Synthesising Recursive Functions for First-Order Model Counting:

Challenges, Progress, and Conjectures

Paulius Dilkas[1] Vaishak Belle[2]

[1]National University of Singapore, Singapore

[2]University of Edinburgh, UK

KR 2023

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: $n^{\underline{m}} = n \cdot (n-1) \cdots (n-m+1)$.
Note: this problem is equivalent to counting $[m] \to [n]$ 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 $\mathtt{P} \subseteq \Gamma \times \Delta$ 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 $\mathtt{P} \subseteq \Gamma \times \Delta$ 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)

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

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
  - such that $|\Gamma| = m$, and $|\Delta| = n$
- Let $\mathtt{P} \subseteq \Gamma \times \Delta$ 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)

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

  - one person cannot occupy multiple seats

$$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \land \mathtt{P}(x, z) \Rightarrow y = z \tag{2}$$

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
  - such that $|\Gamma| = m$, and $|\Delta| = n$
- Let $\mathtt{P} \subseteq \Gamma \times \Delta$ 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)

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

  - one person cannot occupy multiple seats

$$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \wedge \mathtt{P}(x, z) \Rightarrow y = z \tag{2}$$

  - one seat cannot accommodate multiple attendees

$$\forall w, x \in \Gamma.\ \forall y \in \Delta.\ \mathtt{P}(w, y) \wedge \mathtt{P}(x, y) \Rightarrow w = x \tag{3}$$

Let's Express This Problem in Logic!

- Let Γ and Δ be sets (i.e., domains)
  - such that $|\Gamma| = m$, and $|\Delta| = n$
- Let $\mathtt{P} \subseteq \Gamma \times \Delta$ 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)

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

  - one person cannot occupy multiple seats

$$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \wedge \mathtt{P}(x, z) \Rightarrow y = z \tag{2}$$

  - one seat cannot accommodate multiple attendees

$$\forall w, x \in \Gamma.\ \forall y \in \Delta.\ \mathtt{P}(w, y) \wedge \mathtt{P}(x, y) \Rightarrow w = x \tag{3}$$

(1) and (2) constrain $\mathtt{P}$ to be a function, and (3) 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.

Claim

The capabilities of FOMC algorithms can be expanded by enabling them to construct recursive solutions.

Back to Our Example

The following function counts injections:

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

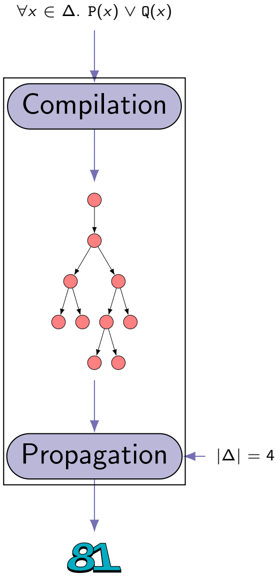
Back to Our Example

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.
- Optimal time complexity to compute $n^{\underline{m}}$ 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 such functions can be found automatically.

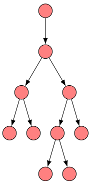
First-Order Knowledge Compilation: Before and After



First-Order Knowledge Compilation: Before and After

$\forall x \in \Delta.\ \mathtt{P}(x) \lor \mathtt{Q}(x)$

Compilation



Propagation

$|\Delta| = 4$

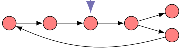
81

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

$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \land \mathtt{P}(x, z) \Rightarrow y = z$

$\forall w, x \in \Gamma.\ \forall y \in \Delta.\ \mathtt{P}(w, y) \land \mathtt{P}(x, y) \Rightarrow w = x$

Compilation



Conversion

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

Simplification

$$f(m, n) = f(m, n - 1) + m \times f(m - 1, n - 1)$$

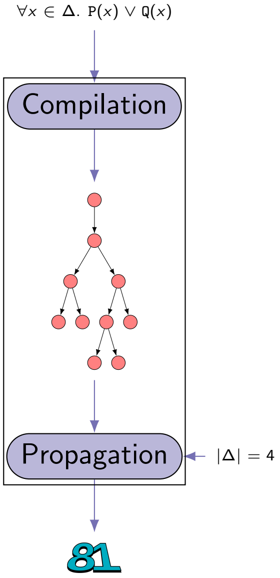
Evaluation

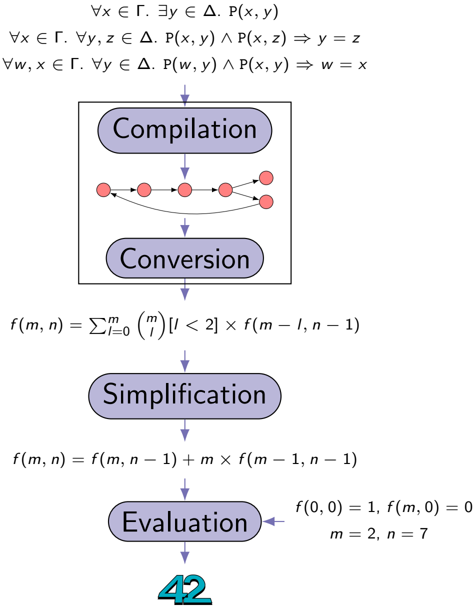
$f(0, 0) = 1,\ f(m, 0) = 0$

$m = 2,\ n = 7$

42

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)...

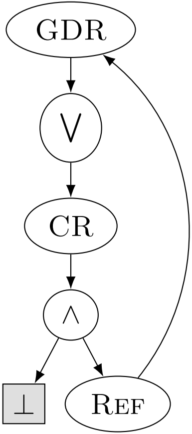
- ...extend d-DNNF circuits (Darwiche 2001) for propositional knowledge compilation with more node types
- ...are acyclic.

First-Order Computational Graphs (FCGs) are...

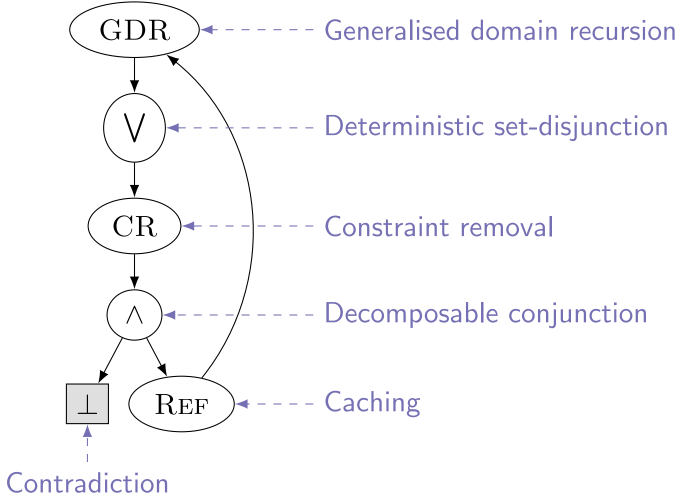
directed ~~acyclic~~ (weakly connected) graphs with:

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

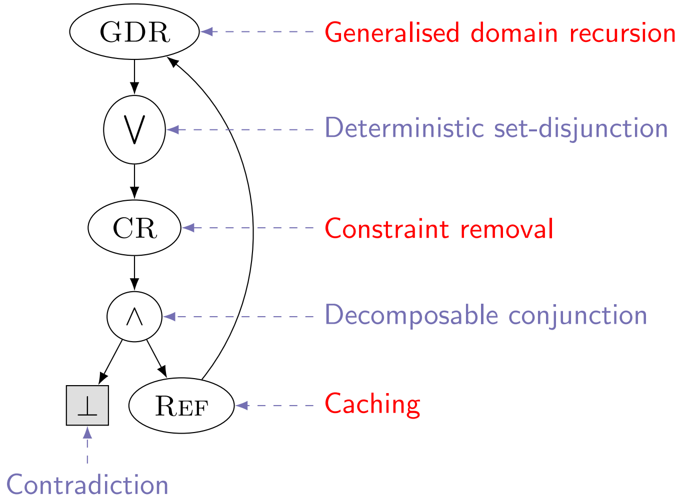
How to Interpret an FCG



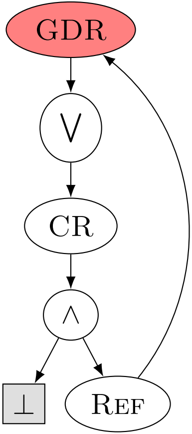
How to Interpret an FCG



How to Interpret an FCG

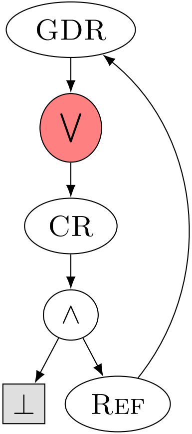


How to Interpret an FCG



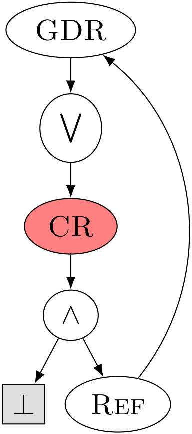
$f(m, n) =$

How to Interpret an FCG



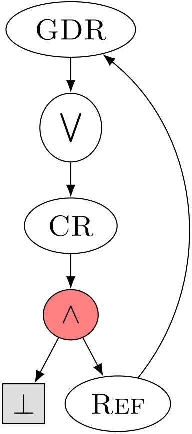
$$f(m,n) = \sum_{l=0}^{m} \binom{m}{l}$$

How to Interpret an FCG



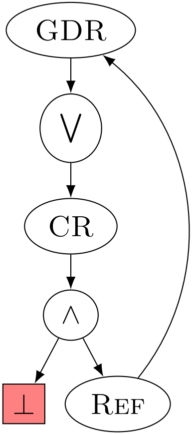
$$f(m, n) = \sum_{l=0}^{m} \binom{m}{l}$$

How to Interpret an FCG



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

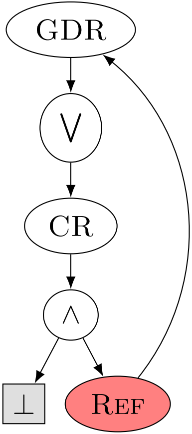
How to Interpret an FCG



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

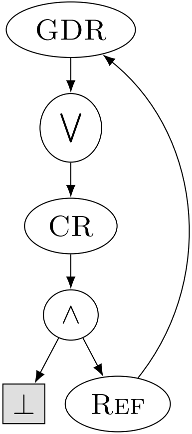
$$[\phi] = \begin{cases} 1 & \text{if } \phi \\ 0 & \text{if } \neg\phi \end{cases}$$

How to Interpret an FCG



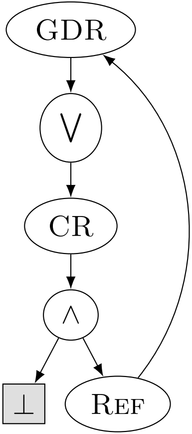
$$f(m,n) = \sum_{l=0}^{m} \binom{m}{l} [l < 2] \times f(m-l, n-1)$$

How to Interpret an FCG



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

How to Interpret an FCG



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

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:

$$(\forall x, y \in \Omega.\ x = y) \land \tag{1}$$

$$(\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \land \mathtt{P}(x, z) \Rightarrow y = z) \land \tag{2}$$

$$(\forall w, x \in \Gamma.\ \forall y \in \Delta.\ \mathtt{P}(w, y) \land \mathtt{P}(x, y) \Rightarrow w = x) \tag{3}$$

Example Compilation Rule: Independence

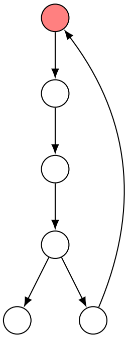
Input formula:

$$(\forall x, y \in \Omega.\ x = y) \wedge \tag{1}$$
$$(\forall x \in \Gamma.\ \forall y, z \in \Delta.\ \mathtt{P}(x, y) \wedge \mathtt{P}(x, z) \Rightarrow y = z) \wedge \tag{2}$$
$$(\forall w, x \in \Gamma.\ \forall y \in \Delta.\ \mathtt{P}(w, y) \wedge \mathtt{P}(x, y) \Rightarrow w = x) \tag{3}$$

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

$$G = \begin{tikzcd} & \wedge \arrow[dl] \arrow[dr] & \\ \star & & \star \end{tikzcd}, \qquad L = \langle (1), (2) \wedge (3) \rangle$$

New Rule 1/3: Generalised Domain Recursion



Example

Input formula:

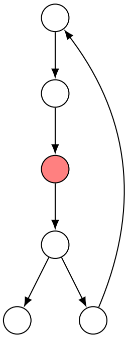
$$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ y \neq z \Rightarrow \neg \mathrm{P}(x, y) \vee \neg \mathrm{P}(x, z)$$

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

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

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

New Rule 2/3: Constraint Removal



Example

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

$$\forall x \in \Gamma.\ \forall y, z \in \Delta.\ x \neq c \land y \neq z \Rightarrow \neg \mathrm{P}(x, y) \lor \neg \mathrm{P}(x, z)$$

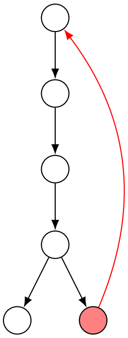
$$\forall w, x \in \Gamma.\ \forall y \in \Delta.\ w \neq c \land x \neq c \land w \neq x \Rightarrow \neg \mathrm{P}(w, y) \lor \neg \mathrm{P}(x, y)$$

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

$$\forall x \in \Gamma'.\ \forall y, z \in \Delta.\ y \neq z \Rightarrow \neg \mathrm{P}(x, y) \lor \neg \mathrm{P}(x, z)$$

$$\forall w, x \in \Gamma'.\ \forall y \in \Delta.\ w \neq x \Rightarrow \neg \mathrm{P}(w, y) \lor \neg \mathrm{P}(x, y)$$

New Rule 3/3: Identifying Possibilities for Recursion



Goal

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

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 Γ and Δ be two sets with cardinalities $|\Gamma| = m$ and $|\Delta| = n$. Our new compilation rules enables us to count $\Gamma \to \Delta$ 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!

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:
  - 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?