

Synthesising Recursive Functions for First-Order Model Counting

Paulius Dilkas

Joint work with Vaishak Belle (University of Edinburgh, UK)

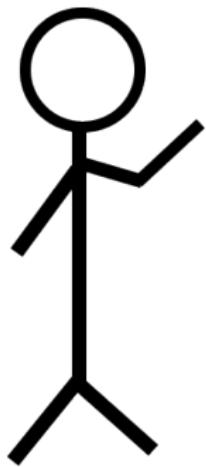
PLSE Seminar, 21st April 2023

National University of Singapore, Singapore

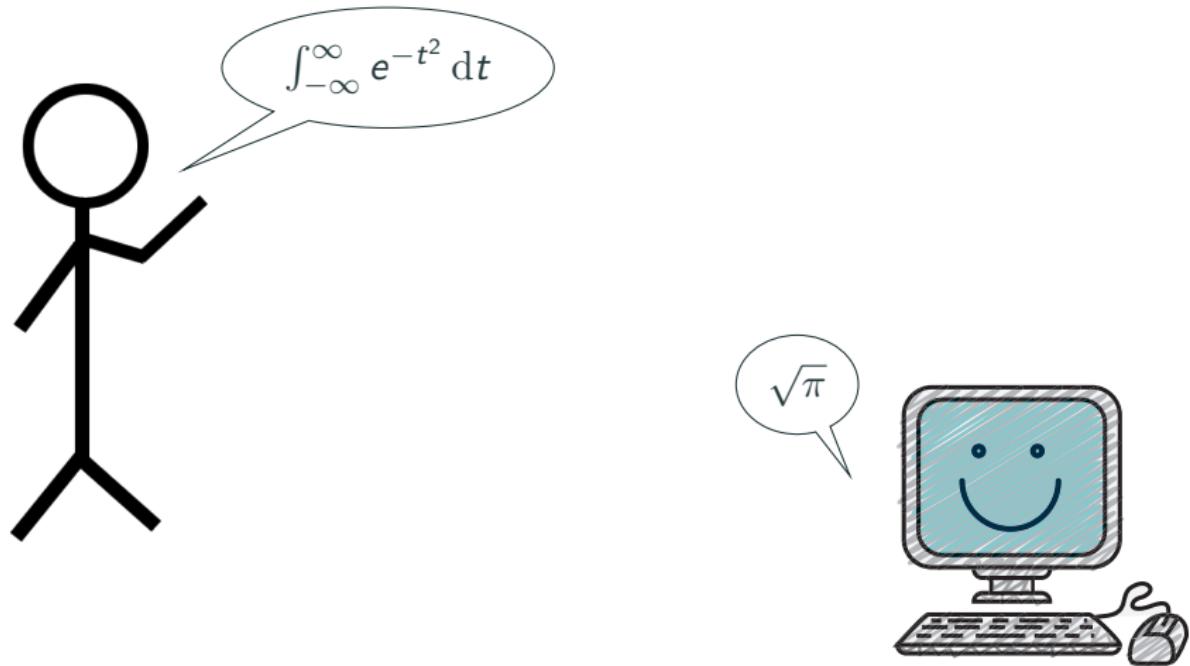


Engineering and
Physical Sciences
Research Council

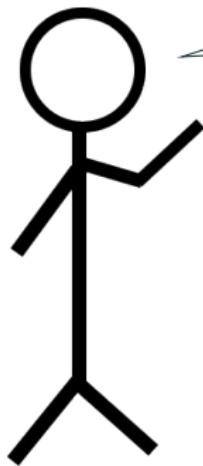
What Computers Can and Cannot Do



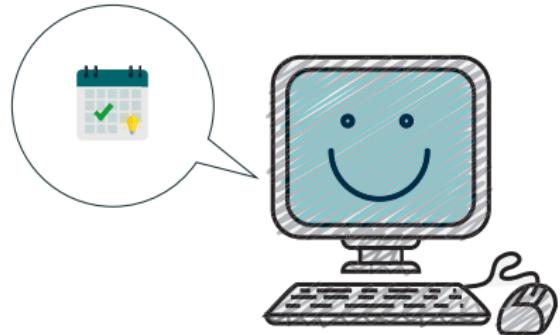
What Computers Can and Cannot Do



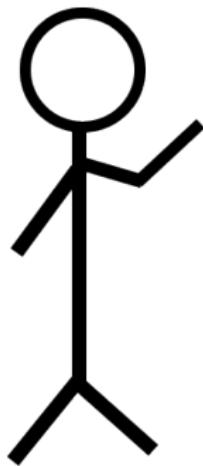
What Computers Can and Cannot Do



Produce a **schedule** for the
nurses at the local hospital.



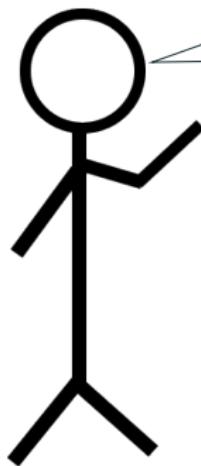
What Computers Can and Cannot Do



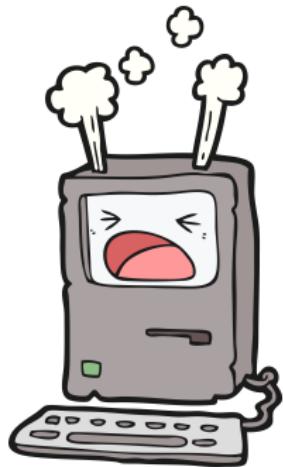
Paint a baroque oil painting of a
raccoon queen wearing a crown.



What Computers Can and Cannot Do



If I shuffle a deck of n cards,
how many possible outcomes
are there?



Terms and conditions apply.

Who Cares About Counting?

Who Cares About Counting?

Probabilistic Programming

*Inference and learning in probabilistic logic
programs using weighted Boolean formulas*

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

*Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)*

Who Cares About Counting?

Probabilistic Programming

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

*Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)*

Neuro-symbolic AI

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

Jingyi Xu¹ Ziliu Zhang² Tal Friedman¹ Yitao Liang¹ Guy Van den Broeck¹

Who Cares About Counting?

Probabilistic Programming

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

*Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)*

Neuro-symbolic AI

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

Jingyi Xu¹ Ziliu Zhang² Tal Friedman¹ Yitao Liang¹ Guy Van den Broeck¹

Natural Language Processing

Joint Inference for Knowledge Extraction from Biomedical Literature

Hoifung Poon*

Dept. of Computer Sci. & Eng.
University of Washington
Seattle, WA 98195
hoifung@cs.washington.edu

Lucy Vanderwende

Microsoft Research
Redmond, WA 98052
Lucy.Vanderwende@microsoft.com

Who Cares About Counting?

Probabilistic Programming

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

*Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)*

Robotics

**Learning Relational Affordance Models for Robots
in Multi-Object Manipulation Tasks**

Bogdan Moldovan Plinio Moreno Martijn van Otterlo José Santos-Victor Luc De Raedt

Neuro-symbolic AI

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

Jingyi Xu¹ Ziliu Zhang² Tal Friedman¹ Yitao Liang¹ Guy Van den Broeck¹

Natural Language Processing

Joint Inference for Knowledge Extraction from Biomedical Literature

Hoifung Poon*

Dept. of Computer Sci. & Eng.
University of Washington
Seattle, WA 98195
hoifung@cs.washington.edu

Lucy Vanderwende

Microsoft Research
Redmond, WA 98052
Lucy.Vanderwende@microsoft.com

Who Cares About Counting?

Probabilistic Programming

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

*Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)*

Neuro-symbolic AI

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

Jingyi Xu¹ Zili Zhang² Tal Friedman¹ Yitao Liang¹ Guy Van den Broeck¹

Natural Language Processing

Joint Inference for Knowledge Extraction from Biomedical Literature

Hoifung Poon*

Dept. of Computer Sci. & Eng.
University of Washington
Seattle, WA 98195
hoifung@cs.washington.edu

Lucy Vanderwende

Microsoft Research
Redmond, WA 98052
Lucy.Vanderwende@microsoft.com

Robotics

**Learning Relational Affordance Models for Robots
in Multi-Object Manipulation Tasks**

Bogdan Moldovan Plinio Moreno Martijn van Otterlo José Santos-Victor Luc De Raedt

Bioinformatics

PheNetic: Network-based interpretation of unstructured gene lists in E. coli
Dries De Maeyer¹, Joris Renkens², Lore Cloots¹, Luc De Raedt^{1,2}, Kathleen Marchal^{1,3}

¹Center of Microbial and Plant Genetics, Kasteelpark Arenberg 20, B-3001, Leuven, Belgium

²Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

³Department of Plant Biotechnology and Bioinformatics, Ghent University, Technologiepark 927, 9052 Gent, Belgium

Who Cares About Counting?

Probabilistic Programming

Inference and learning in probabilistic logic programs using weighted Boolean formulas

DAAN FIERENS, GUY VAN DEN BROECK, JORIS RENKENS,
DIMITAR SHTERIONOV, BERND GUTMANN, INGO THON,
GERDA JANSSENS and LUC DE RAEDT

Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
(e-mail: FirstName.LastName@cs.kuleuven.be)

Neuro-symbolic AI

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

Jingyi Xu¹ Zili Zhang² Tal Friedman¹ Yitao Liang¹ Guy Van den Broeck¹

Natural Language Processing

Joint Inference for Knowledge Extraction from Biomedical Literature

Hoifung Poon*

Dept. of Computer Sci. & Eng.
University of Washington
Seattle, WA 98195
hoifung@cs.washington.edu

Lucy Vanderwende

Microsoft Research
Redmond, WA 98052
Lucy.Vanderwende@microsoft.com

Robotics

Learning Relational Affordance Models for Robots
in Multi-Object Manipulation Tasks

Bogdan Moldovan Plinio Moreno Martijn van Otterlo José Santos-Victor Luc De Raedt

Bioinformatics

PheNetic: Network-based interpretation of unstructured gene lists in *E. coli*
Dries De Maeyer¹, Joris Renkens², Lore Cloots¹, Luc De Raedt¹, Kathleen Marchal^{1,3}

¹Center of Microbial and Plant Genetics, Kasteelpark Arenberg 20, B-3001, Leuven, Belgium

²Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

³Department of Plant Biotechnology and Bioinformatics, Ghent University, Technologiepark 927, 9052 Gent, Belgium

Combinatorics

Automatic Conjecturing of P-Recursions
Using Lifted Inference

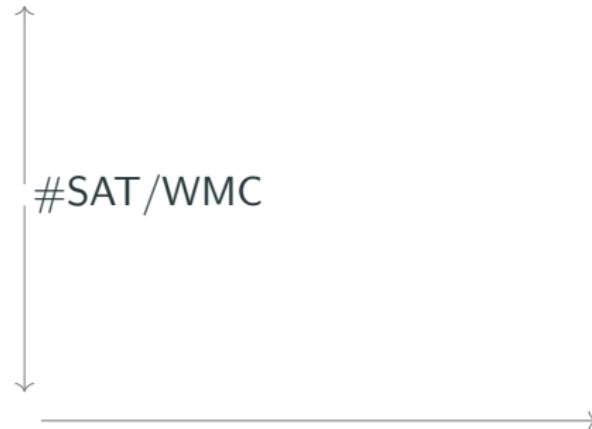
Jáchym Barvínek^{1(✉)}, Timothy van Bremen², Yuyi Wang³, Filip Železný¹,
and Ondřej Kuzelka¹

¹ Czech Technical University in Prague, Prague, Czech Republic
barvijac@fel.cvut.cz

² KU Leuven, Leuven, Belgium

³ ETH Zurich, Zurich, Switzerland

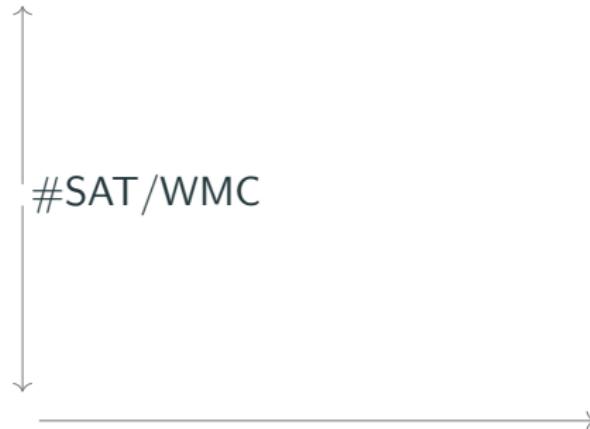
(Some of the) Many Ways to Count



#SAT (Valiant 1979)

- Input formula: $x \vee y$
- Interpretations: $\emptyset, \{x\}, \{y\}, \{x, y\}$
- Models: $\{x\}, \{y\}, \{x, y\}$

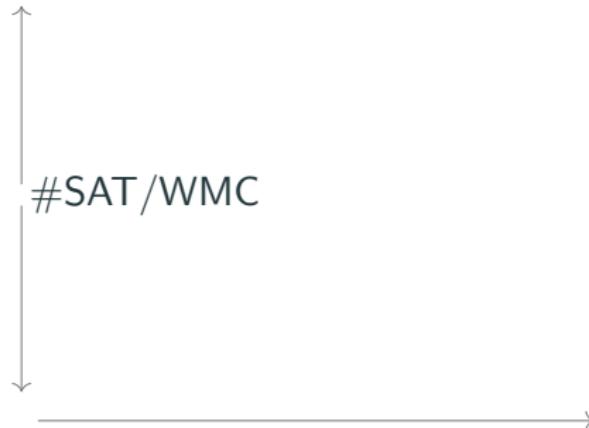
(Some of the) Many Ways to Count



#SAT (Valiant 1979)

- Input formula: $x \vee y$
- Interpretations: $\emptyset, \{x\}, \{y\}, \{x, y\}$
- Models: $\{x\}, \{y\}, \{x, y\}$
- Answer (model count): 3

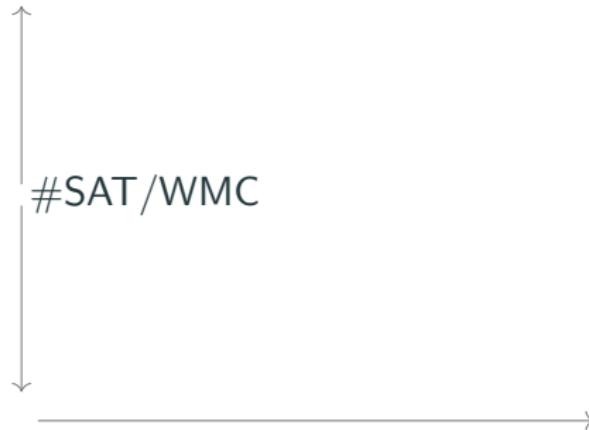
(Some of the) Many Ways to Count



Weighted Model Counting (Chavira and Darwiche 2008)

- Input formula: $x \vee y$
- Input weights: $w(x) = 0.3, w(\neg x) = 0.7,$
 $w(y) = 0.2, w(\neg y) = 0.8$

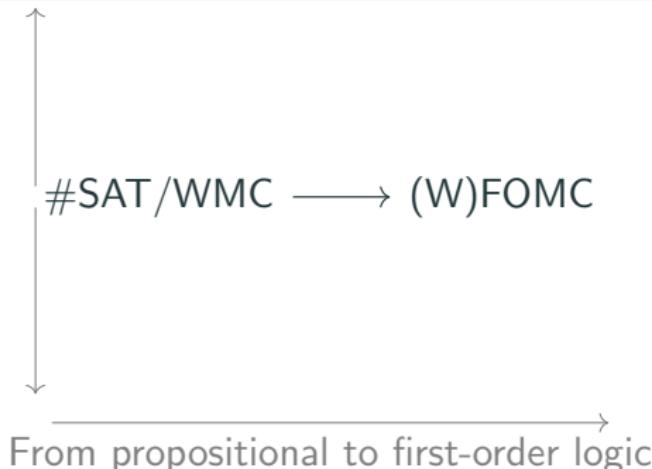
(Some of the) Many Ways to Count



Weighted Model Counting (Chavira and Darwiche 2008)

- Input formula: $x \vee y$
- Input weights: $w(x) = 0.3, w(\neg x) = 0.7,$
 $w(y) = 0.2, w(\neg y) = 0.8$
- Answer (weighted model count):
 $w(x)w(y) + w(x)w(\neg y) + w(\neg x)w(y) = 0.44$

(Some of the) Many Ways to Count

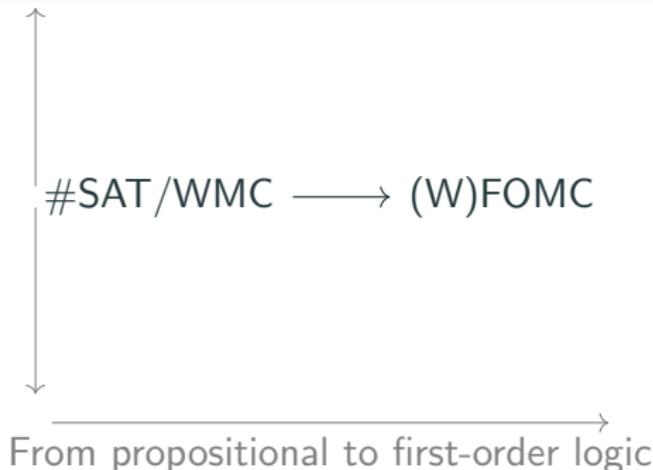


(Weighted) (Symmetric) First-Order Model Counting

(Van den Broeck et al. 2011)

- Input formula: $\forall \textcolor{teal}{x} \in \Delta. P(\textcolor{teal}{x})$
- Input weights: $w^+(P) = 0.3, w^-(P) = 0.7$
- Input domain size(s): $|\Delta| = 2$

(Some of the) Many Ways to Count

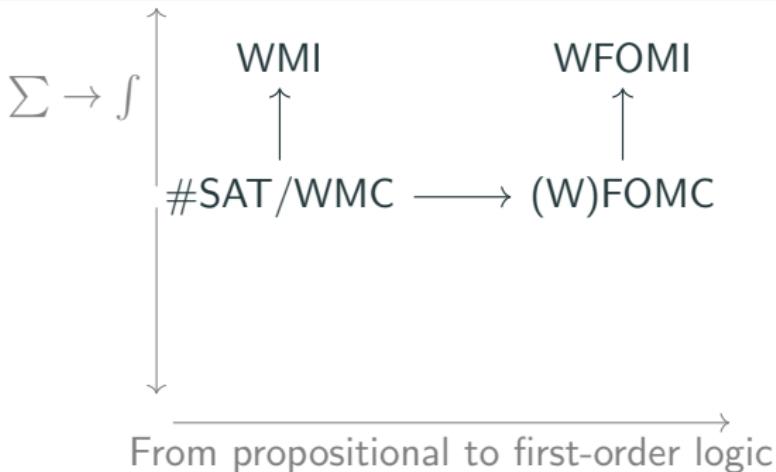


(Weighted) (Symmetric) First-Order Model Counting

(Van den Broeck et al. 2011)

- Input formula: $\forall \textcolor{teal}{x} \in \Delta. P(\textcolor{teal}{x})$
- Input weights: $w^+(P) = 0.3, w^-(P) = 0.7$
- Input domain size(s): $|\Delta| = 2$
- Answer: $(w^+(P))^{|\Delta|} = 0.09$

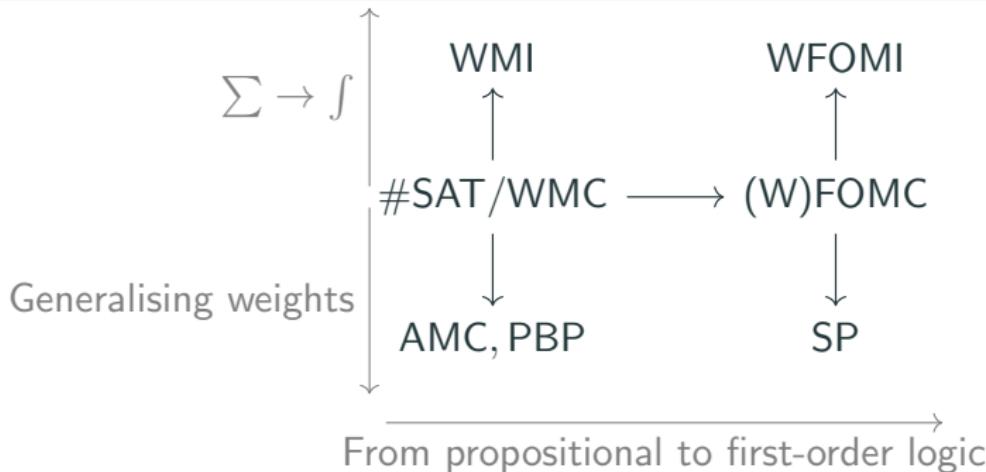
(Some of the) Many Ways to Count



Extensions to Continuous Domains

- Weighted model integration
 - (Belle, Passerini and Van den Broeck 2015)
- Weighted first-order model integration
 - (Feldstein and Belle 2021)

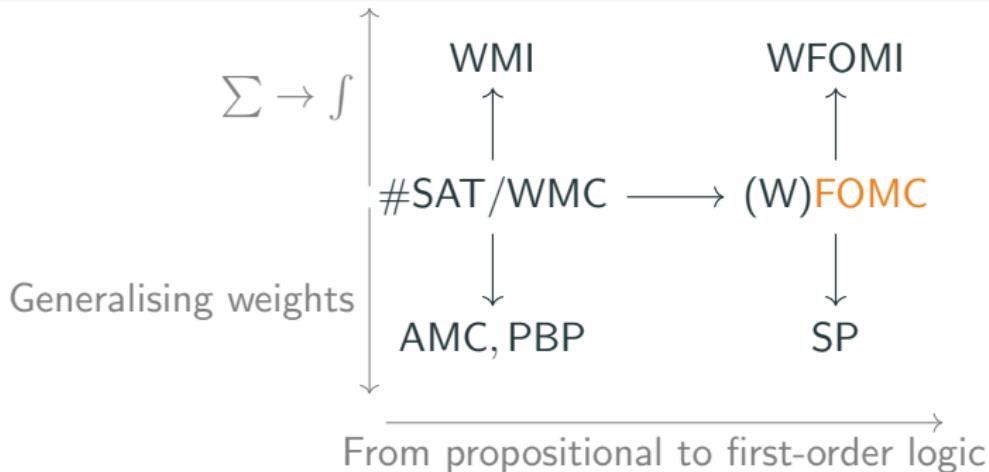
(Some of the) Many Ways to Count



Generalisations of the Weight Function

- Algebraic model counting
 - (Kimmig, Van den Broeck and De Raedt 2017)
 - From $\mathbb{R}_{\geq 0}$ to commutative semirings
- Pseudo-Boolean projection (D. and Belle 2021)
 - Weights not necessarily on literals
- Semiring programming (Belle and De Raedt 2020)

(Some of the) Many Ways to Count



(Unweighted) First-Order Model Counting

- Example formula:

$$\forall x \in \Delta. P(x) \vee Q(x).$$

- Let $\Delta := \{1, 2\}$.

- Interpretations: all subsets of

$$\{P(1), Q(1), P(2), Q(2)\}.$$

(Unweighted) First-Order Model Counting

- Example formula:

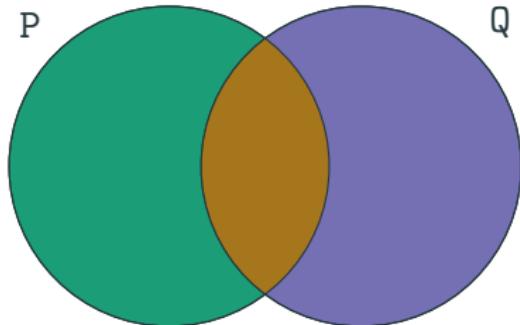
$$\forall x \in \Delta. P(x) \vee Q(x).$$

- Let $\Delta := \{1, 2\}$.

- Interpretations: all subsets of $\{P(1), Q(1), P(2), Q(2)\}$.

- Models:

$$\begin{array}{lll} \{P(1), P(2)\}, & \{P(1), Q(2)\}, & \{P(1), P(2), Q(2)\}, \\ \{Q(1), P(2)\}, & \{Q(1), Q(2)\}, & \{Q(1), P(2), Q(2)\}, \\ \{P(1), Q(1), P(2)\}, & \{P(1), Q(1), Q(2)\}, & \{P(1), Q(1), P(2), Q(2)\}. \end{array}$$



Intuition

- Each 1-ary predicate is like a subset.
- For $n > 1$, each n -ary predicate is like a relation.
- FOMC counts combinations of relations.

Exact Algorithms for FOMC

- **ForcLift** (Van den Broeck et al. 2011)
 - knowledge compilation to **FO d-DNNF**
- **L2C** (Kazemi and Poole 2016)
 - knowledge compilation to **C++ code**
- **Alchemy** (Gogate and Domingos 2016)
 - **DPLL**-style search
- **FastWFOMC** (van Bremen and Kuželka 2021)
 - knowledge compilation to **sd-DNNF**

Exact Algorithms for FOMC

- **ForcLift** (Van den Broeck et al. 2011)
 - knowledge compilation to **FO d-DNNF**
- **L2C** (Kazemi and Poole 2016)
 - knowledge compilation to **C++** code
- **Alchemy** (Gogate and Domingos 2016)
 - **DPLL**-style search
- **FastWFOMC** (van Bremen and Kuželka 2021)
 - knowledge compilation to **sd-DNNF**

Our Contribution



+



=



ForcLift

Recursion

Crane

ForcLift and First-Order Knowledge Compilation

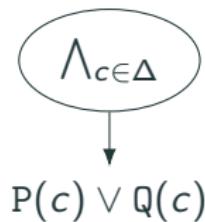
$$\forall x \in \Delta. \ P(x) \vee Q(x)$$

ForcLift and First-Order Knowledge Compilation

$$\forall x \in \Delta. \ P(x) \vee Q(x)$$

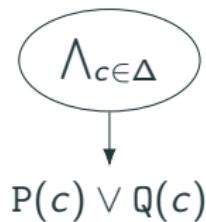
Independent partial grounding (introduces a constant $c \in \Delta$)

ForcLift and First-Order Knowledge Compilation



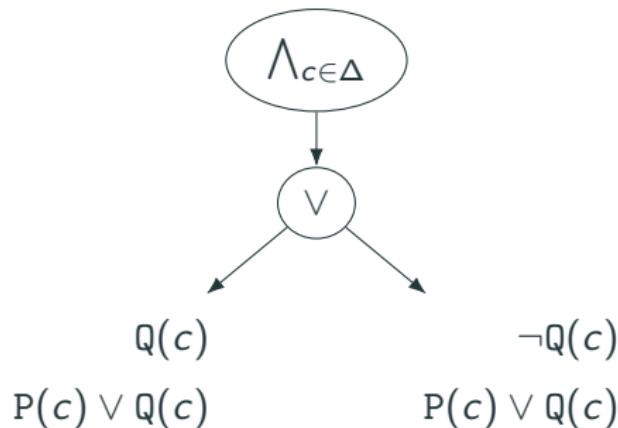
Independent partial grounding (introduces a constant $c \in \Delta$)

ForcLift and First-Order Knowledge Compilation



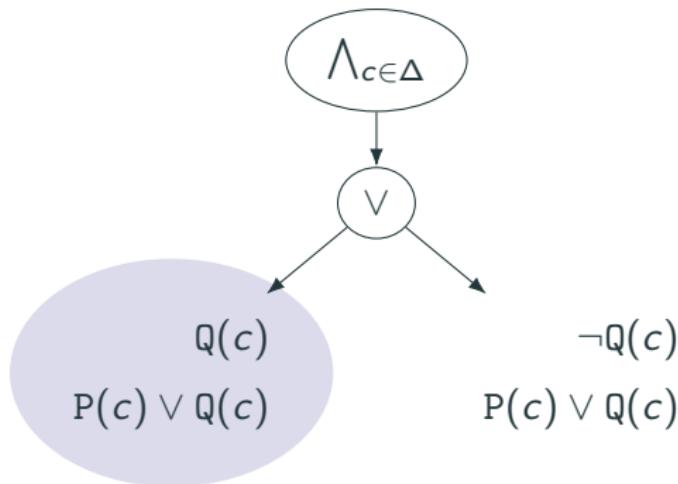
Shannon decomposition (a.k.a. Boole's expansion theorem) on $Q(c)$

ForcLift and First-Order Knowledge Compilation



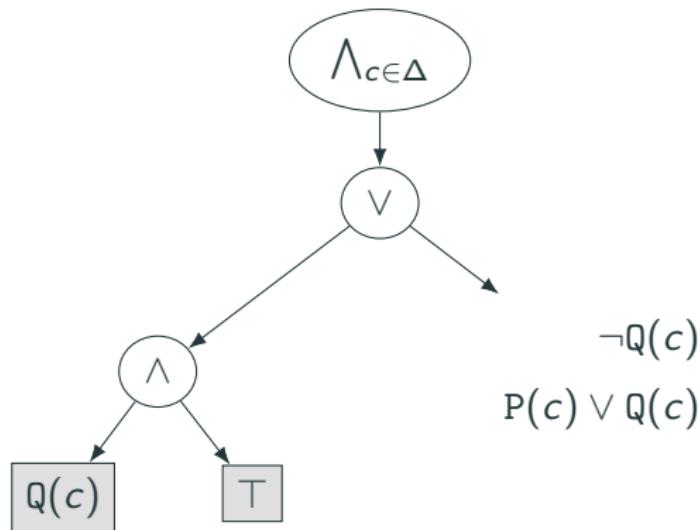
Shannon decomposition (a.k.a. Boole's expansion theorem) on $\text{Q}(c)$

ForcLift and First-Order Knowledge Compilation



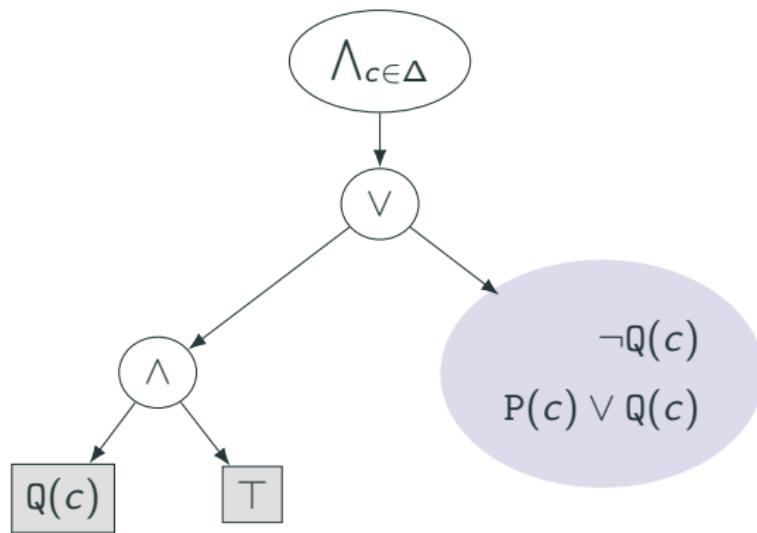
Positive unit propagation of $Q(c)$

ForcLift and First-Order Knowledge Compilation



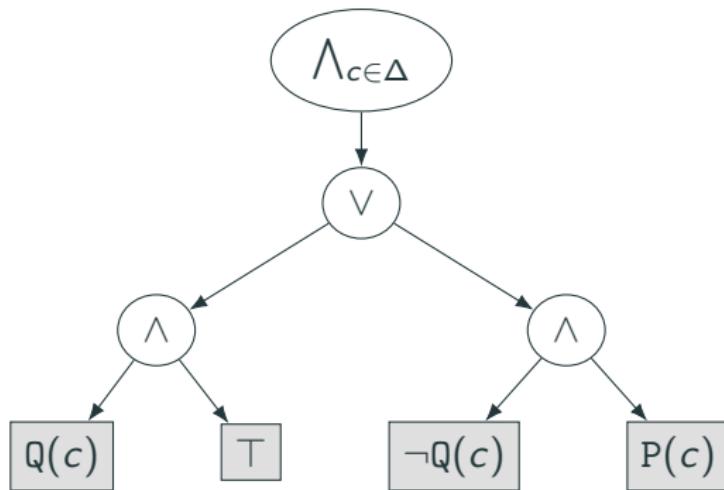
Positive unit propagation of $Q(c)$

ForcLift and First-Order Knowledge Compilation



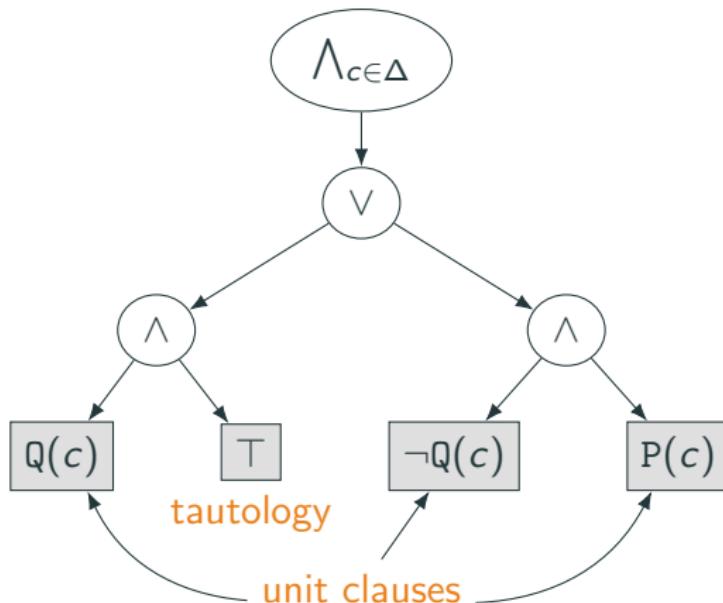
Negative unit propagation of $\neg Q(c)$

ForcLift and First-Order Knowledge Compilation



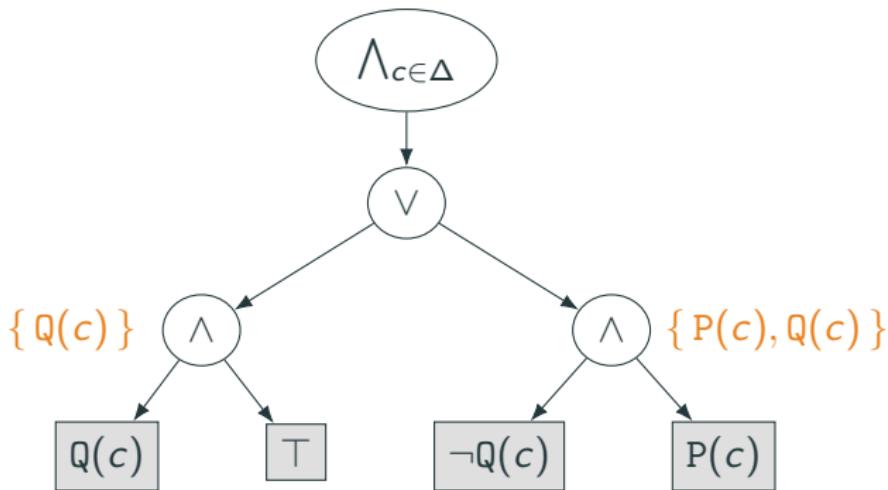
Negative unit propagation of $\neg Q(c)$

ForcLift and First-Order Knowledge Compilation



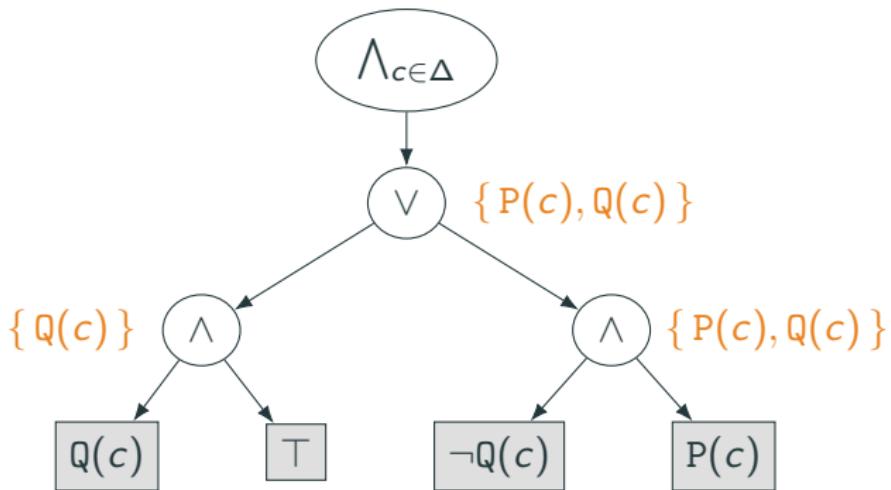
Compilation is complete ✓

ForcLift and First-Order Knowledge Compilation



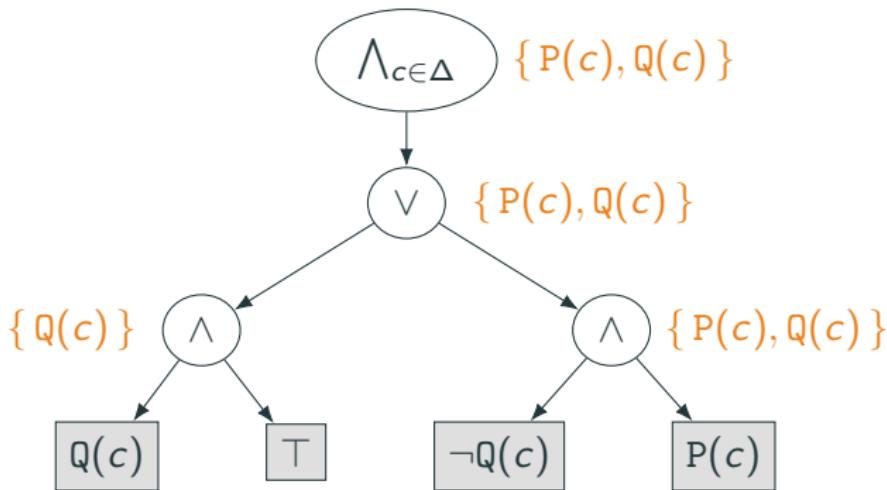
Smoothing: propagating atoms upwards

ForcLift and First-Order Knowledge Compilation



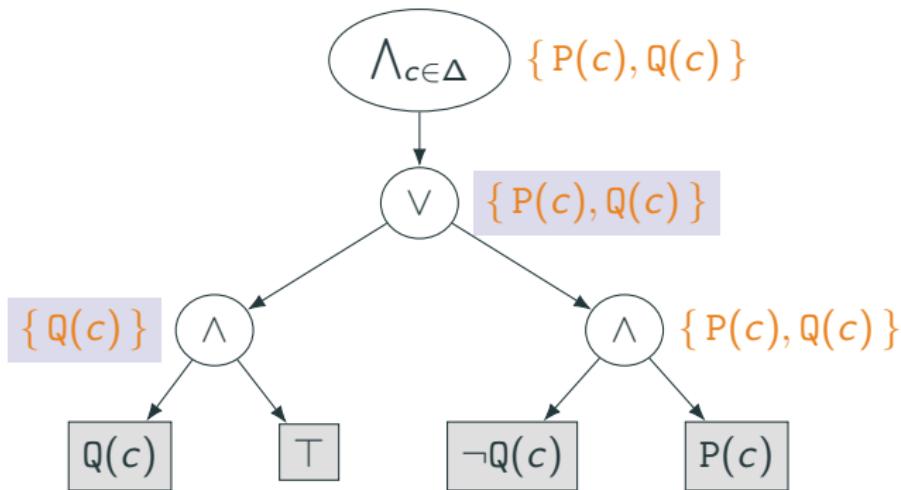
Smoothing: propagating atoms upwards

ForcLift and First-Order Knowledge Compilation



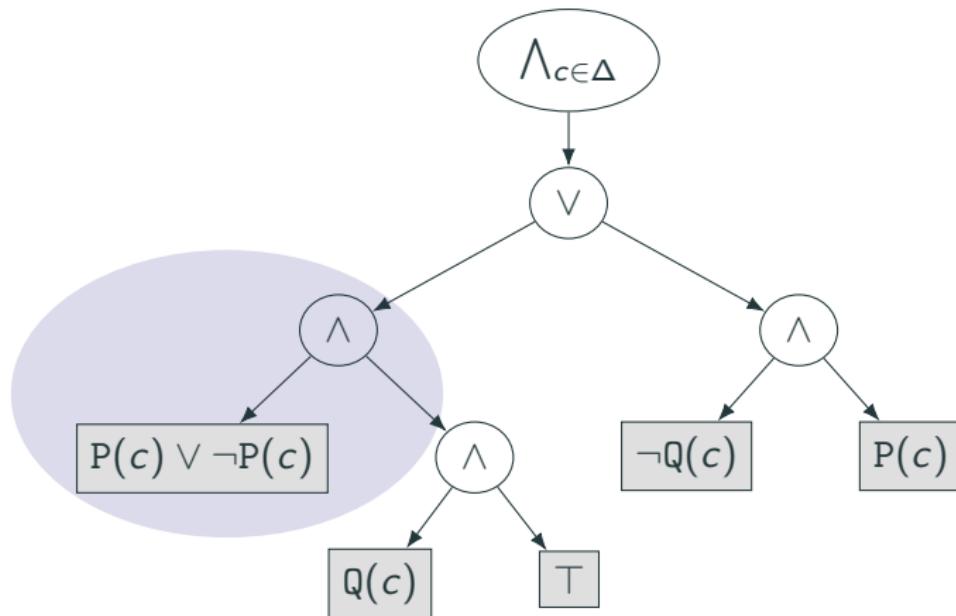
Smoothing: propagating atoms upwards

ForcLift and First-Order Knowledge Compilation



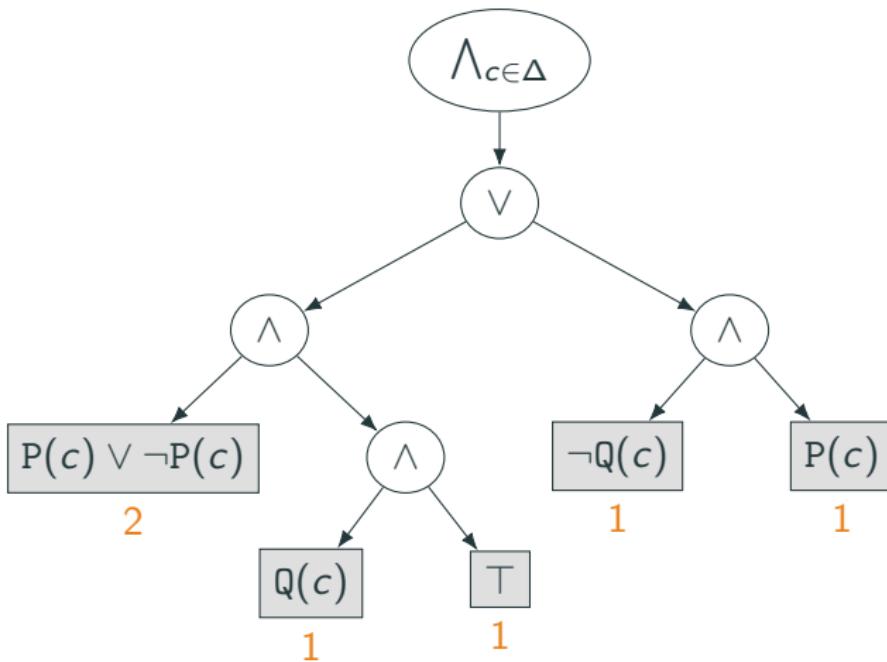
Smoothing: adding new atoms

ForcLift and First-Order Knowledge Compilation



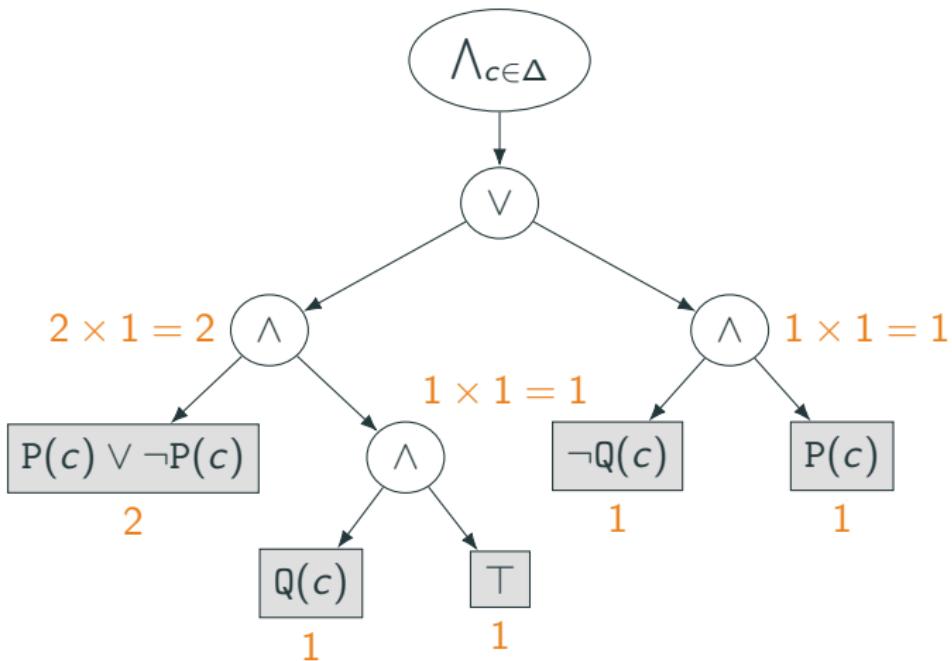
Smoothing: adding new atoms

ForcLift and First-Order Knowledge Compilation



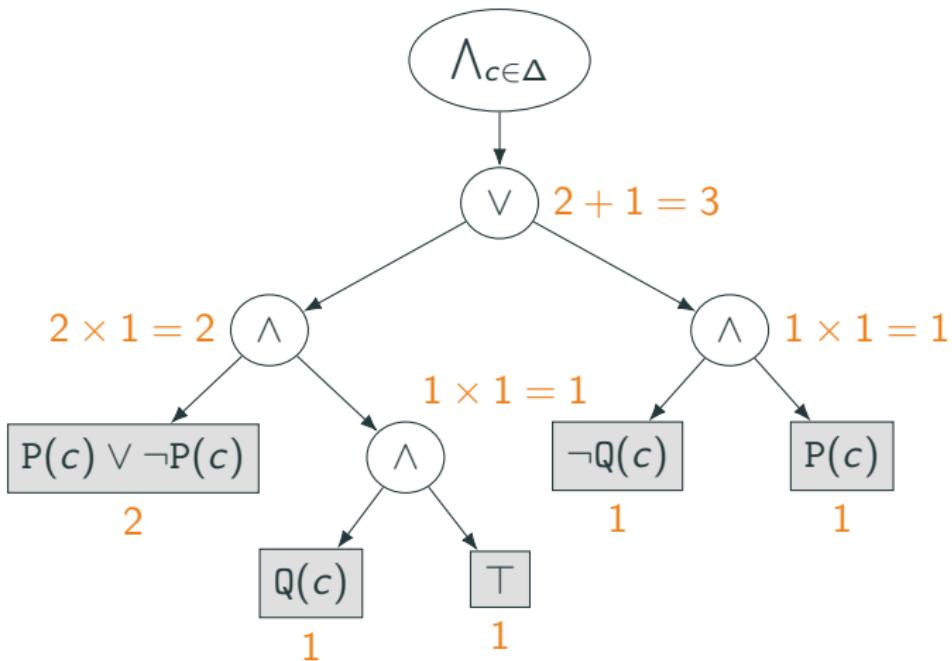
Propagating the model count

ForcLift and First-Order Knowledge Compilation



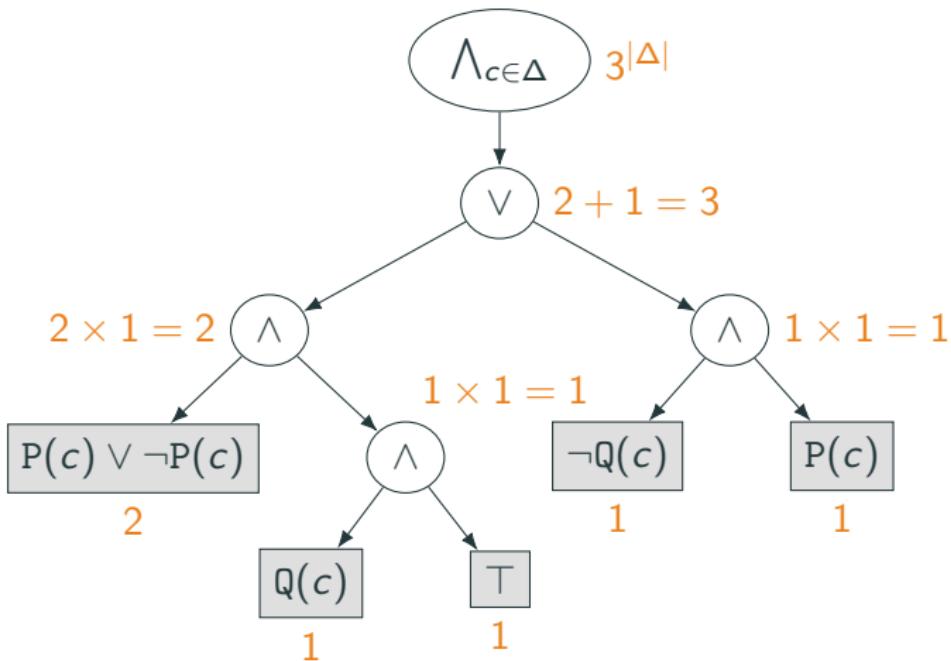
Propagating the model count

ForcLift and First-Order Knowledge Compilation



Propagating the model count

ForcLift and First-Order Knowledge Compilation



Propagating the model count

A (Slightly) More Complicated Example

Suppose this room has n seats, and there are $m \leq n$ people in the audience. How many ways are there to seat everyone?

A (Slightly) More Complicated Example

Suppose this room has n seats, and there are $m \leq n$ people in the audience. How many ways are there to seat everyone?

More explicitly, we assume that:

- each attendee gets exactly one seat,
- and a seat can accommodate at most one person.

A (Slightly) More Complicated Example

Suppose this room has n seats, and there are $m \leq n$ people in the audience. How many ways are there to seat everyone?

More explicitly, we assume that:

- each attendee gets exactly one seat,
- and a seat can accommodate at most one person.

Answer: $n^m = n \cdot (n - 1) \cdots (n - m + 1)$.

Note: this problem is equivalent to counting $[m] \rightarrow [n]$ injections.

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
 - such that $|\Gamma| = m$, and $|\Delta| = n$.
- Let $P \subseteq \Gamma \times \Delta$ be a relation (i.e., predicate) over Γ and Δ .
- We can describe all of the constraints in first-order logic:

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
 - such that $|\Gamma| = m$, and $|\Delta| = n$.
- Let $P \subseteq \Gamma \times \Delta$ be a relation (i.e., predicate) over Γ and Δ .
- We can describe all of the constraints in first-order logic:
 - each attendee gets a seat (i.e., at least one seat)

$$\forall x \in \Gamma. \exists y \in \Delta. P(x, y) \tag{1}$$

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
 - such that $|\Gamma| = m$, and $|\Delta| = n$.
- Let $P \subseteq \Gamma \times \Delta$ be a relation (i.e., predicate) over Γ and Δ .
- We can describe all of the constraints in first-order logic:
 - each attendee gets a seat (i.e., at least one seat)

$$\forall x \in \Gamma. \exists y \in \Delta. P(x, y) \quad (1)$$

- one person cannot occupy multiple seats

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z \quad (2)$$

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
 - such that $|\Gamma| = m$, and $|\Delta| = n$.
- Let $P \subseteq \Gamma \times \Delta$ be a relation (i.e., predicate) over Γ and Δ .
- We can describe all of the constraints in first-order logic:
 - each attendee gets a seat (i.e., at least one seat)

$$\forall x \in \Gamma. \exists y \in \Delta. P(x, y) \quad (1)$$

- one person cannot occupy multiple seats

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z \quad (2)$$

- one seat cannot accommodate multiple attendees

$$\forall w, x \in \Gamma. \forall y \in \Delta. P(w, y) \wedge P(x, y) \Rightarrow w = x \quad (3)$$

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
 - such that $|\Gamma| = m$, and $|\Delta| = n$.
- Let $P \subseteq \Gamma \times \Delta$ be a relation (i.e., predicate) over Γ and Δ .
- We can describe all of the constraints in first-order logic:
 - each attendee gets a seat (i.e., at least one seat)

$$\forall x \in \Gamma. \exists y \in \Delta. P(x, y) \quad (1)$$

- one person cannot occupy multiple seats

$$\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z \quad (2)$$

- one seat cannot accommodate multiple attendees

$$\forall w, x \in \Gamma. \forall y \in \Delta. P(w, y) \wedge P(x, y) \Rightarrow w = x \quad (3)$$

(1) and (2) constrain P to be a function, and (3) makes it injective.

Recursion



Back to Our Example

The following function counts injections:

$$f(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ and } n = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ f(m, n - 1) + m \times f(m - 1, n - 1) & \text{otherwise.} \end{cases}$$

Back to Our Example

The following function counts injections:

$$f(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ and } n = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ f(m, n - 1) + m \times f(m - 1, n - 1) & \text{otherwise.} \end{cases}$$

- $f(m, n)$ can be computed in $\Theta(mn)$ time
 - using dynamic programming.

Back to Our Example

The following function counts injections:

$$f(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ and } n = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ f(m, n - 1) + m \times f(m - 1, n - 1) & \text{otherwise.} \end{cases}$$

- $f(m, n)$ can be computed in $\Theta(mn)$ time
 - using dynamic programming.
- Optimal time complexity to compute n^m is $\Theta(m)$.
- But $\Theta(mn)$ is still much better than translating to propositional logic and solving a $\#P$ -complete problem.

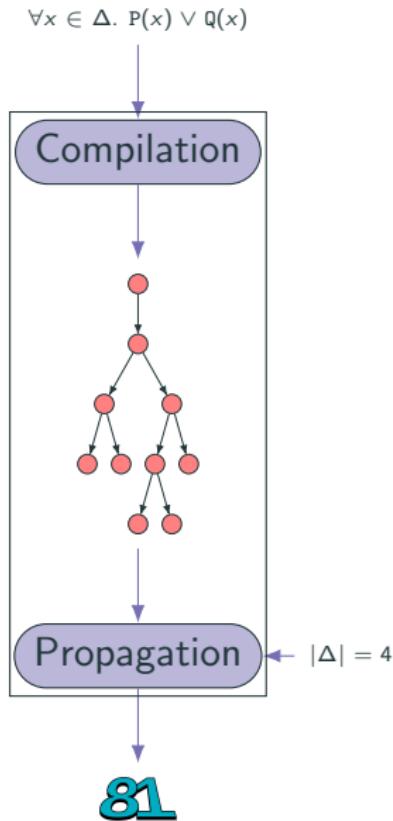
Back to Our Example

The following function counts injections:

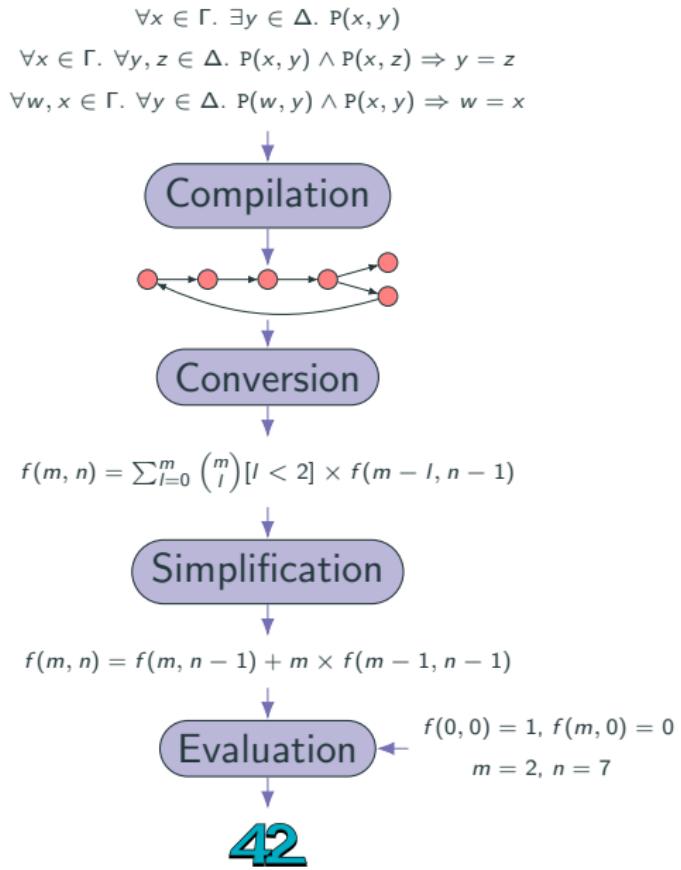
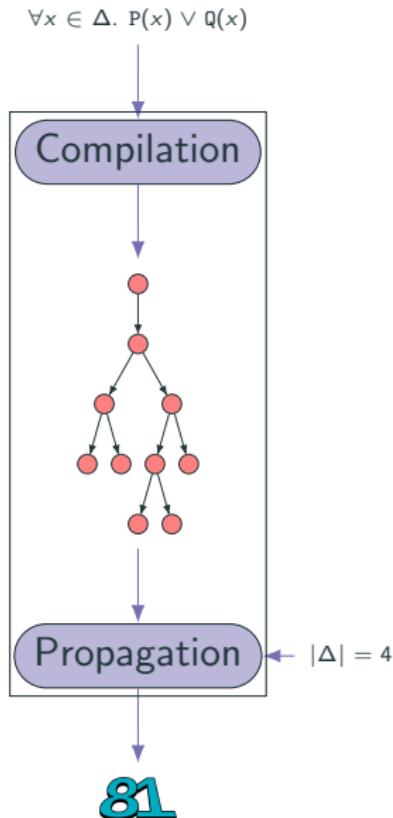
$$f(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ and } n = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ f(m, n - 1) + m \times f(m - 1, n - 1) & \text{otherwise.} \end{cases}$$

- $f(m, n)$ can be computed in $\Theta(mn)$ time
 - using dynamic programming.
- Optimal time complexity to compute n^m is $\Theta(m)$.
- But $\Theta(mn)$ is still much better than translating to propositional logic and solving a **#P-complete** problem.
- The rest of this talk is about how to construct such functions automatically.

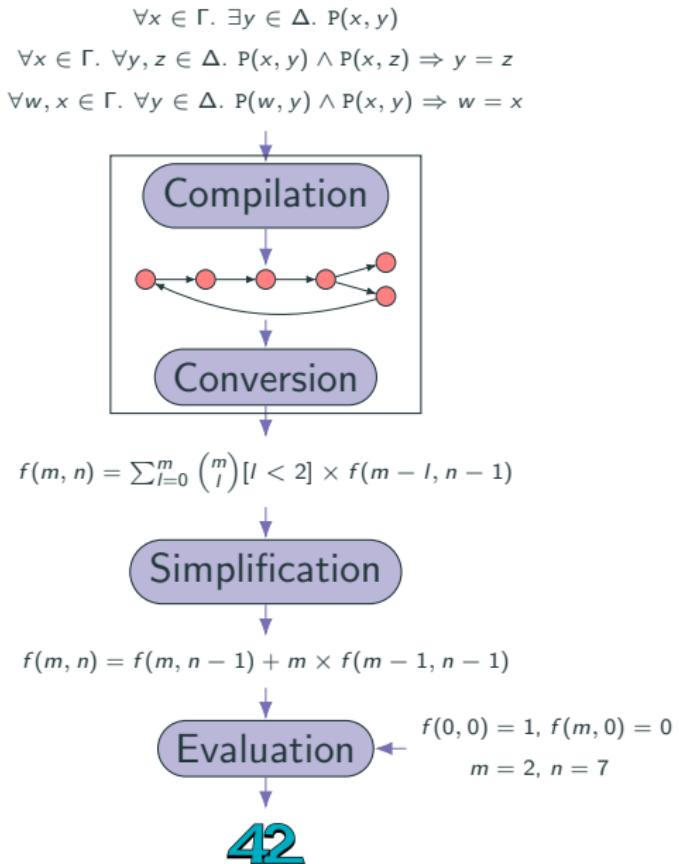
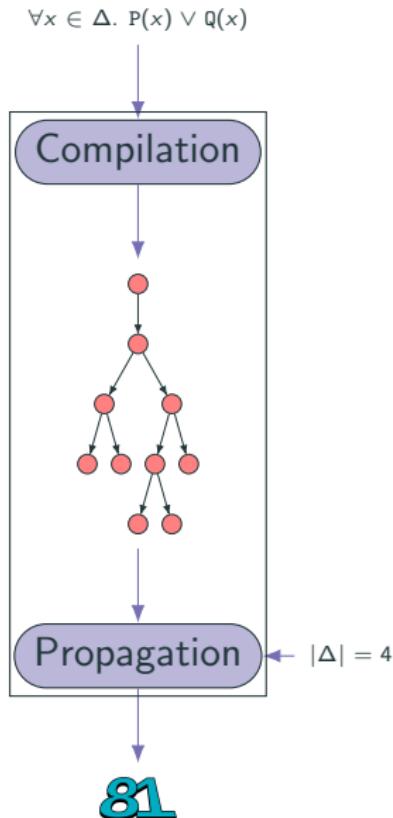
First-Order Knowledge Compilation: Before and After



First-Order Knowledge Compilation: Before and After



First-Order Knowledge Compilation: Before and After



Circuits vs Graphs

Circuits (Van den Broeck et al. 2011)...

- ... extend d-DNNF circuits (Darwiche 2001) for propositional knowledge compilation with **more node types**
- ... are **acyclic**.

Circuits vs Graphs

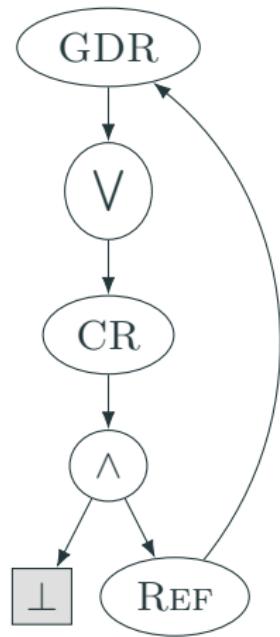
Circuits (Van den Broeck et al. 2011)...

- ... extend d-DNNF circuits (Darwiche 2001) for propositional knowledge compilation with **more node types**
- ... are **acyclic**.

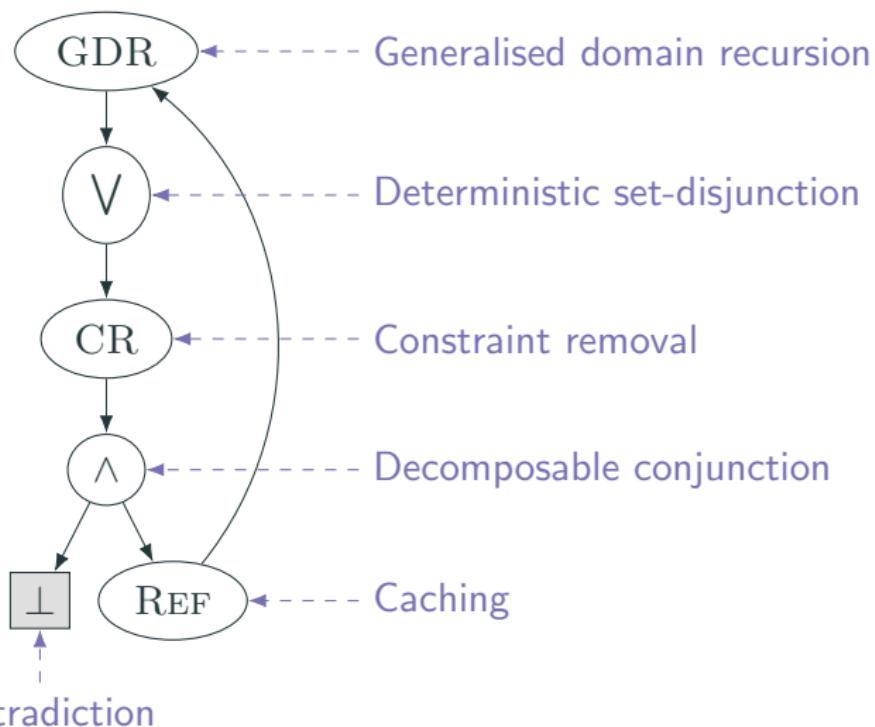
First-Order Computational Graphs (FCGs) are...
directed **acyclic** (weakly connected) graphs with:

- a single source,
- labelled nodes,
- and ordered outgoing edges.

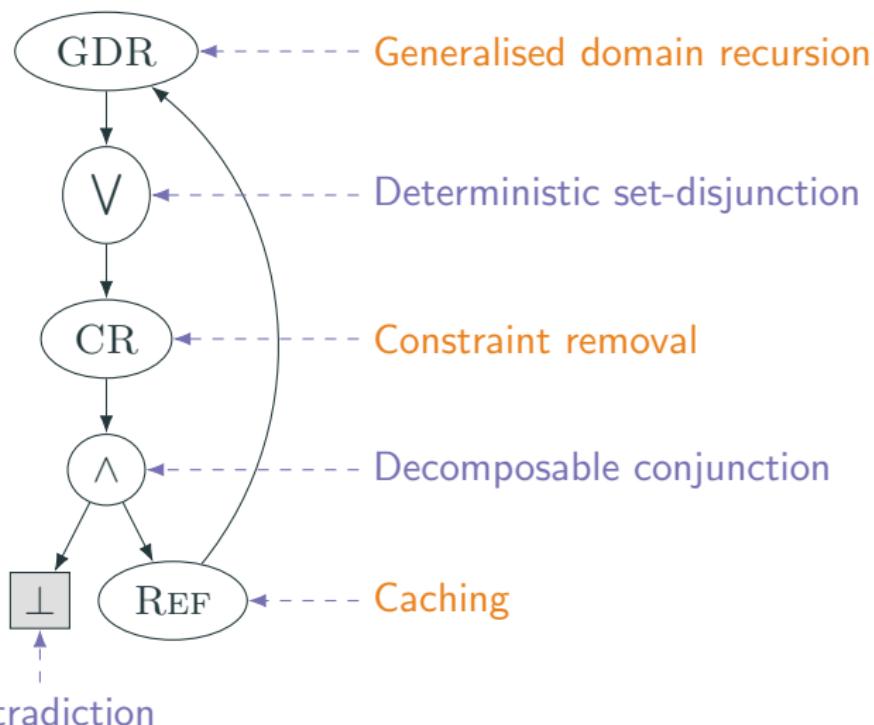
How to Interpret an FCG



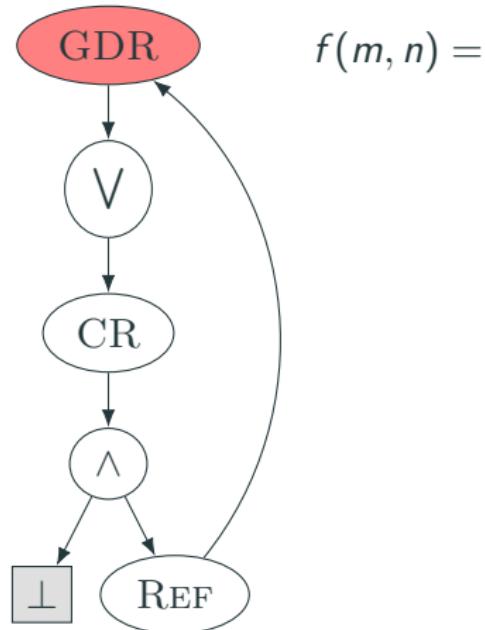
How to Interpret an FCG



How to Interpret an FCG

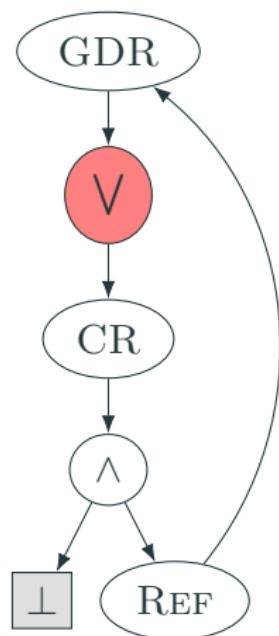


How to Interpret an FCG



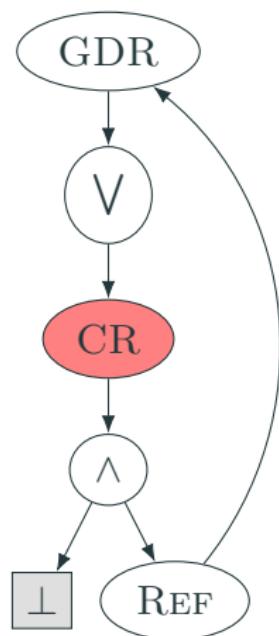
$$f(m, n) =$$

How to Interpret an FCG



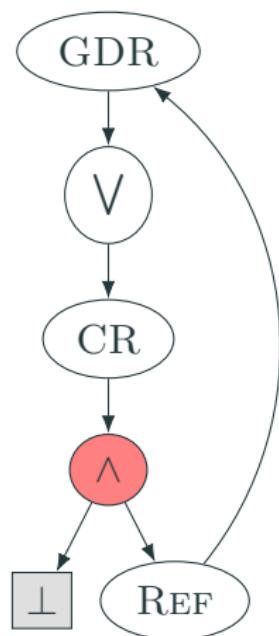
$$f(m, n) = \sum_{l=0}^m \binom{m}{l}$$

How to Interpret an FCG



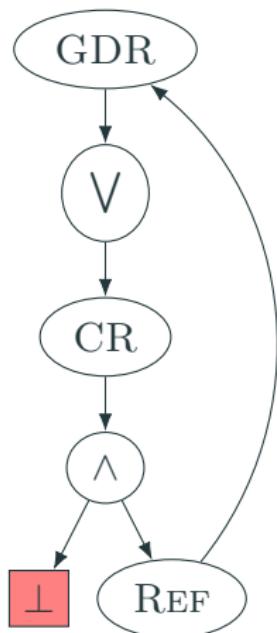
$$f(m, n) = \sum_{l=0}^m \binom{m}{l}$$

How to Interpret an FCG



$$f(m, n) = \sum_{l=0}^m \binom{m}{l} \quad \times$$

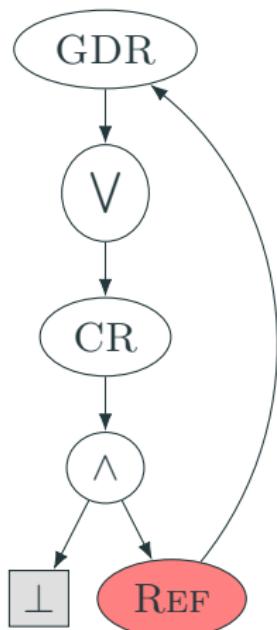
How to Interpret an FCG



$$f(m, n) = \sum_{l=0}^m \binom{m}{l} [l < 2] \times$$

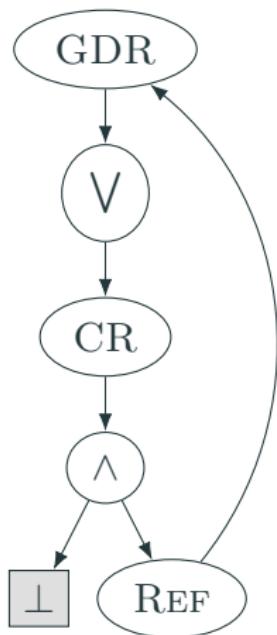
$$[\phi] = \begin{cases} 1 & \text{if } \phi \\ 0 & \text{if } \neg\phi \end{cases}$$

How to Interpret an FCG



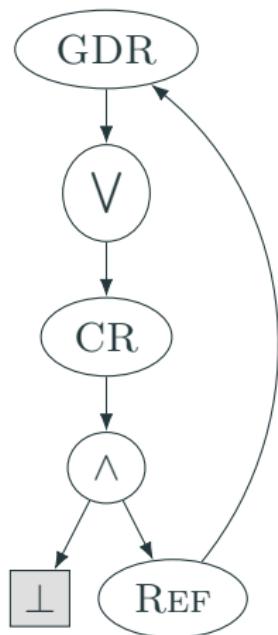
$$f(m, n) = \sum_{l=0}^m \binom{m}{l} [l < 2] \times f(m - l, n - 1)$$

How to Interpret an FCG



$$\begin{aligned}f(m, n) &= \sum_{l=0}^m \binom{m}{l} [l < 2] \times f(m - l, n - 1) \\&= \binom{m}{0} \times f(m - 0, n - 1) \\&\quad + \binom{m}{1} \times f(m - 1, n - 1)\end{aligned}$$

How to Interpret an FCG



$$\begin{aligned}f(m, n) &= \sum_{l=0}^m \binom{m}{l} [l < 2] \times f(m - l, n - 1) \\&= \binom{m}{0} \times f(m - 0, n - 1) \\&\quad + \binom{m}{1} \times f(m - 1, n - 1) \\&= f(m, n - 1) + m \times f(m - 1, n - 1)\end{aligned}$$

Compilation: How FCGs Are Built

Definition

A **(compilation) rule** is a function that takes a **formula** and returns a set of **(G, L)** pairs, where

- G is a (possibly incomplete) FCG,
- and L is a list of formulas.

The formulas in L are then **compiled**, and the resulting FCGs are **inserted** into G according to a **set order**.

Example Compilation Rule: Independence

Input formula:

$$(\forall x, y \in \Omega. x = y) \wedge \quad (1)$$

$$(\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z) \wedge \quad (2)$$

$$(\forall w, x \in \Gamma. \forall y \in \Delta. P(w, y) \wedge P(x, y) \Rightarrow w = x) \quad (3)$$

Example Compilation Rule: Independence

Input formula:

$$(\forall x, y \in \Omega. x = y) \wedge \quad (1)$$

$$(\forall x \in \Gamma. \forall y, z \in \Delta. P(x, y) \wedge P(x, z) \Rightarrow y = z) \wedge \quad (2)$$

$$(\forall w, x \in \Gamma. \forall y \in \Delta. P(w, y) \wedge P(x, y) \Rightarrow w = x) \quad (3)$$

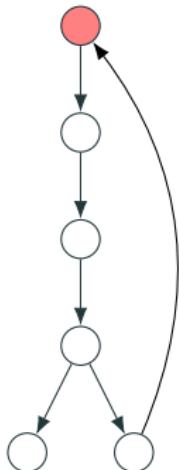
The independence compilation rule returns one (G, L) pair:



New Rule 1/3: Generalised Domain Recursion

Example

Input formula:



$$\forall x \in \Gamma. \forall y, z \in \Delta. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

Output formula (with a new constant $c \in \Gamma$):

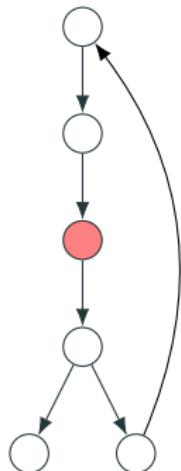
$$\forall y, z \in \Delta. y \neq z \Rightarrow \neg P(c, y) \vee \neg P(c, z)$$

$$\begin{aligned} \forall x \in \Gamma. \forall y, z \in \Delta. & x \neq c \wedge y \neq z \Rightarrow \\ & \neg P(x, y) \vee \neg P(x, z) \end{aligned}$$

New Rule 2/3: Constraint Removal

Example

Input formula (with a constant $c \in \Gamma$):



$$\forall x \in \Gamma. \forall y, z \in \Delta. x \neq c \wedge y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

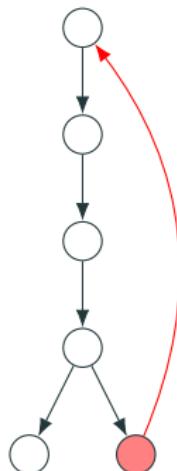
$$\forall w, x \in \Gamma. \forall y \in \Delta. w \neq c \wedge x \neq c \wedge w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

Output formula (with a new domain $\Gamma' := \Gamma \setminus \{c\}$):

$$\forall x \in \Gamma'. \forall y, z \in \Delta. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma'. \forall y \in \Delta. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

New Rule 3/3: Identifying Possibilities for Recursion



Goal

Check if the input formula is equivalent (up to domains) to a previously encountered formula.

Outline

1. Consider pairs of 'similar' clauses.
2. Consider bijections between their sets of variables.
3. Extend each such bijection to a map between sets of domains.
4. If the bijection makes the clauses equivalent, and the domain map is compatible with previous domain maps, move on to another pair of clauses.

How These Rules Fit Together (1/5)

$$\forall x \in \Gamma. \forall y, z \in \Delta. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma. \forall y \in \Delta. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$



Generalised domain recursion



$$\forall y, z \in \Delta. y \neq z \Rightarrow \neg P(c, y) \vee \neg P(c, z)$$

$$\forall x \in \Gamma. \forall y, z \in \Delta. y \neq z \wedge x \neq c \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall x \in \Gamma. \forall y \in \Delta. x \neq c \Rightarrow \neg P(c, y) \vee \neg P(x, y)$$

$$\forall w \in \Gamma. \forall y \in \Delta. w \neq c \Rightarrow \neg P(w, y) \vee \neg P(c, y)$$

$$\forall w, x \in \Gamma. \forall y \in \Delta. w \neq x \wedge w \neq c \wedge x \neq c \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

How These Rules Fit Together (2/5)

$$\forall y, z \in \Delta. \ y \neq z \Rightarrow \neg P(c, y) \vee \neg P(c, z)$$

$$\forall x \in \Gamma. \ \forall y, z \in \Delta. \ y \neq z \wedge x \neq c \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall x \in \Gamma. \ \forall y \in \Delta. \ x \neq c \Rightarrow \neg P(c, y) \vee \neg P(x, y)$$

$$\forall w \in \Gamma. \ \forall y \in \Delta. \ w \neq c \Rightarrow \neg P(w, y) \vee \neg P(c, y)$$

$$\forall w, x \in \Gamma. \ \forall y \in \Delta. \ w \neq x \wedge w \neq c \wedge x \neq c \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$



Atom counting and unit propagation



$$\forall y, z \in \Delta^{\top}. \ y \neq z \Rightarrow \perp$$

$$\forall x \in \Gamma. \ \forall y, z \in \Delta^{\perp}. \ y \neq z \wedge x \neq c \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma. \ \forall y \in \Delta^{\perp}. \ w \neq x \wedge w \neq c \wedge x \neq c \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

How These Rules Fit Together (3/5)

$$\forall y, z \in \Delta^\top. y \neq z \Rightarrow \perp$$

$$\forall x \in \Gamma. \forall y, z \in \Delta^\perp. y \neq z \wedge x \neq c \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma. \forall y \in \Delta^\perp. w \neq x \wedge w \neq c \wedge x \neq c \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$



Constraint removal



$$\forall y, z \in \Delta^\top. y \neq z \Rightarrow \perp$$

$$\forall x \in \Gamma'. \forall y, z \in \Delta^\perp. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma'. \forall y \in \Delta^\perp. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

How These Rules Fit Together (4/5)

$$\forall y, z \in \Delta^T. y \neq z \Rightarrow \perp$$

$$\forall x \in \Gamma'. \forall y, z \in \Delta^\perp. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma'. \forall y \in \Delta^\perp. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$



Independence

$$\forall y, z \in \Delta^T. y \neq z \Rightarrow \perp$$

$$\forall x \in \Gamma'. \forall y, z \in \Delta^\perp. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma'. \forall y \in \Delta^\perp. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

How These Rules Fit Together (5/5): Recursion

$$\forall x \in \Gamma'. \forall y, z \in \Delta^\perp. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma'. \forall y \in \Delta^\perp. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

||

$$\forall x \in \Gamma. \forall y, z \in \Delta. y \neq z \Rightarrow \neg P(x, y) \vee \neg P(x, z)$$

$$\forall w, x \in \Gamma. \forall y \in \Delta. w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

+

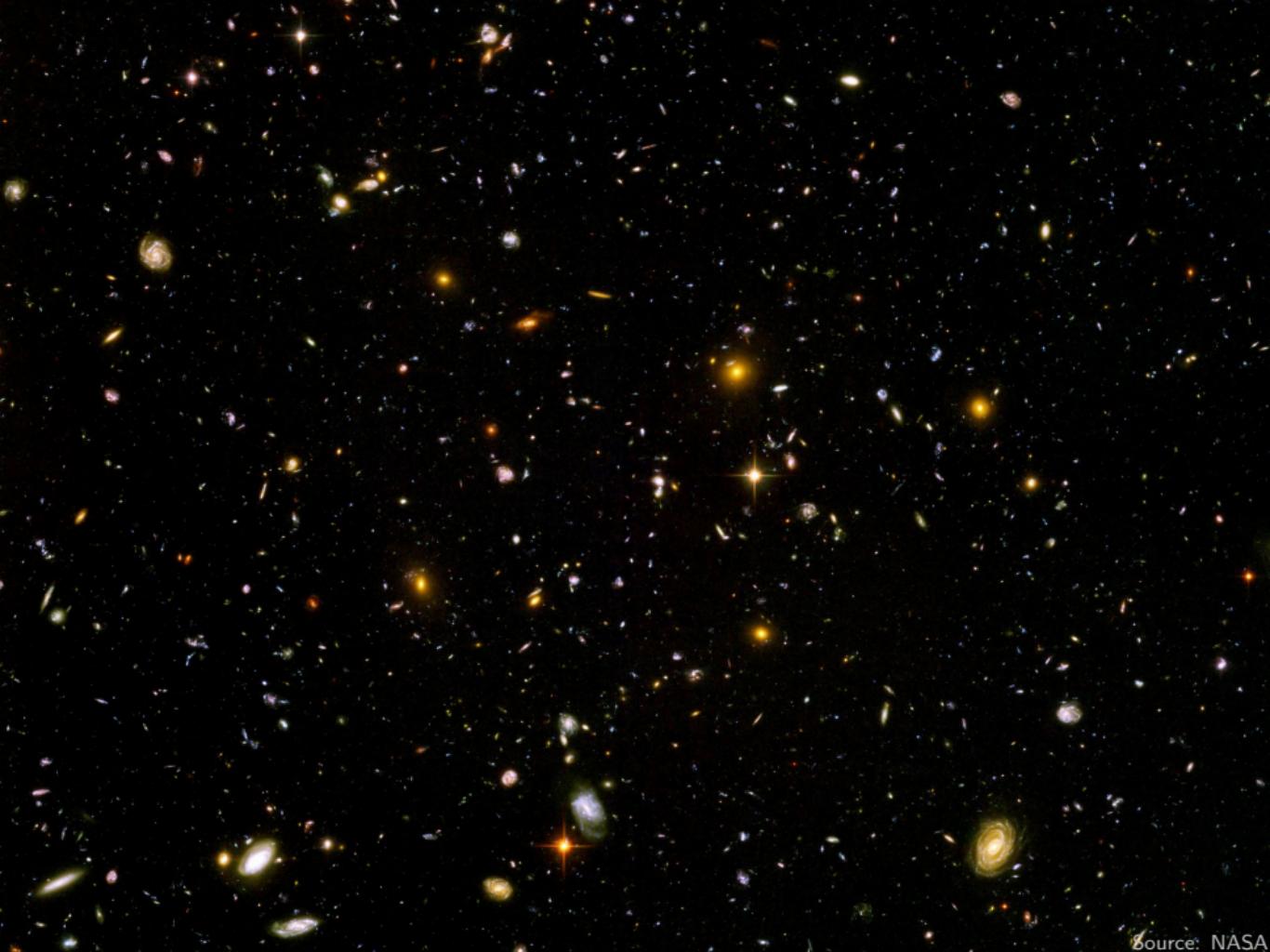
$$\{ \Gamma \mapsto \Gamma', \Delta \mapsto \Delta^\perp \}$$

Resulting Improvements to Counting Functions

Let Γ and Δ be two sets with cardinalities $|\Gamma| = m$ and $|\Delta| = n$.

Our new rules enable Crane to efficiently count $\Gamma \rightarrow \Delta$ functions such as:

- injections in $\Theta(mn)$ time
 - by hand: $\Theta(m)$
- partial injections in $\Theta(mn)$ time
 - by hand: $\Theta(\min\{m, n\}^2)$
- bijections in $\Theta(m)$ time
 - optimal!



Source: NASA

What Have We Learned?

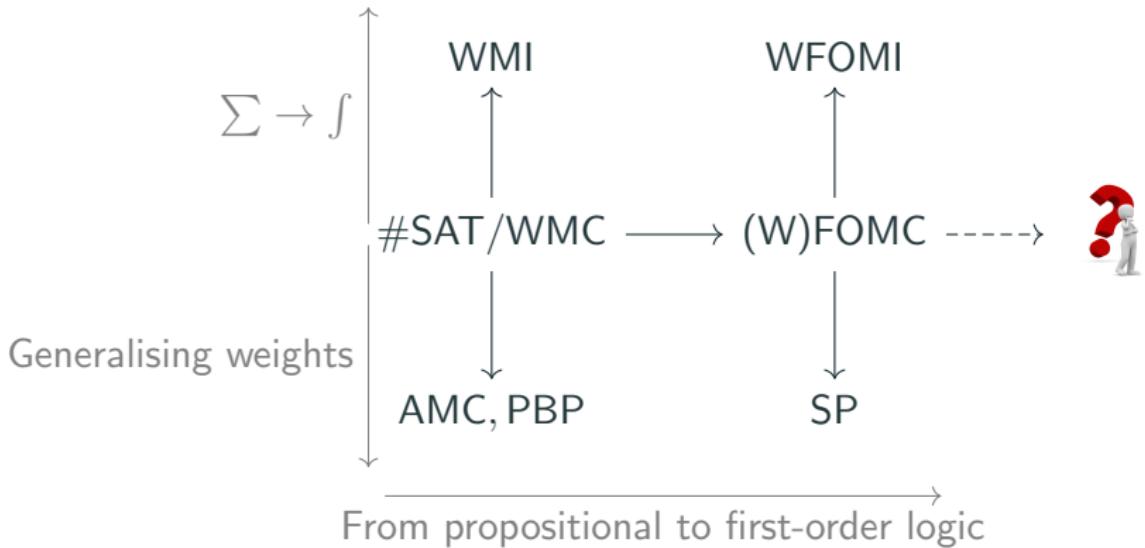
- Knowledge compilation can build **graphs with cycles**.
- Graphs (as well as circuits) define **functions**.
- Cycles can represent **recursive calls**, including:
 - mutual recursion
 - and function calls as complex as $f(n - k - 2)$.
- Recursion helps us solve counting problems that were previously beyond the reach of FOMC.
- In some cases, even if a polynomial-time solution is already known, Crane is able to find **more efficient solutions**, with a lower degree polynomial.

Future →

← Past



Beyond First-Order Logic



Algebraic Solutions to Parameterised Problems (1/2)

- Suppose we have a Markov logic network that models the probability P that some system will fail.
- Here:
 - domain sizes describe the numbers of various components,
 - and weights express probabilities that:
 - some component fails,
 - or some combination of failures leads to another failure.
- Crane can express P as a function of the domain sizes and weights.

Algebraic Solutions to Parameterised Problems (2/2)

With the help of a computer algebra system, we can then:

- determine how P scales with the number of users,
- find combinations of domain sizes that keep P below some threshold,
- find ranges of weights that keep P sufficiently small across a range of domain size values.

Algebraic Solutions to Parameterised Problems (2/2)

With the help of a computer algebra system, we can then:

- determine how P scales with the number of users,
- find combinations of domain sizes that keep P below some threshold,
- find ranges of weights that keep P sufficiently small across a range of domain size values.

Reasoning with **functions**



Reasoning with **numbers**