The wondrous world of credal and deep probabilistic circuits

SIPTA Seminar

Erik Quaeghebeur

Eindhoven University of Technology

22 January 2025, 15:00 CET

$$1 + (2 \times 3)$$

Overview

Circuits, conceptually

Credal PCs

Probabilistic integral circuits

Overview

Circuits, conceptually

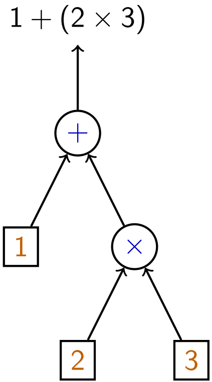
- From expression trees to circuits
- Probabilistic circuits

Credal PCs

Probabilistic integral circuits

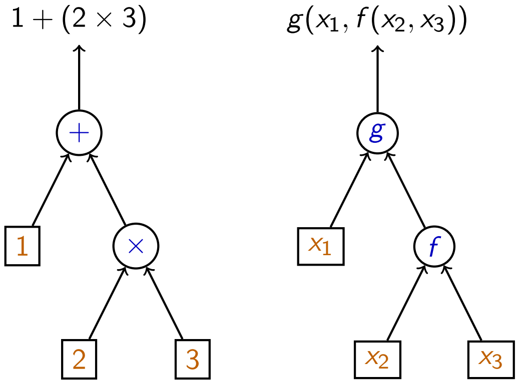
Expression trees

- Graphical representation for *expressions*
- Edges show *composition* order
- Nodes show symbols for values and functions



Expression trees

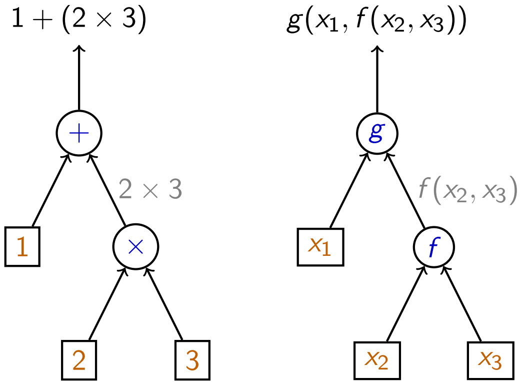
- Graphical representation for *expressions*
- Edges show *composition* order
- Nodes show symbols for values and functions



Expression trees

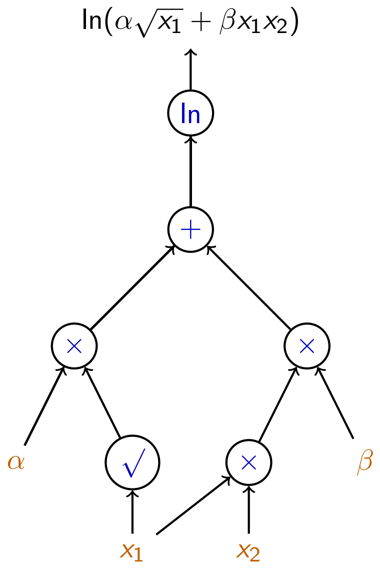
- Graphical representation for *expressions*
- Edges show *composition* order
- Nodes show symbols for values and functions

- Edges 'carry' partial *expressions*



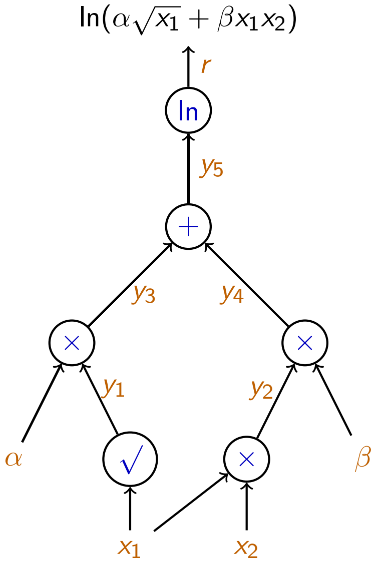
Computation graphs

- Graphical representation
  for *composite function computations*
  Need not be a tree; often an SDAG
- Edges show *computation* order
- Nodes show symbols
  for component functions



Computation graphs

- Graphical representation
  for *composite function computations*
  Need not be a tree; often an SDAG
- Edges show *computation* order
- Nodes show symbols
  for component functions
- Edges 'carry' input/computed *values*
- Usually distinction made between
  parameters and variables

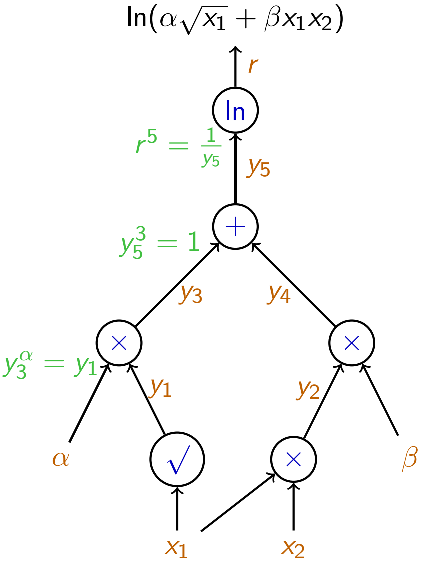


Computation graphs

- Graphical representation
  for *composite function computations*
  Need not be a tree; often an SDAG
- Edges show *computation* order
- Nodes show symbols
  for component functions

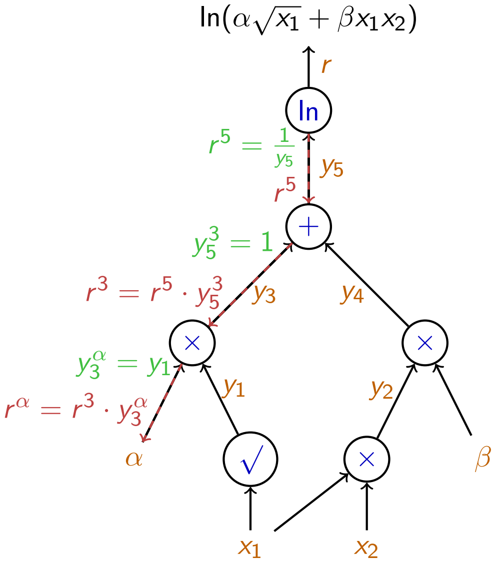
- Edges 'carry' input/computed *values*
- Usually distinction made between
  parameters and variables

- Also used for gradient backpropagation



Computation graphs

- Graphical representation
  for *composite function computations*
  Need not be a tree; often an SDAG
- Edges show *computation* order
- Nodes show symbols
  for component functions
- Edges 'carry' input/computed *values*
- Usually distinction made between
  parameters and variables
- Also used for gradient backpropagation

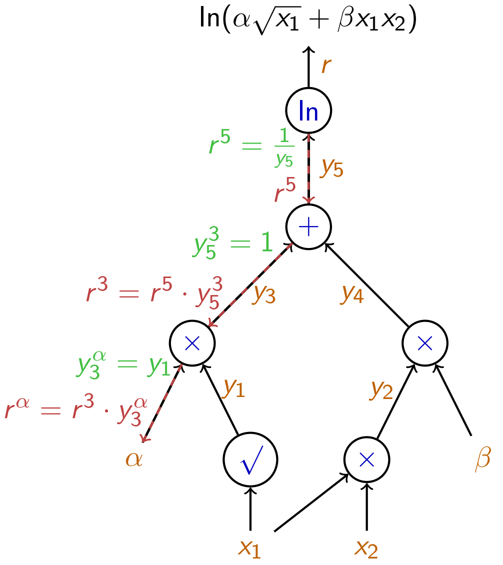


Computation graphs

- Graphical representation
  for *composite function computations*
  Need not be a tree; often an SDAG
- Edges show *computation* order
- Nodes show symbols
  for component functions

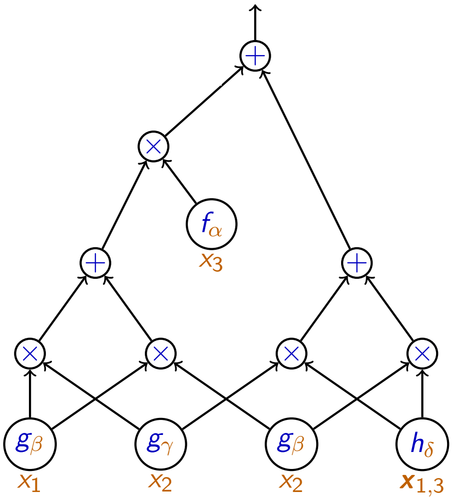
- Edges 'carry' input/computed *values*
- Usually distinction made between
  parameters and variables

- Also used for gradient backpropagation
- 'Values' can be multidimensional arrays



Circuits

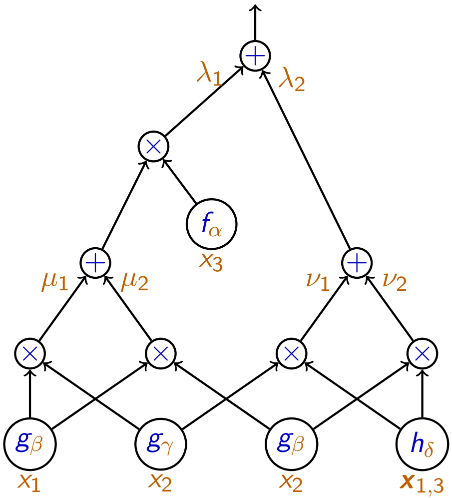
- Compact representation of computation graph
- Input variables shown to indicate input function arguments
- Parameters are 'attached' to functions



Circuits

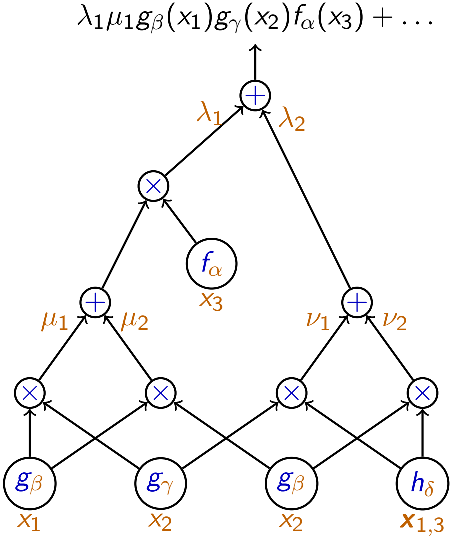
- Compact representation of computation graph
- Input variables shown to indicate input function arguments
- Parameters are 'attached' to functions

- For internal nodes classically only products and (weighted) sums
- Recently also other functions



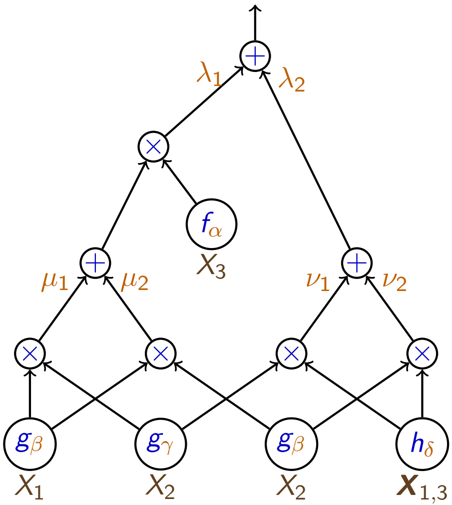
Circuits

- Compact representation of computation graph
- Input variables shown to indicate input function arguments
- Parameters are 'attached' to functions
- For internal nodes classically only products and (weighted) sums
- Recently also other functions
- Polynomial in input function values



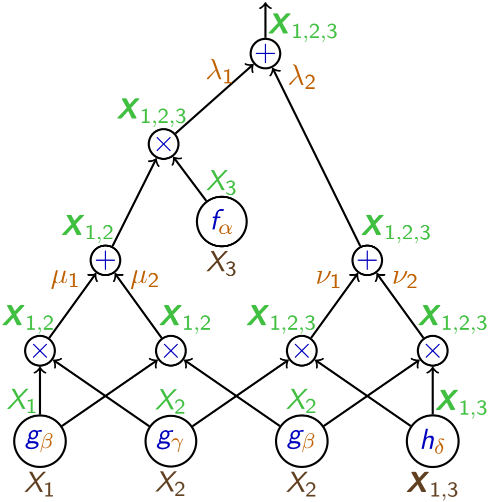
Probabilistic circuits (PCs)

- Circuits where
  - *Random* variables considered
  - input functions are *probability mass functions/densities*
  - weighted sums are *convex mixtures*
  - products represent *independence*
- Output is 'likelihood' for represented joint probability distribution (normalization not always imposed)



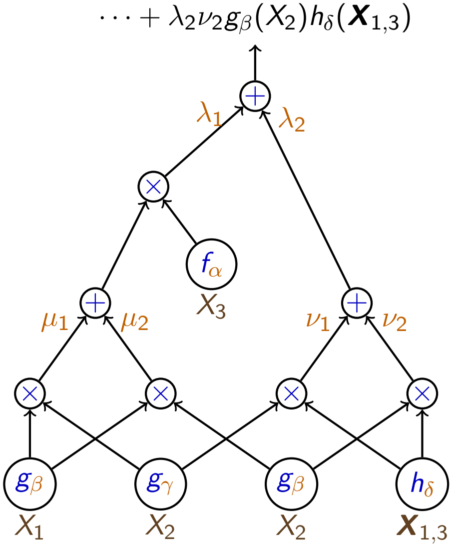
Probabilistic circuits (PCs)

- Circuits where
  - *Random* variables considered
  - input functions are *probability mass functions/densities*
  - weighted sums are *convex mixtures*
  - products represent *independence*
- Output is 'likelihood' for represented joint probability distribution (normalization not always imposed)
- Scope of functions
- Important properties: *smoothness, decomposability*

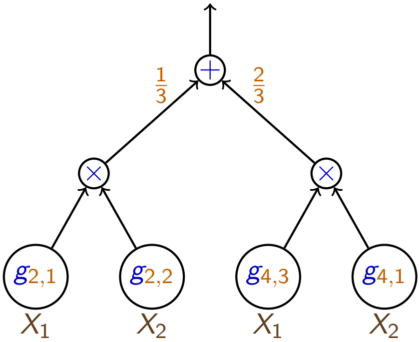


Probabilistic circuits (PCs)

- Circuits where
  - *Random* variables considered
  - input functions are *probability mass functions/densities*
  - weighted sums are *convex mixtures*
  - products represent *independence*
- Output is 'likelihood' for represented joint probability distribution (normalization not always imposed)
- Scope of functions
- Important properties: *smoothness, decomposability*
- Multilinear in input function values

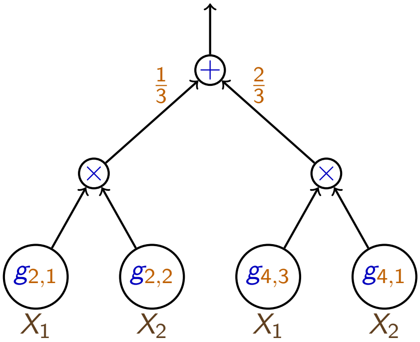


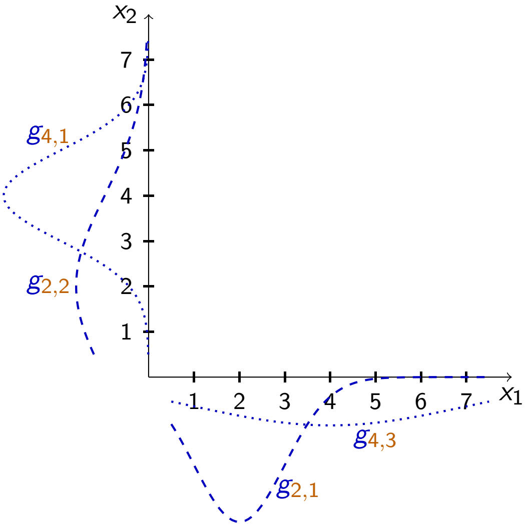
PCs are hierarchical mixture models



$$g_{\mu,\sigma} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\tfrac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

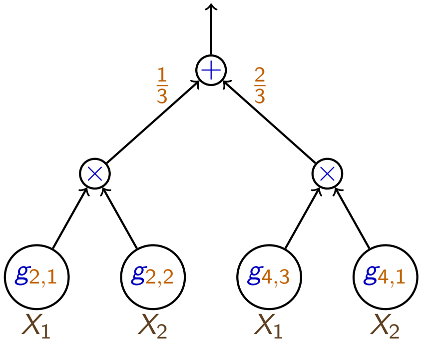
PCs are hierarchical mixture models



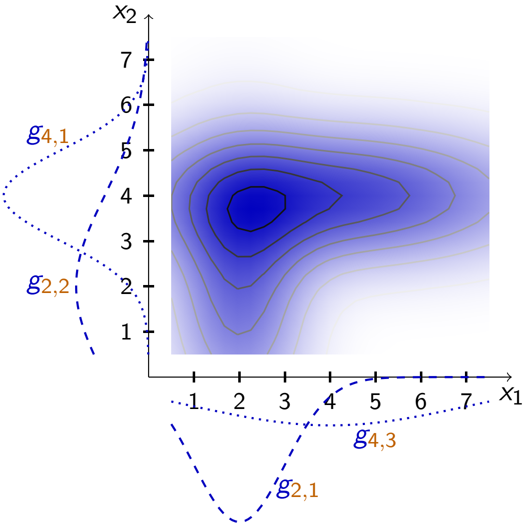


$$g_{\mu,\sigma} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

PCs are hierarchical mixture models

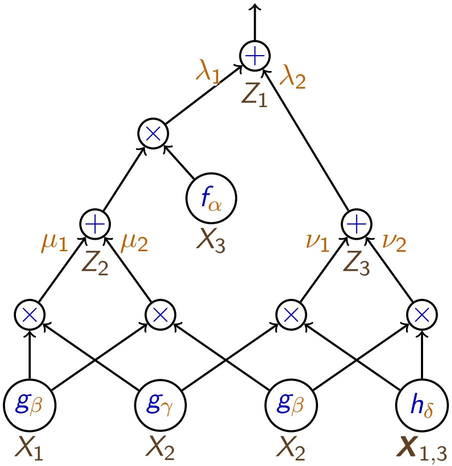


$$g_{\mu,\sigma} = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$



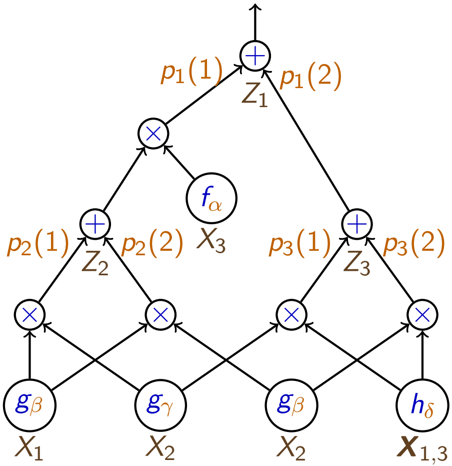
Latent variable interpretation for PCs

- *Latent* random variables
  are associated to sum nodes



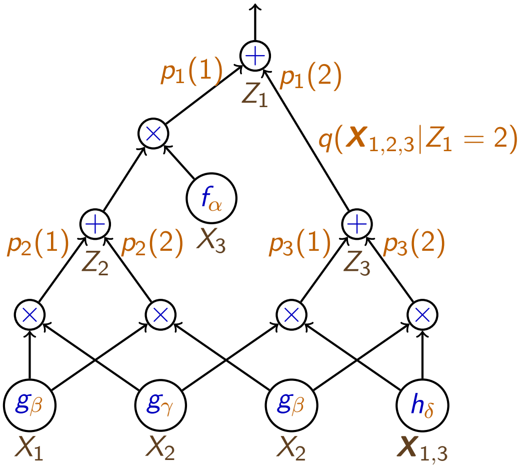
Latent variable interpretation for PCs

- *Latent* random variables are associated to sum nodes
- Sum weights are interpreted as probability values



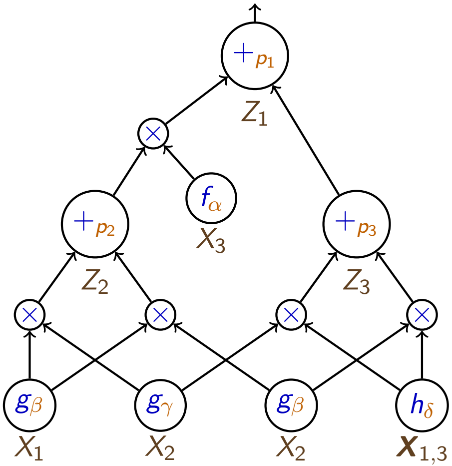
Latent variable interpretation for PCs

- *Latent* random variables are associated to sum nodes
- Sum weights are interpreted as probability values
- Sum node inputs are interpreted as conditional probability values



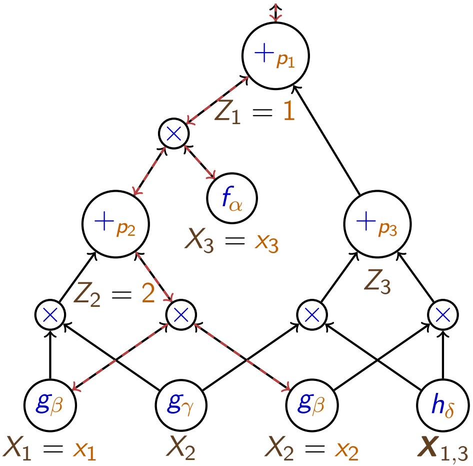
Latent variable interpretation for PCs

- *Latent* random variables are associated to sum nodes
- Sum weights are interpreted as probability values
- Sum node inputs are interpreted as conditional probability values



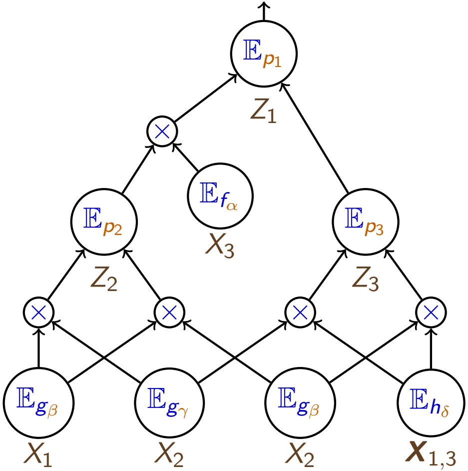
Latent variable interpretation for PCs

- *Latent* random variables are associated to sum nodes
- Sum weights are interpreted as probability values
- Sum node inputs are interpreted as conditional probability values
- Sampling the PC:
  - recursively sample the (latent) distributions from root to leaves
  - select single path at sum nodes
  - follow all paths at product nodes



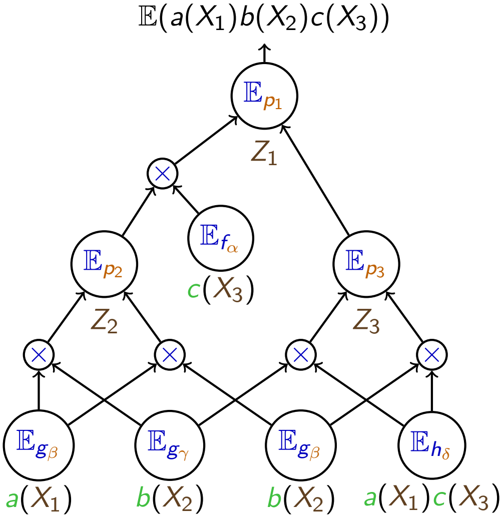
Expectation circuits

- 'Sum' nodes compute *expectations* of functions defined by incoming edge values
- Replace input distributions by their *expectation* operators; input changed from variable values to functions
- Output is joint expectation



Expectation circuits

- ‘Sum’ nodes compute *expectations* of functions defined by incoming edge values
- Replace input distributions by their *expectation* operators; input changed from variable values to functions
- Output is joint expectation
- Joint function must *factorize* for single-pass computation
- Otherwise: use algorithm with only factorizable functions



PCs are often efficient

(Which is why they are popular)

- Class of PCs;
  - each characterized by *size* (number of edges)
  - leaf computation assumed 'efficient'
- Classes of queries:
  - EVI: $\mathbb{E}(a(X_1)b(X_2)c(X_3))$
  - MAR: $\mathbb{E}(a(X_1)b(X_2))$
  - CON: $\mathbb{E}(a(X_1)b(X_2)|X_3 = x_3)$
  - MAP: $\text{argmax}_{\boldsymbol{x}_{1,2}} \mathbb{E}(a(x_1)b(x_2)|X_3 = x_3)$
  - ...

PCs are often efficient

(Which is why they are popular)

- Class of PCs;
  - each characterized by *size* (number of edges)
  - leaf computation assumed 'efficient'

*A query class is **tractable** for a model class if the computational cost of running such a query on such a model is polynomial in the model's size.*

- Classes of queries:
  - EVI: $\mathbb{E}(a(X_1)b(X_2)c(X_3))$
  - MAR: $\mathbb{E}(a(X_1)b(X_2))$
  - CON: $\mathbb{E}(a(X_1)b(X_2)|X_3 = x_3)$
  - MAP: $\text{argmax}_{\boldsymbol{x}_{1,2}} \mathbb{E}(a(x_1)b(x_2)|X_3 = x_3)$
  - ...

- Tractability of PCs:
  - EVI: ✔(single pass, linear)
  - MAR: ✔(single pass, linear)
  - CON: ✔(double pass, linear)
  - MAP: ✘($\geq$ NP-hard)

Overview

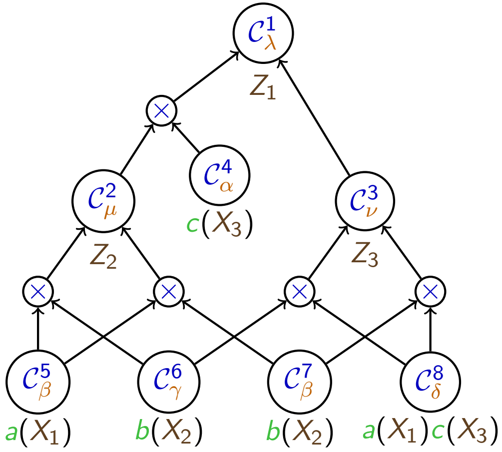
Circuits, conceptually

Credal PCs

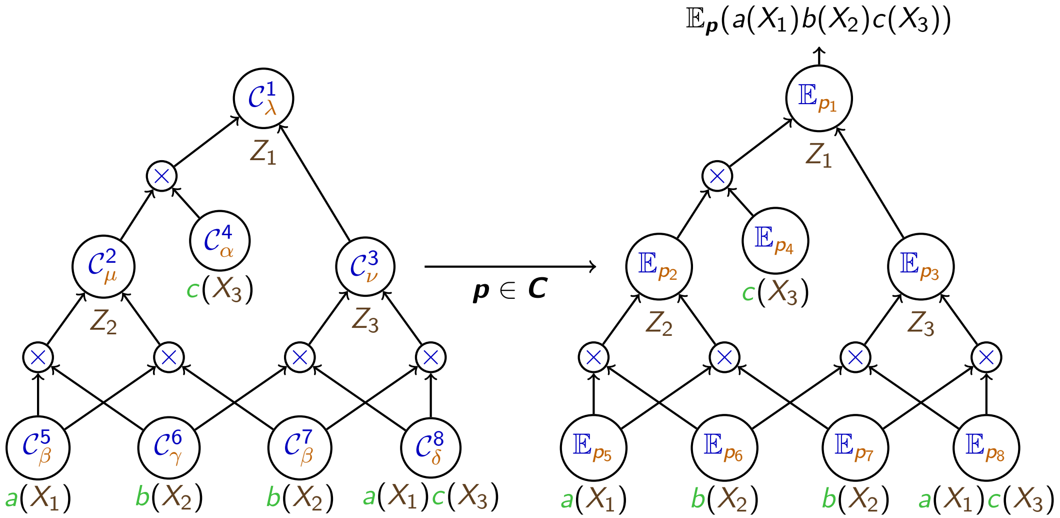
- Definition
- Propagation
- Opportunities

Probabilistic integral circuits

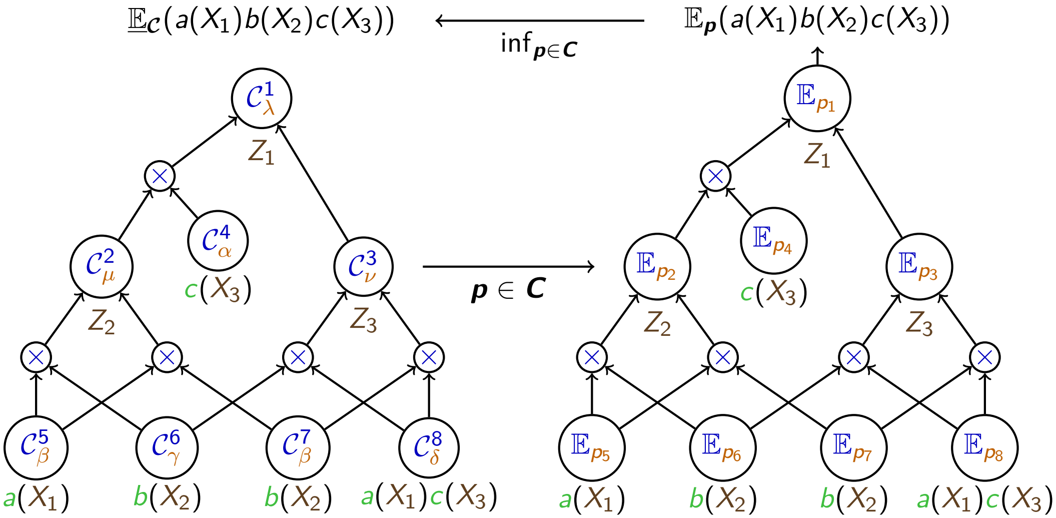
Credal PCs (aka credal sum-product networks)



Credal PCs (aka credal sum-product networks)



Credal PCs *are defined as lower envelopes of PCs*



Working with credal PCs

- Assumed that local credal sets are
  - coherent (nonempty)
  - 'efficient' to work with (linear-vacuous, 2-monotone, simple LP, etc.)

Working with credal PCs

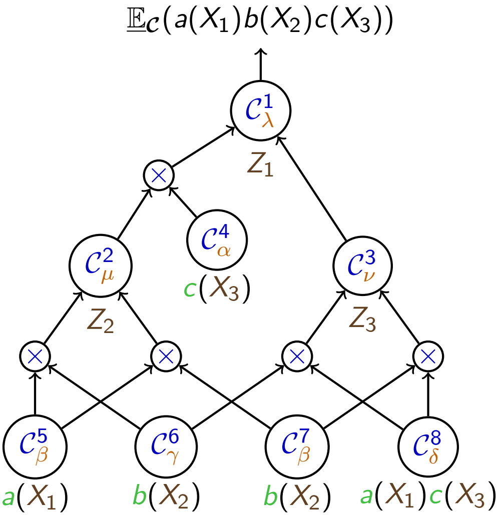
- Assumed that local credal sets are
  - coherent (nonempty)
  - ‘efficient’ to work with (linear-vacuous, 2-monotone, simple LP, etc.)

- Consequences of lower envelope definition:
  - *complete independence* implicitly assumed
  - $\underline{\mathbb{E}}_{\mathcal{C}}$ is coherent
  - Upper expectations follow from conjugacy (or use upper envelope)

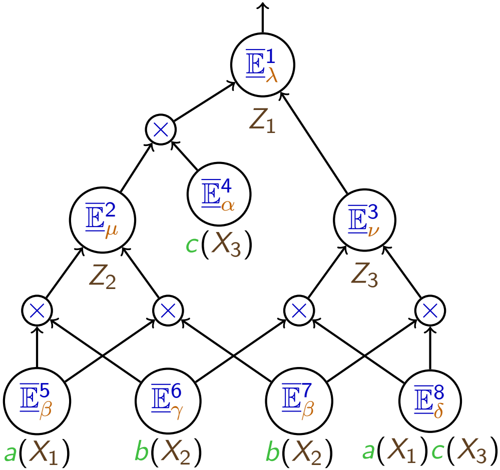
Working with credal PCs

- Assumed that local credal sets are
  - coherent (nonempty)
  - ‘efficient’ to work with (linear-vacuous, 2-monotone, simple LP, etc.)
- Consequences of lower envelope definition:
  - *complete independence* implicitly assumed
  - $\underline{\mathbb{E}}_{\mathcal{C}}$ is coherent
  - Upper expectations follow from conjugacy (or use upper envelope)
- How to calculate the lower envelope?
  - Gradient descent on PC parameters using backpropagation (likely inefficient, to be combined with constrained optimization in local credal sets)
  - Work directly with lower and upper expectations?

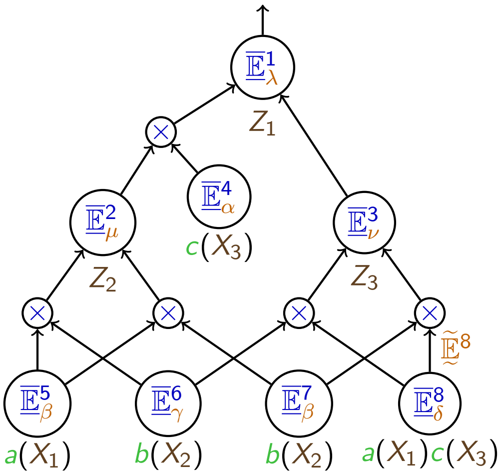
Lower–upper expectation circuits



Lower–upper expectation circuits



Lower–upper expectation circuits

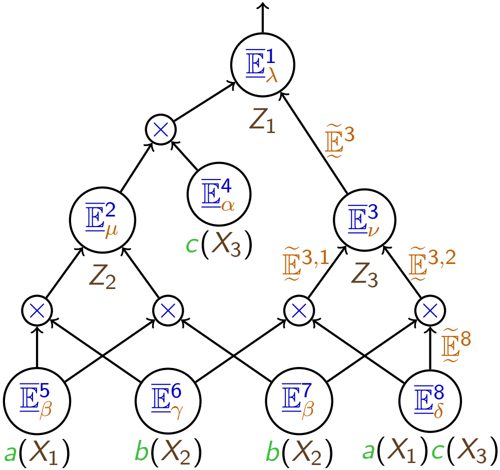


Propagation calculation rules

- Leaf expectation nodes with incoming function d^i:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(d^i) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(d^i)$$

Lower–upper expectation circuits



Propagation calculation rules

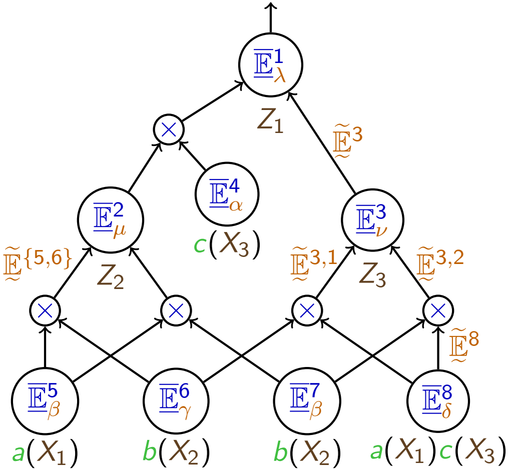
- Leaf expectation nodes with incoming function d^i:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(d^i) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(d^i)$$

- Interior expectation nodes:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(\underset{\sim}{\mathbb{E}}^{i,Z_i}) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(\widetilde{\mathbb{E}}^{i,Z_i})$$

Lower–upper expectation circuits



Propagation calculation rules

- Leaf expectation nodes with incoming function d^i:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(d^i) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(d^i)$$

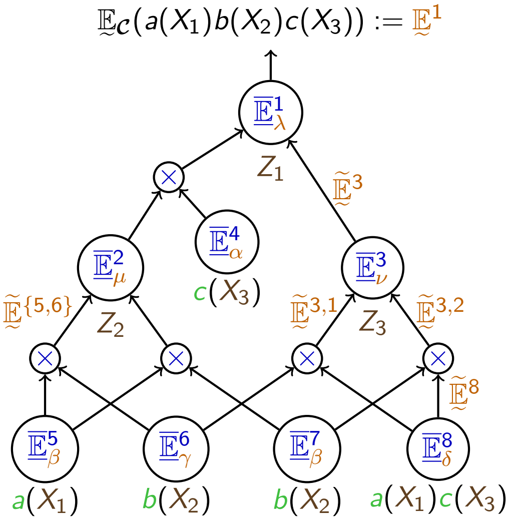
- Interior expectation nodes:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(\widetilde{\underset{\sim}{\mathbb{E}}}^{i,Z_i}) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(\widetilde{\underset{\sim}{\mathbb{E}}}^{i,Z_i})$$

- Product nodes (using interval arithmetic):

$$[\underset{\sim}{\mathbb{E}}^J, \widetilde{\mathbb{E}}^J] := \prod_{j\in J}[\underset{\sim}{\mathbb{E}}^j, \widetilde{\mathbb{E}}^j]$$

Lower–upper expectation circuits



Propagation calculation rules

- Leaf expectation nodes with incoming function d^i:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(d^i) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(d^i)$$

- Interior expectation nodes:

$$\underset{\sim}{\mathbb{E}}^i = \underline{\mathbb{E}}^i(\underset{\sim}{\mathbb{E}}^{i,Z_i}) \qquad \widetilde{\mathbb{E}}^i = \overline{\mathbb{E}}^i(\widetilde{\mathbb{E}}^{i,Z_i})$$

- Product nodes (using interval arithmetic):

$$[\underset{\sim}{\mathbb{E}}^J, \widetilde{\mathbb{E}}^J] := \prod_{j\in J}[\underset{\sim}{\mathbb{E}}^j, \widetilde{\mathbb{E}}^j]$$

Using propagation to compute credal PC lower expectations[1]

Exact inference

It sometimes holds that $\underline{\mathbb{E}}_{\mathcal{C}}(f) = \underset{\sim}{\mathbb{E}}_{\mathcal{C}}(f)$:

- For *tree* topologies, this holds for *any factorizing f*
- For *SDAG* topologies, this holds for *any nonnegative factorizing f*

[1] All credal PC results from Montalvan, Centen, Krak, Quaeghebeur & De Campos's "Beyond tree-shaped credal probabilistic circuits", *IJAR* 171:109047 (2024).

Using propagation to compute credal PC lower expectations[1]

Exact inference

It sometimes holds that $\underline{\mathbb{E}}_{\mathcal{C}}(f) = \underset{\sim}{\mathbb{E}}_{\mathcal{C}}(f)$:

- For *tree* topologies, this holds for *any factorizing f*
- For *SDAG* topologies, this holds for *any nonnegative factorizing f*

Computational cost

A single propagation pass is linear in the size of the circuit, so for the cases above

- EVI: ✔
- MAR: ✔

[1]All credal PC results from Montalvan, Centen, Krak, Quaeghebeur & De Campos's "Beyond tree-shaped credal probabilistic circuits", *IJAR* 171:109047 (2024).

Using propagation to compute credal PC lower expectations[1]

Exact inference

It sometimes holds that $\underline{\mathbb{E}}_{\mathcal{C}}(f) = \underset{\sim}{\mathbb{E}}_{\mathcal{C}}(f)$:

- For *tree* topologies, this holds for *any factorizing f*
- For *SDAG* topologies, this holds for *any nonnegative factorizing f*

Computational cost

A single propagation pass is linear in the size of the circuit, so for the cases above

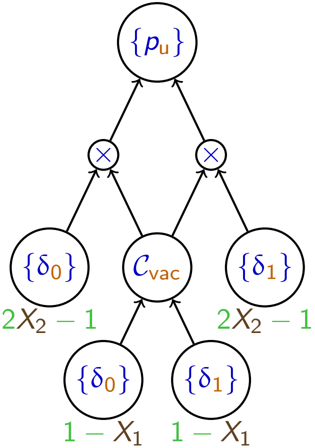
- EVI: ✔
- MAR: ✔

Approximate inference: outer approximation

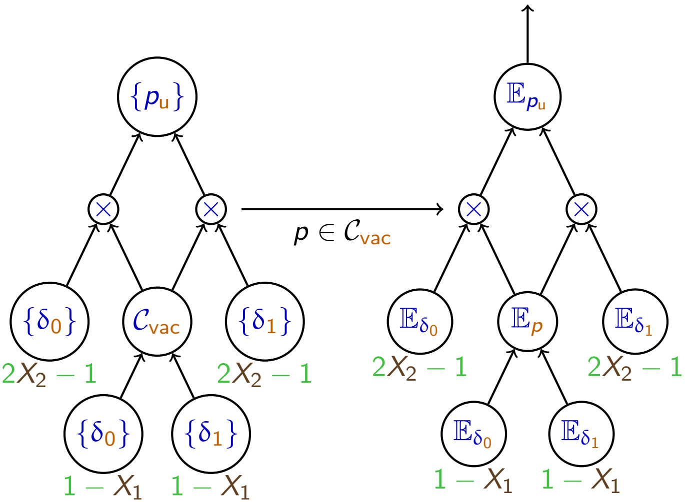
It always holds that $\underset{\sim}{\mathbb{E}}_{\mathcal{C}}(f) \leq \underline{\mathbb{E}}_{\mathcal{C}}(f)$ for any factorizing f.

[1] All credal PC results from Montalvan, Centen, Krak, Quaeghebeur & De Campos's "Beyond tree-shaped credal probabilistic circuits", *IJAR* 171:109047 (2024).

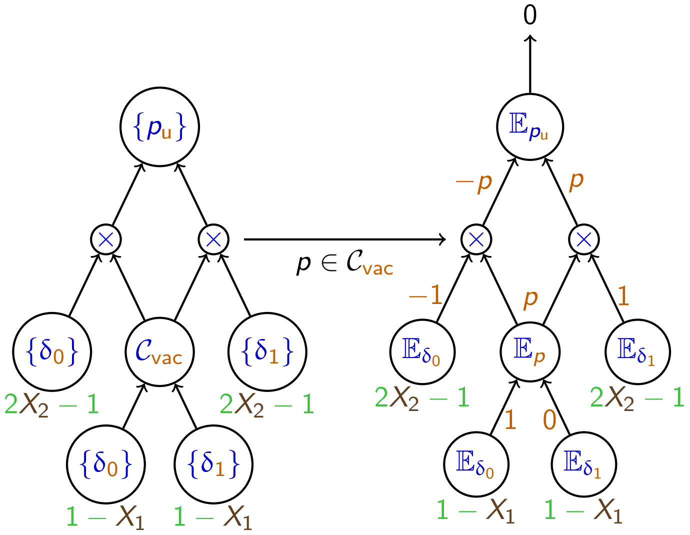
The problem with cycles (binary variables X_1 and X_2)



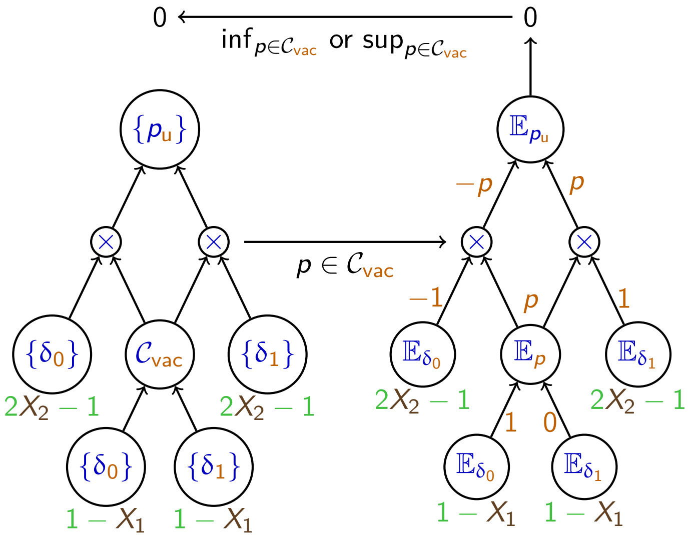
The problem with cycles (binary variables X_1 and X_2)



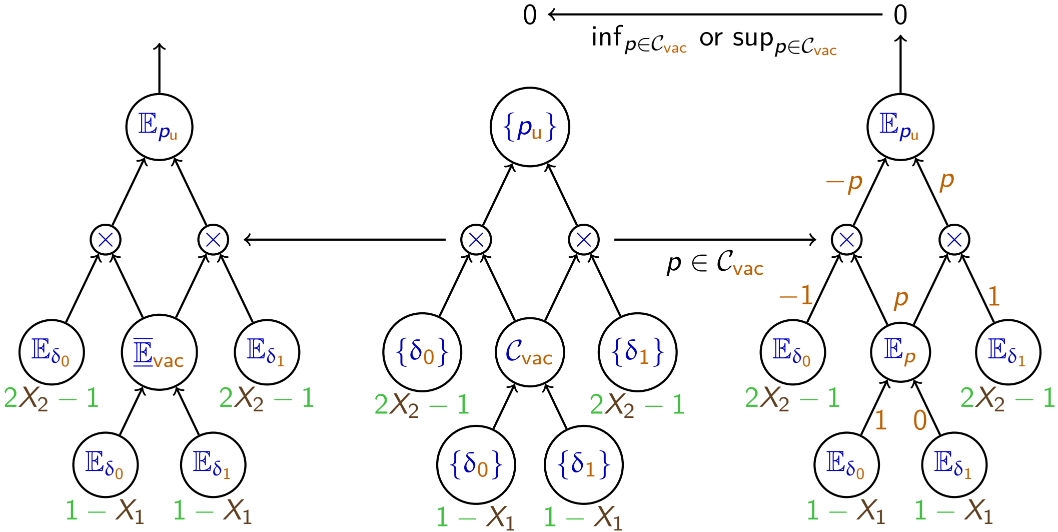
The problem with cycles (binary variables X_1 and X_2)



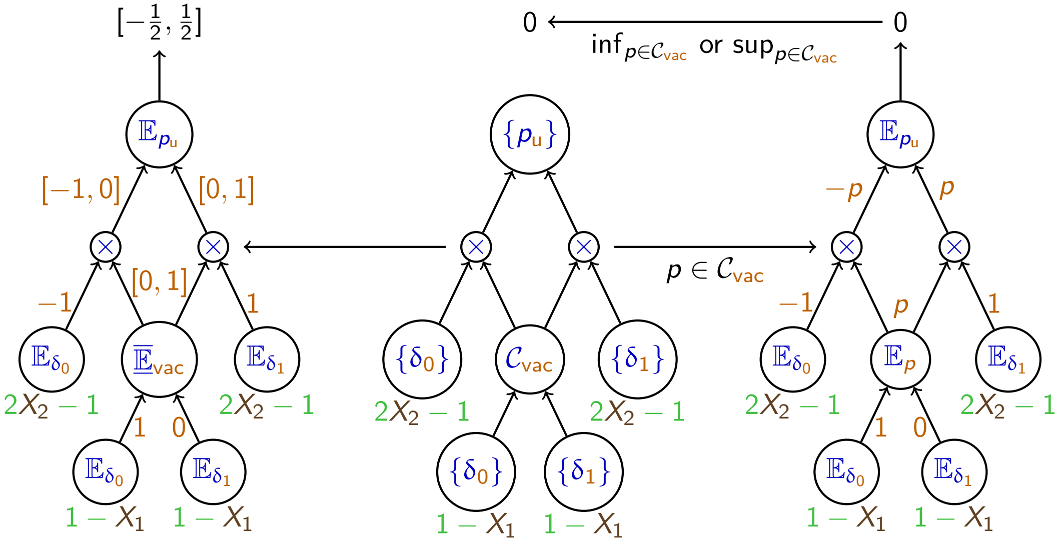
The problem with cycles (binary variables X_1 and X_2)



The problem with cycles (binary variables X_1 and X_2)



The problem with cycles (binary variables X_1 and X_2)

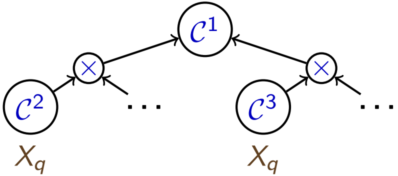


SDAGs designed for tractable inference

Setup

- *General focused query function*:
  $f(\boldsymbol{X}) = g(X_q)h(\boldsymbol{X}_{\neq q})$, with
  - g any function
  - h nonnegative factorizing

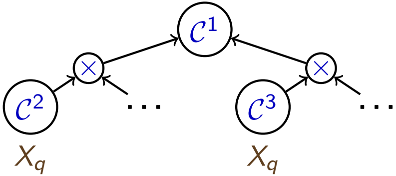
- *X_q-queried structure*:
  no path between the root and any input node with scope X_q contains edges that are part of a cycle



SDAGs designed for tractable inference

Setup

- *General focused query function*: $f(\boldsymbol{X}) = g(X_q)h(\boldsymbol{X}_{\neq q})$, with
  - g any function
  - h nonnegative factorizing
- *X_q-queried structure*: no path between the root and any input node with scope X_q contains edges that are part of a cycle



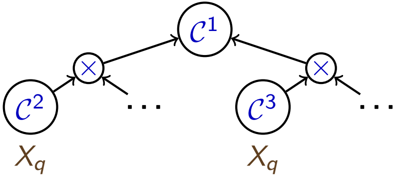
Results

- In this setup $\underline{\mathbb{E}}_{\mathcal{C}}(f) = \utilde{\mathbb{E}}_{\mathcal{C}}(f)$

SDAGs designed for tractable inference

Setup

- *General focused query function*: $f(\boldsymbol{X}) = g(X_q)h(\boldsymbol{X}_{\neq q})$, with
  - g any function
  - h nonnegative factorizing
- *X_q-queried structure*: no path between the root and any input node with scope X_q contains edges that are part of a cycle



Results

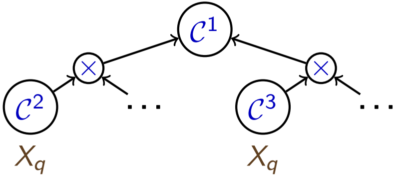
- In this setup $\mathbb{E}_{\mathcal{C}}(f) = \mathbb{E}_{\tilde{\mathcal{C}}}(f)$
- Also $\mathbb{E}_{\mathcal{C}}(g(X_q)|\boldsymbol{X}_e = \boldsymbol{x}_e)$ with $\boldsymbol{X}_e \subseteq \boldsymbol{X}_{\neg q}$ can be computed exactly by propagation, as the solution ζ of

$$\mathbb{E}_{\mathcal{C}}\left(\mathbb{I}_{\boldsymbol{x}_e}(\boldsymbol{X}_e)(g(X_q) - \zeta)\right) = 0$$

SDAGs designed for tractable inference

Setup

- *General focused query function*: $f(\boldsymbol{X}) = g(X_q)h(\boldsymbol{X}_{\neq q})$, with
  - g any function
  - h nonnegative factorizing
- X_q*-queried structure*: no path between the root and any input node with scope X_q contains edges that are part of a cycle



Results

- In this setup $\underline{\mathbb{E}}_{\mathcal{C}}(f) = \underset{\sim}{\underline{\mathbb{E}}}_{\mathcal{C}}(f)$
- Also $\underline{\mathbb{E}}_{\mathcal{C}}(g(X_q) | \boldsymbol{X}_{\mathrm{e}} = \boldsymbol{x}_{\mathrm{e}})$ with $\boldsymbol{X}_{\mathrm{e}} \subseteq \boldsymbol{X}_{\neg q}$ can be computed exactly by propagation, as the solution ζ of

$$\underline{\mathbb{E}}_{\mathcal{C}}\Big(\mathbb{I}_{\boldsymbol{x}_{\mathrm{e}}}(\boldsymbol{X}_{\mathrm{e}})(g(X_q) - \zeta)\Big) = 0$$

- Determining *credal dominance* between decision options $\hat{o}$ and $\check{o}$ is a special case if this, as it requires investigating

$$\underline{\mathbb{E}}_{\mathcal{C}}\Big(\mathbb{I}_{\hat{o}}(X_q) - \mathbb{I}_{\check{o}}(X_q) \Big| \boldsymbol{X}_{\mathrm{e}} = \boldsymbol{x}_{\mathrm{e}}\Big) > 0$$

Opportunities in the area of imprecise-probabilistic PCs

- Embracing lower–upper expectation circuits fully by considering SDAGs as a compact notation for trees?
- Other ‘product’ operators?
- Learning from data beyond adding ‘imprecision’ to sum nodes?

Overview

Circuits, conceptually

Credal PCs

Probabilistic integral circuits

- From deep PCs to PICs
- Training PICs
- Opportunities

Evolution and characteristics of deep PCs

- Great increase in size over time (now in the billions of parameters)

Evolution and characteristics of deep PCs

- Great increase in size over time (now in the billions of parameters)
- Choice of structures expands
  - Data-based, mostly trees (e.g., Learn-SPN and derivatives)
  - Random SDAGs (e.g., RAT-SPN)
  - Application-based (e.g., PD, QG, QT for images)
- Tendency to overparameterize and tensorize structures
  (making it well-adapted to deep-learning implementation)

Evolution and characteristics of deep PCs

- Great increase in size over time (now in the billions of parameters)
- Choice of structures expands
  - Data-based, mostly trees (e.g., Learn-SPN and derivatives)
  - Random SDAGs (e.g., RAT-SPN)
  - Application-based (e.g., PD, QG, QT for images)
- Tendency to overparameterize and tensorize structures (making it well-adapted to deep-learning implementation)
- Cycles seem to have a positive effect on expressiveness (reuse of components)
- Training seems harder than in neural-based deep learning models

Probabilistic integral circuits (PICs)[2]

Observation Other classes of deep generative models are easier to train (VAEs, Flows)

Hypothesis Their intertwined use of *continuous* latent variables and *neural networks* play an important role in this

---

[2] Results and material from

- Gala, De Campos, Peharz, Vergari & Quaeghebeur's "Probabilistic integral circuits", AISTATS 2024, PMLR 238:2143–2151 (2024).
- Gala, De Campos, Vergari & Quaeghebeur's "Scaling continuous latent variable models as probabilistic integral circuits", NeurIPS 2024 (2024).

Various slide elements gracefully provided by Gennaro Gala.

Probabilistic integral circuits (PICs)[2]

Observation Other classes of deep generative models are easier to train (VAEs, Flows)

Hypothesis Their intertwined use of *continuous* latent variables and *neural networks* play an important role in this

Objective Add *continuous* latent variables and *neural networks* to improve training and scaling

Key addition *Integration nodes* that replace sum nodes

---

[2] Results and material from

- Gala, De Campos, Peharz, Vergari & Quaeghebeur's "Probabilistic integral circuits", AISTATS 2024, PMLR 238:2143–2151 (2024).
- Gala, De Campos, Vergari & Quaeghebeur's "Scaling continuous latent variable models as probabilistic integral circuits", NeurIPS 2024 (2024).

Various slide elements gracefully provided by Gennaro Gala.

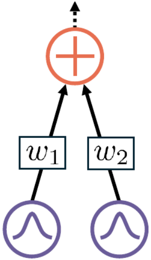
Probabilistic integral circuits (PICs)[2]

Observation Other classes of deep generative models are easier to train (VAEs, Flows)

Hypothesis Their intertwined use of *continuous* latent variables and *neural networks* play an important role in this

Objective Add *continuous* latent variables and *neural networks* to improve training and scaling

Key addition *Integration nodes* that replace sum nodes



---

[2] Results and material from

- Gala, De Campos, Peharz, Vergari & Quaeghebeur's "Probabilistic integral circuits", AISTATS 2024, PMLR 238:2143–2151 (2024).
- Gala, De Campos, Vergari & Quaeghebeur's "Scaling continuous latent variable models as probabilistic integral circuits", NeurIPS 2024 (2024).

Various slide elements gracefully provided by Gennaro Gala.

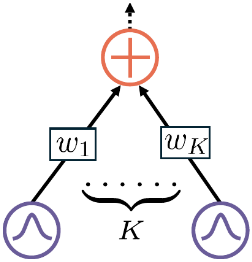
Probabilistic integral circuits (PICs)[2]

Observation Other classes of deep generative models are easier to train (VAEs, Flows)

Hypothesis Their intertwined use of *continuous* latent variables and *neural networks* play an important role in this

Objective Add *continuous* latent variables and *neural networks* to improve training and scaling

Key addition *Integration nodes* that replace sum nodes



[2]Results and material from

- Gala, De Campos, Peharz, Vergari & Quaeghebeur's "Probabilistic integral circuits", AISTATS 2024, PMLR 238:2143–2151 (2024).
- Gala, De Campos, Vergari & Quaeghebeur's "Scaling continuous latent variable models as probabilistic integral circuits", NeurIPS 2024 (2024).

Various slide elements gracefully provided by Gennaro Gala.

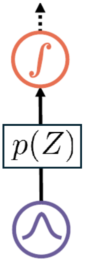
Probabilistic integral circuits (PICs)[2]

Observation Other classes of deep generative models are easier to train (VAEs, Flows)

Hypothesis Their intertwined use of *continuous* latent variables and *neural networks* play an important role in this

Objective Add *continuous* latent variables and *neural networks* to improve training and scaling

Key addition *Integration nodes* that replace sum nodes

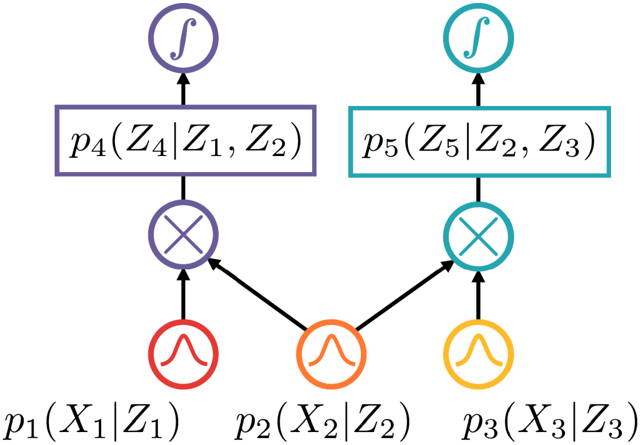


[2] Results and material from

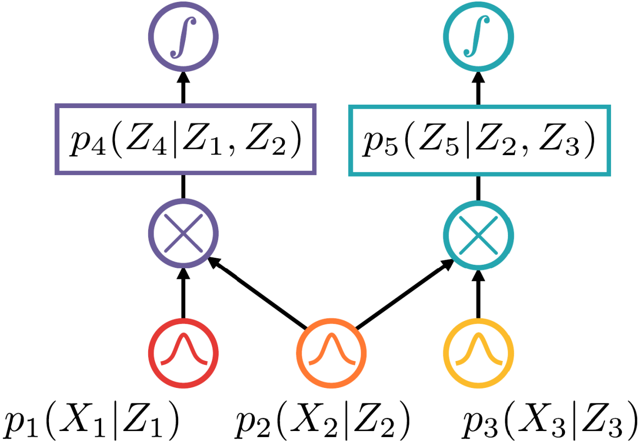
- Gala, De Campos, Peharz, Vergari & Quaeghebeur's "Probabilistic integral circuits", AISTATS 2024, PMLR 238:2143–2151 (2024).
- Gala, De Campos, Vergari & Quaeghebeur's "Scaling continuous latent variable models as probabilistic integral circuits", NeurIPS 2024 (2024).

Various slide elements gracefully provided by Gennaro Gala.

PICs are 'symbolic' circuits

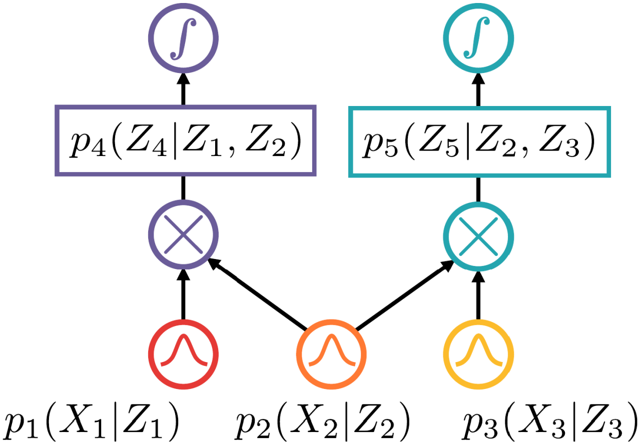


PICs are 'symbolic' circuits



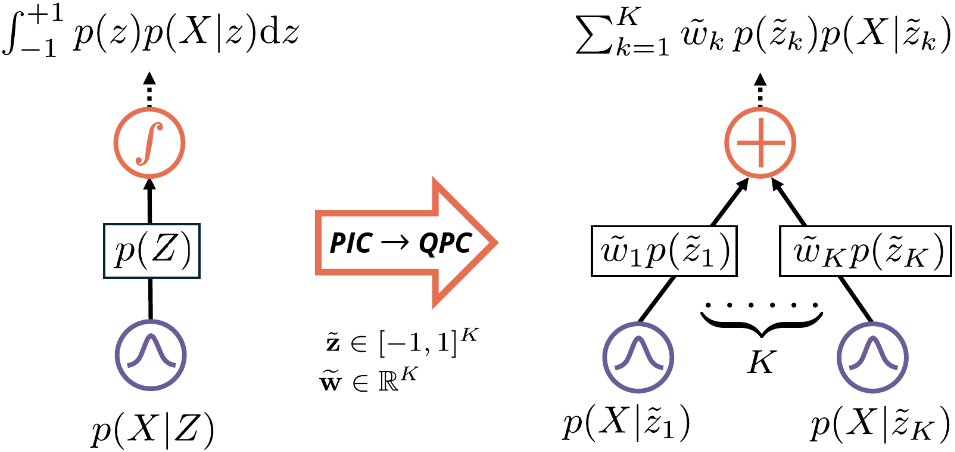
- Integral node:
  - *integrates out* latent variables
  - *leaves unaffected* non-latent variables
  - *introduces* new latent variable
  - parameterized with a *neural network*

PICs are 'symbolic' circuits

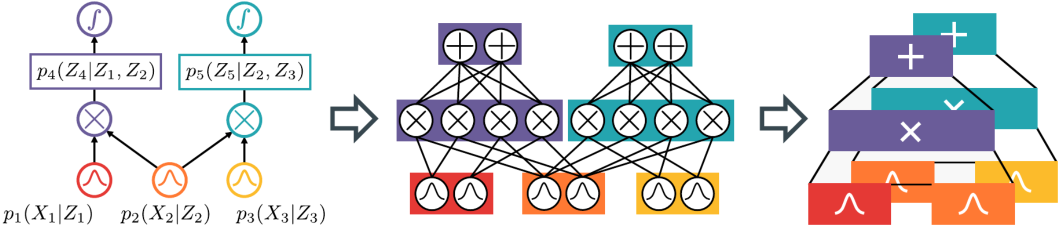


- Integral node:
  - *integrates out* latent variables
  - *leaves unaffected* non-latent variables
  - *introduces* new latent variable
  - parameterized with a *neural network*
- *Edges carry functions*, not values!
- Generally not tractable as such

Approximating PICs with PCs by numerical quadrature

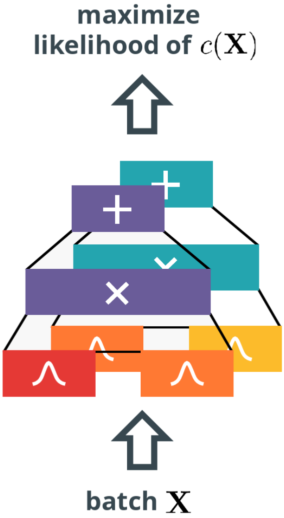


Approximating PICs with PCs by numerical quadrature



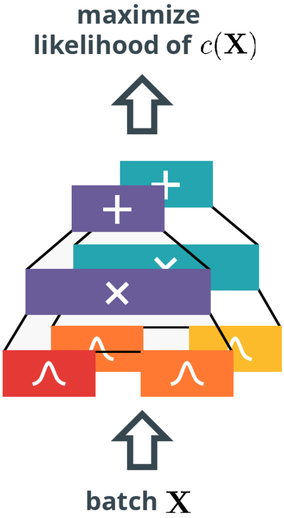
- tensorization and overparameterization is applied
- locally dense
- *folding* to exploit parallel computation

Training a PIC via its QPC



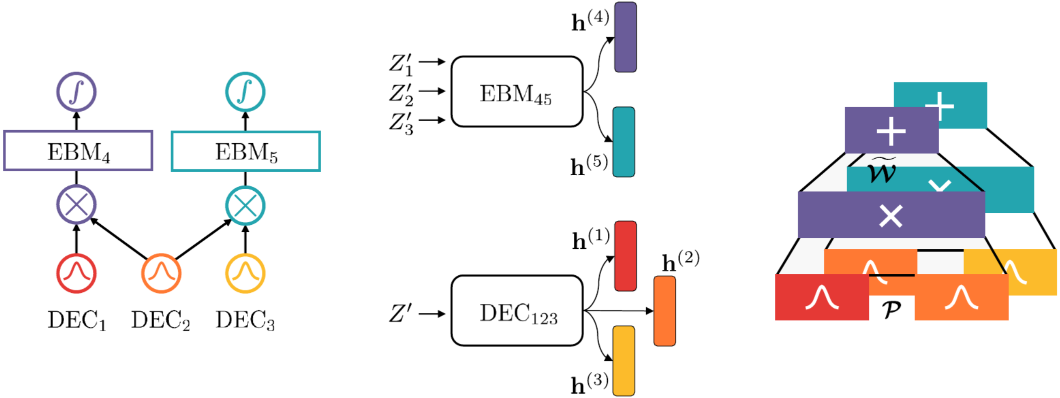
- Training QPC:
  - use gradient ascent to maximize likelihood
  - calculate gradients using backpropagation
  - use random batches of data
  - take maximizing step *for sum weights* per batch
  - repeat with many batches until convergence

Training a PIC via its QPC



- Training QPC:
  - use gradient ascent to maximize likelihood
  - calculate gradients using backpropagation
  - use random batches of data
  - take maximizing step *for sum weights* per batch
  - repeat with many batches until convergence
- Training PIC:
  - as above, but
  - propagate gradients to neural network parameters
  - take maximizing step *for neural network parameters* per batch

Making training practical using neural functional sharing



Takeaway & opportunities in the area of deep PCs

- State-of-the-art deep PCs (PICs) combine ideas from various fields and require ML engineering to design.
- Methods to directly train PICs without generating QPCs?
- Create imprecise-probabilistic PIC variants?

The wondrous world of credal and deep probabilistic circuits

SIPTA Seminar

Erik Quaeghebeur

Eindhoven University of Technology

22 January 2025, 15:00 CET