
- Boolean random variables
  e.g., Cavity (do I have a cavity?).
  Domain is < true , false >
- Discrete random variables
  e.g., possible value of Weather is one of
  < sunny, rainy, cloudy, snow >
- Domain values must be exhaustive and mutually exclusive
- Elementary propositions are constructed by assignment of a value to a random variable: e.g.,
  - Cavity = false (abbreviated as ¬cavity)
  - Cavity = true (abbreviated as cavity)
- (Complex) propositions formed from elementary propositions and standard logical connectives, e.g., Weather = sunny ∨ Cavity = false

Probabilities

- Axioms (for propositions $a, b, \top = (a \vee \neg a)$, and $\bot = \neg \top$):
  - $0 \leq P(a) \leq 1$; $P(\top) = 1$; $P(\bot) = 0$
  - $(P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
- Joint probability distribution of $\mathbf{X} = \{X_1, \ldots, X_n\}$
  - $\boldsymbol{P}(X_1, \ldots, X_n)$
  - gives the probability of every atomic event on $\boldsymbol{X}$
- Conditional probability
  $P(a \mid b) = P(a \wedge b) / P(b)$ if $P(b) > 0$
- Chain rule

$$\boldsymbol{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \boldsymbol{P}(X_i | X_1, \ldots, X_{i-1})$$

- Marginalization: $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y, z)$
- Conditioning on Z:
  - $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y|z)\boldsymbol{P}(z)$ (discrete)
  - $\boldsymbol{P}(Y) = \int \boldsymbol{P}(Y|z)\boldsymbol{P}(z)dz$ (continuous)
    $= \mathbb{E}_{\boldsymbol{z} \sim P(z)}$P(Y|z) (expected value notation)
- Bayes' Rule

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

Color Convention in this Course

- Formulae, when occurring inline
- Newly introduced terminology and definitions
- Important results (observations, theorems) as well as emphasizing some aspects
- Examples are given with standard orange with possibly light orange frame
- Comments and notes in nearly opaque post-it
- Algorithms and program code
- Reminders (in the grey fog of your memory)

Today's lecture is based on the following

- Mainly
  - D. Duvenaud/J. Loyd: Introduction toProbabilistic Programming. Talk given at Computational and Biological Learning Lab, University of Cambridge, March 2013 (https://jamesrobertlloyd.com/talks)
- A little bit of
  - Zoubin Ghahramani: Probabilistic Machine Learning and AI, Microsoft AI Summer School Cambridge 2017 http://mlss.tuebingen.mpg.de/2017/speaker_slides/Zoubin1.pdf
  - F. Wood: Probabilistic Programming, PPAML Summer School, Portland 2016, link

References

- Gordon, Henzinger, Nori, and Rajamani
  "Probabilistic programming." In Proceedings of On The Future of Software Engineering (2014).
- Mansinghka,, Kulkarni, Perov, and Tenenbaum
  "Approximate Bayesian image interpretation using generative probabilistic graphics programs." NIPS (2013).
- J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. An Introduction to Probabilistic Programming. arXiv e-prints, page arXiv:1809.10756, Sept. 2018.
- T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka. Picture: A probabilistic programming language for scene perception. In Proceedings of CVPR 2015, 2015, pages 4390–4399.
- D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques - Adaptive Com- putation and Machine Learning. The MIT Press, 2009.