

# Recursive Solutions to First-Order Model Counting

---

Paulius Dilkas<sup>1</sup> Vaishak Belle<sup>2</sup>

TODO: date

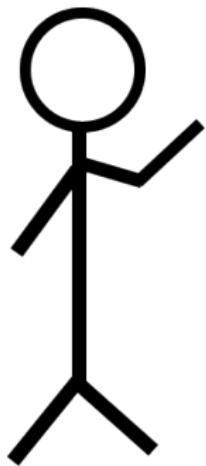
<sup>1</sup>National University of Singapore, Singapore

<sup>2</sup>University of Edinburgh, UK

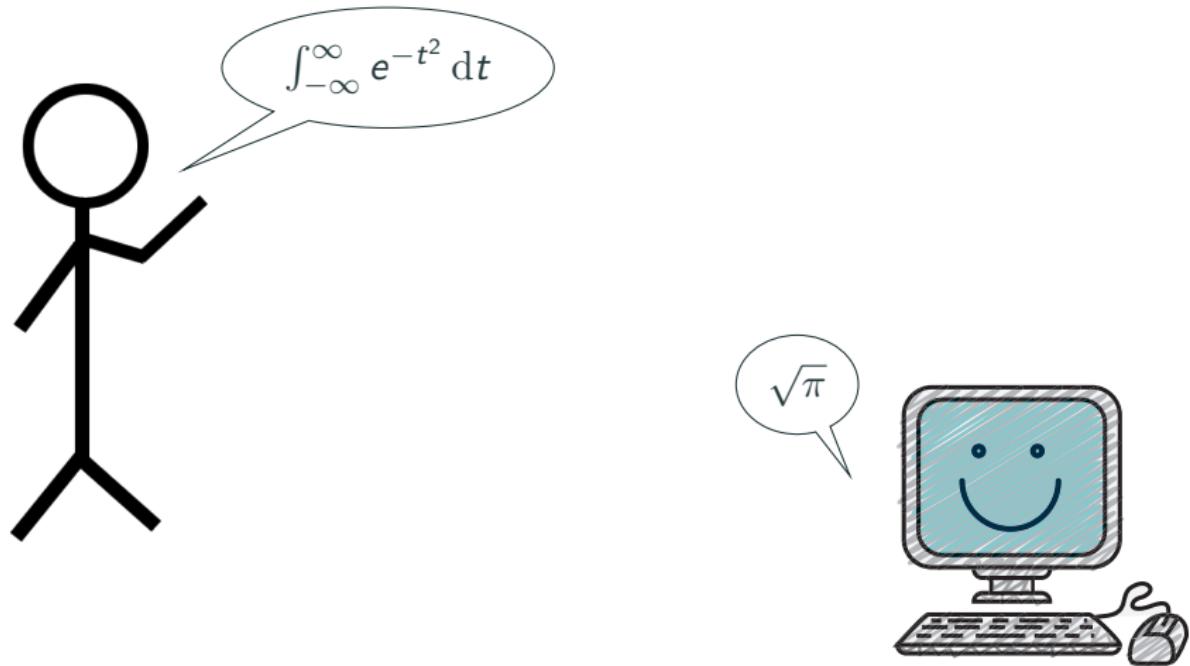


Engineering and  
Physical Sciences  
Research Council

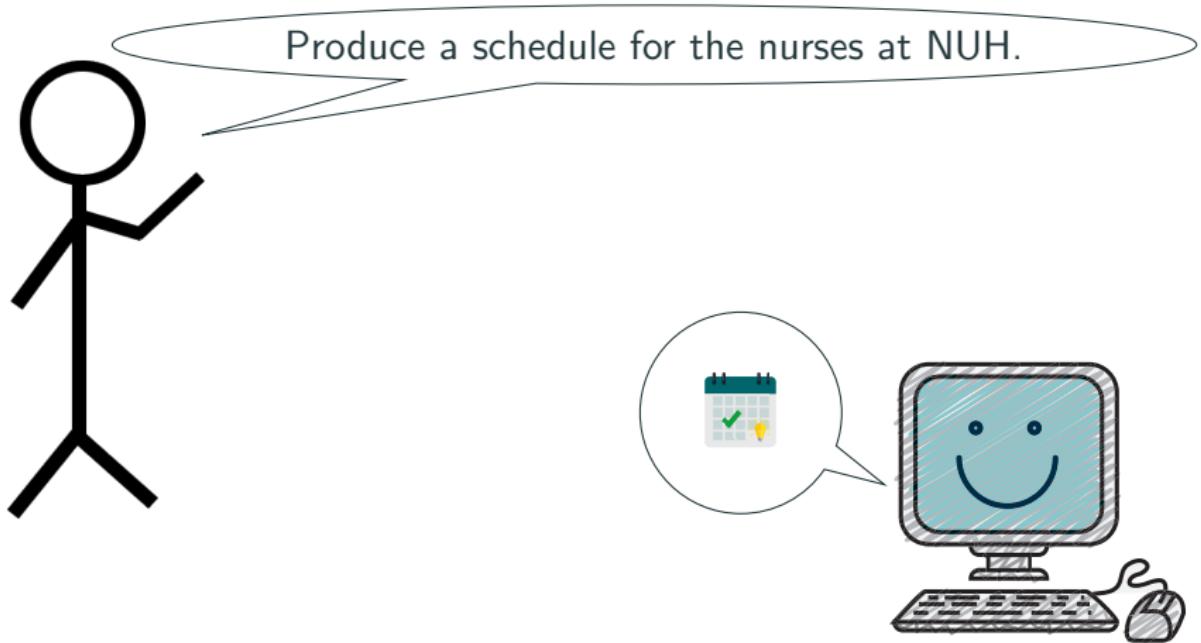
# What Computers Can and Cannot Do



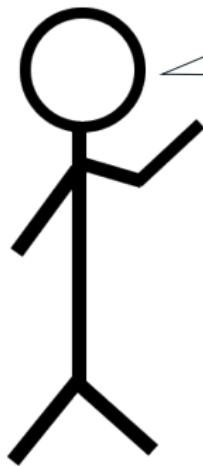
# What Computers Can and Cannot Do



# What Computers Can and Cannot Do



# What Computers Can and Cannot Do



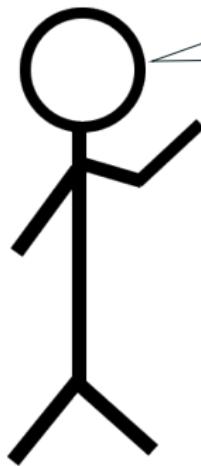
Describe this picture.



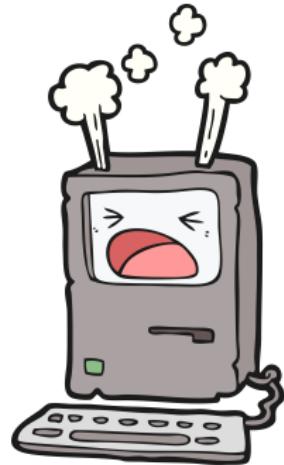
It's a cat with its eyes closed.



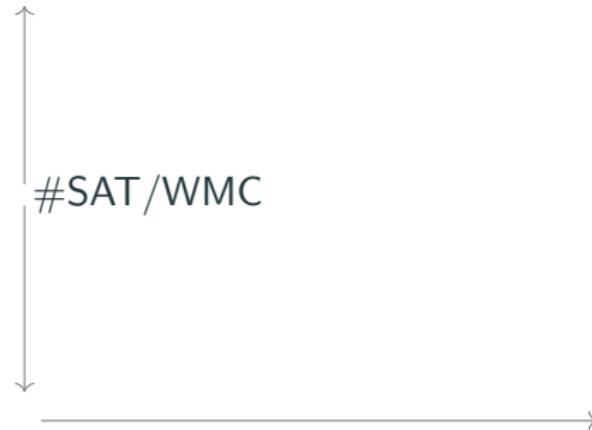
# What Computers Can and Cannot Do



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



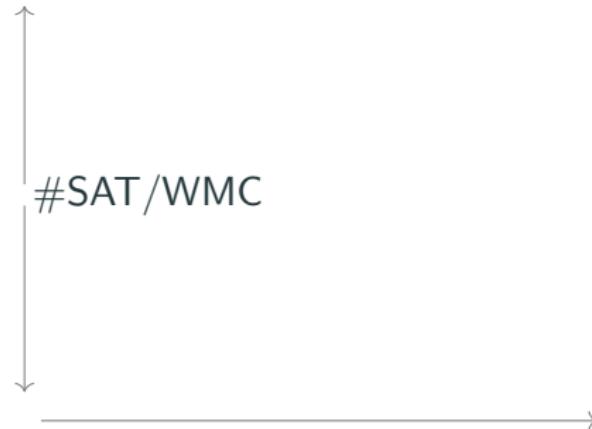
# (Some of the) Many Ways to Count



## #SAT (Valiant 1979)

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

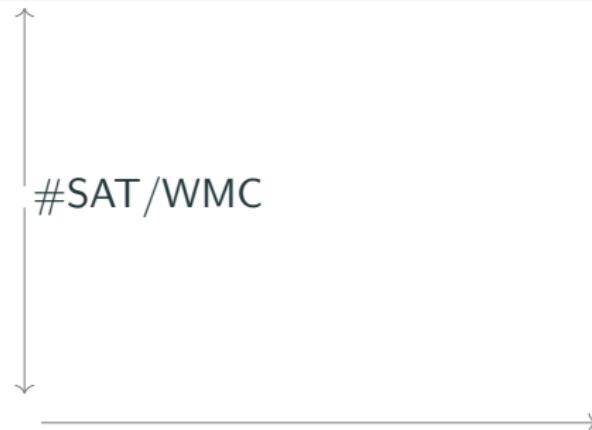
# (Some of the) Many Ways to Count



## #SAT (Valiant 1979)

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

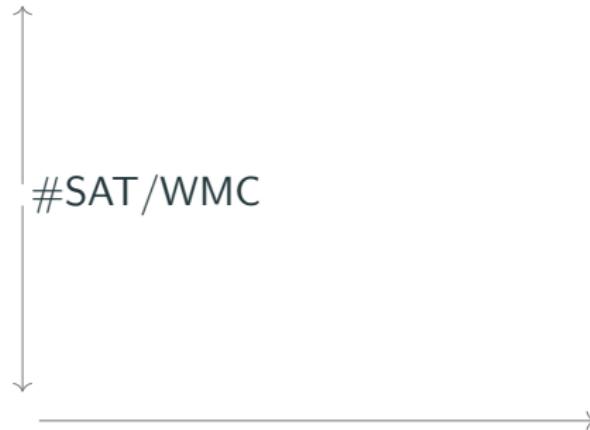
# (Some of the) Many Ways to Count



## Weighted Model Counting (Chavira and Darwiche 2008)

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

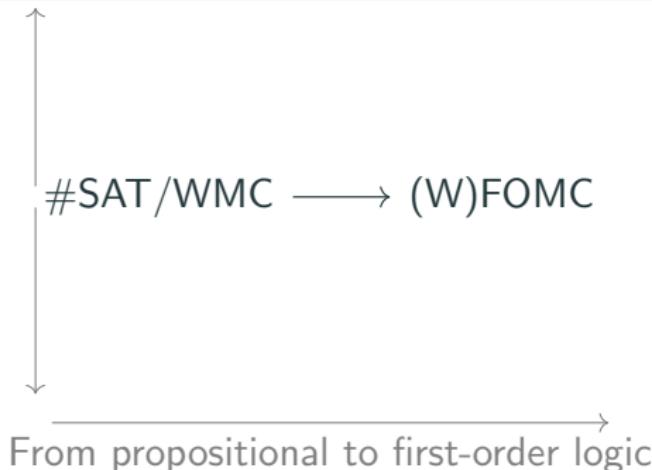
# (Some of the) Many Ways to Count



## 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

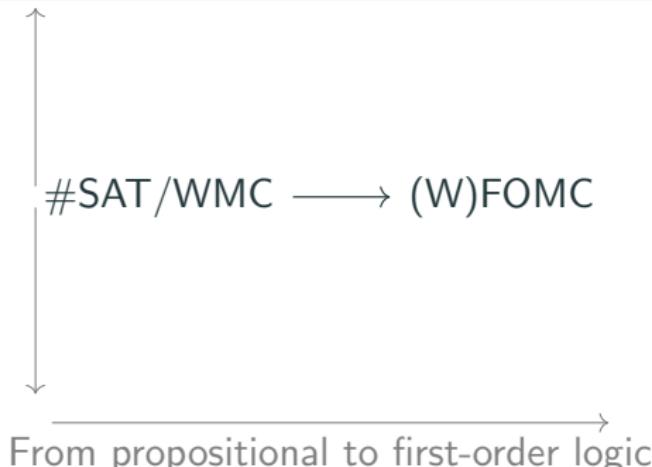


## (Weighted) (Symmetric) First-Order Model Counting

(Van den Broeck et al. 2011)

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

# (Some of the) Many Ways to Count

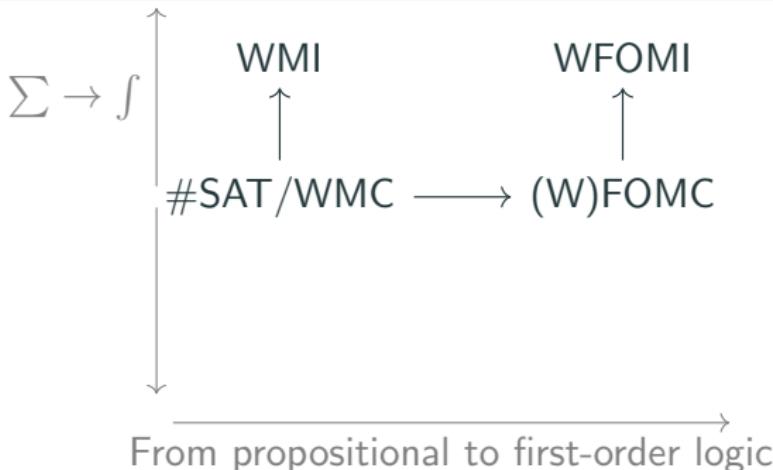


## (Weighted) (Symmetric) First-Order Model Counting

(Van den Broeck et al. 2011)

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

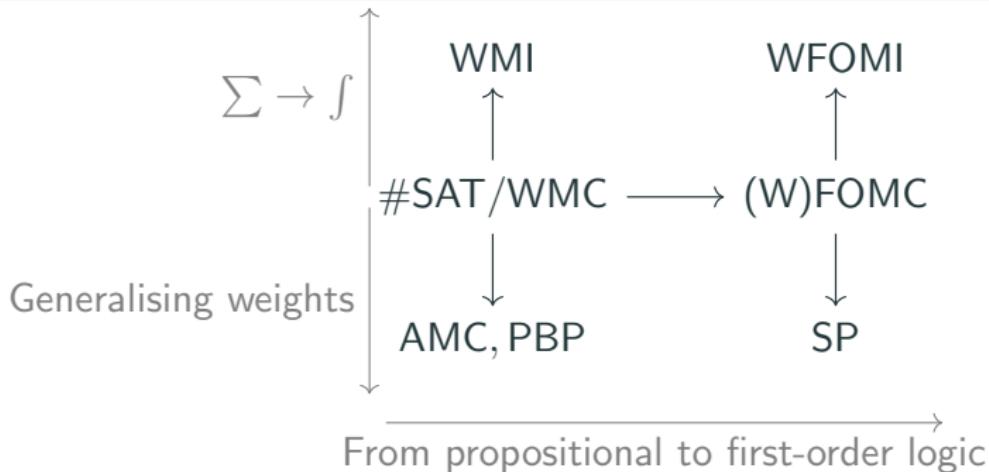
# (Some of the) Many Ways to Count



## 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 Elementary Counting

## A Counting Problem

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

# Some Elementary Counting

## A Counting Problem

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.

# Some Elementary Counting

## A Counting Problem

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  $\Gamma$  and  $\Delta$  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  $\Gamma$  and  $\Delta$ .
- We can describe all of the constraints in first-order logic:

## Let's Express This Problem in Logic!

- Let  $\Gamma$  and  $\Delta$  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  $\Gamma$  and  $\Delta$ .
- We can describe all of the constraints in first-order logic:
  - each attendee gets a seat (i.e., at least one seat)

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

# Let's Express This Problem in Logic!

- Let  $\Gamma$  and  $\Delta$  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  $\Gamma$  and  $\Delta$ .
- We can describe all of the constraints in first-order logic:
  - each attendee gets a seat (i.e., at least one seat)

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

- one person cannot occupy multiple seats

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

# Let's Express This Problem in Logic!

- Let  $\Gamma$  and  $\Delta$  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  $\Gamma$  and  $\Delta$ .
- We can describe all of the constraints in first-order logic:
  - each attendee gets a seat (i.e., at least one seat)

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

- one person cannot occupy multiple seats

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

- one seat cannot accommodate multiple attendees

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

# Let's Express This Problem in Logic!

- Let  $\Gamma$  and  $\Delta$  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  $\Gamma$  and  $\Delta$ .
- We can describe all of the constraints in first-order logic:
  - each attendee gets a seat (i.e., at least one seat)

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

- one person cannot occupy multiple seats

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

- one seat cannot accommodate multiple attendees

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

The first two sentences constrain  $P$  to be a function, and the last one makes it injective.

# Overview of the Problem

- First-order model counting (FOMC) is the problem of counting the models of a sentence in first-order logic.
- The (symmetric) weighted variation of the problem adds weights (e.g., probabilities) to predicates.
  - It is used for efficient probabilistic inference in relational models such as Markov logic networks.
- None of the (implemented) (W)FOMC algorithms are able to count, e.g., injective and bijective functions.

## Claim

This shortcoming can be addressed via support for recursive functions.

# Recursion



## Back to Our Example

For instance, the following function counts injections

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

## Back to Our Example

For instance, the following function counts injections

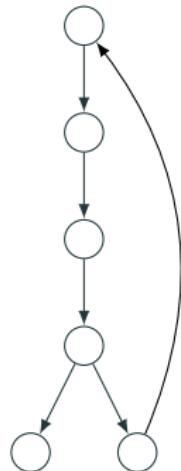
$$f(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ and } n = 0 \\ 0 & \text{if } m > 0 \text{ and } n = 0 \\ f(m, n - 1) + mf(m - 1, n - 1) & \text{otherwise.} \end{cases}$$

- $f$  can be computed in  $\Theta(mn)$  time (via 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 running a WMC algorithm.
- The rest of this talk is about how such functions can be found automatically.

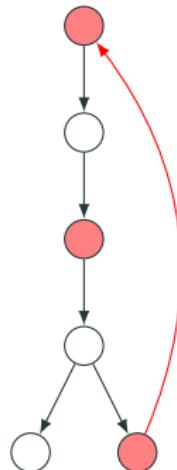
# First-Order Knowledge Compilation with ForcLift

## Workflow Before

1. Compile the formula to a **circuit**
2. Evaluate to get the answer



# First-Order Knowledge Compilation with ForcLift



## Workflow Before

1. Compile the formula to a **circuit**
2. Evaluate to get the answer

## Workflow After

1. Compile the formula to a **graph**
2. Extract the definitions of functions
3. Simplify
4. Supplement with **base cases**
5. Evaluate to get the answer

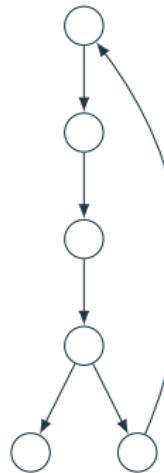
## More Formally...

### Definition

A first-order deterministic decomposable negation normal form computational graph (FCG) is a

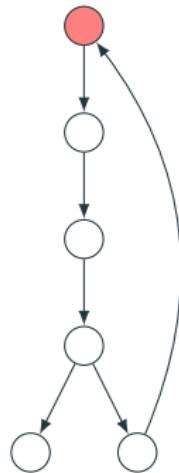
- directed graph
- (which is weakly connected)
- with a single source,
- labelled vertices,
- and ordered outgoing edges.

# How to Interpret an FCG



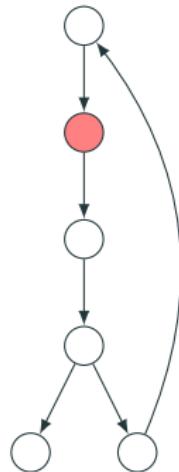
$$f(m, n) =$$

# How to Interpret an FCG



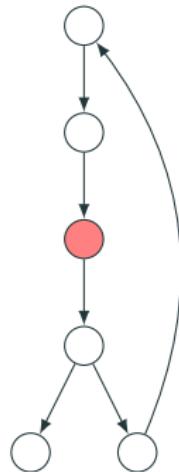
$$f(m, n) =$$

## How to Interpret an FCG



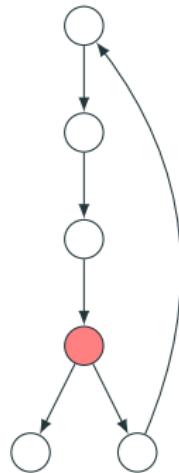
$$f(m, n) = \sum_{l=0}^m \binom{m}{l}$$

# How to Interpret an FCG



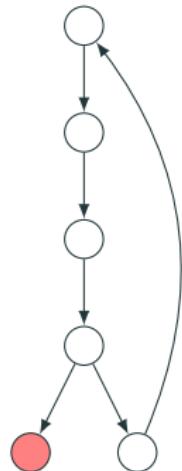
$$f(m, n) = \sum_{l=0}^m \binom{m}{l}$$

# How to Interpret an FCG



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

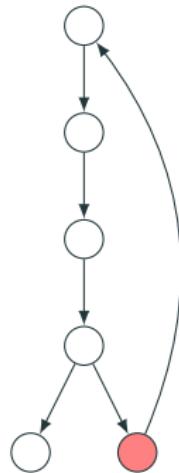
# How to Interpret an FCG



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

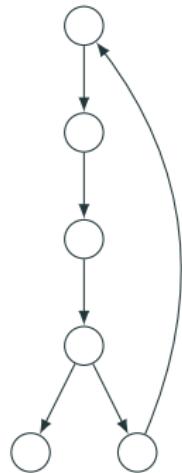
$$[\phi] = \begin{cases} 1 & \text{if } \phi \\ 0 & \text{if } \neg\phi \end{cases}$$

## 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



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

# Compilation Rules

## Definition

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

- $G$  is an FCG,
- and  $L$  is a list of formulas.

## Example 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 Rule: Independence

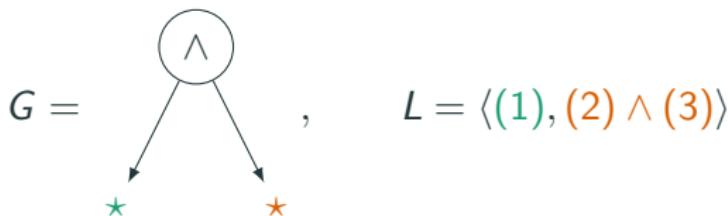
Input formula:

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

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

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

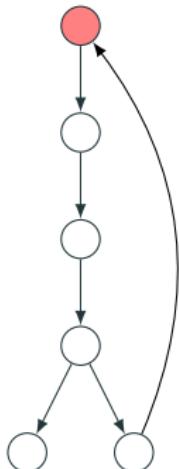
Only one (G, L) pair:



# New Rule 1: 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  $c \in \Gamma$ ):

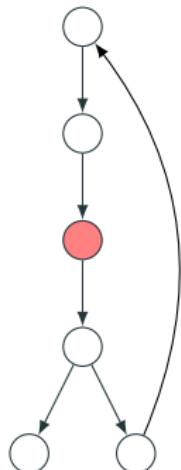
$$\forall y, z \in \Delta. y \neq z \Rightarrow \neg P(c, y) \vee \neg P(c, z)$$

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

## New Rule 2: Constraint Removal

### Example

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



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

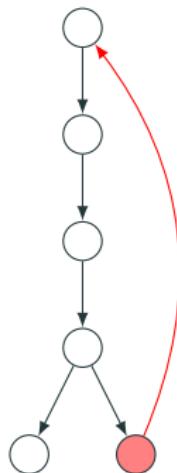
$$\forall w, x \in \Gamma. \forall y \in \Delta. w \neq c \wedge x \neq c \wedge w \neq x \Rightarrow \neg P(w, y) \vee \neg P(x, y)$$

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: Identifying Possibilities for Recursion



### Goal

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

### Rough 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 equal, and the domain map is compatible with previous domain maps, move on to another pair of clauses.

## Resulting Improvements to Counting Functions

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

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

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



# Summary & Future Work

## Summary

The circuits hitherto used for FOMC become more powerful with:

- cycles,
- generalised domain recursion,
- and some more new compilation rules that support domain recursion.

## Future Work

- Automate:
  - extracting and simplifying the definitions of functions,
  - finding all base cases.
- Open questions:
  - What kind of **sequences** are computable in this way?
  - Would using a **different logic** extend the capabilities of FOMC further?