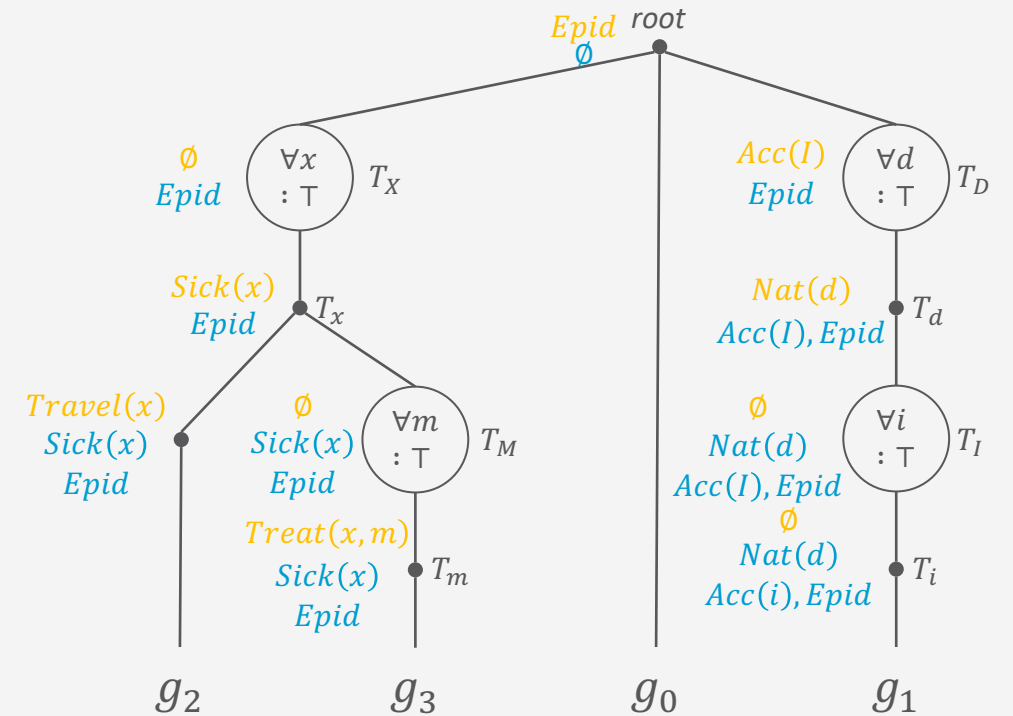


Exact Inference: Lifted Inference

Statistical Relational Artificial Intelligence
(StaRAI)



Contents

1. Introduction

- Notions of AI, agents, StaRAI

2. Modelling Formalisms

- a. *Logic*
- b. *Probabilistic Graphical Models (PGMs)*
- c. *Probabilistic Relational Models (PRMs)*
- Syntax, semantics, inference tasks

3. Inference

- a. *Exact Inference*
 - i. *(Weighted) Model Counting*
 - ii. *Variable Elimination*
 - iii. *Lifted Inference*

b. *Approximate Inference*

- *Inference algorithms*

4. Learning

- Relation learning in logic
- Parameter / structure learning for PGMs
- From PGMs to PRMs, PRM learning

5. Sequential Models and Inference

- Syntax, semantics, inference tasks, algorithms

6. Decision-theoretic Models and Inference

- Preferences, utility
- Syntax, semantics, inference task, algorithm

Outline: 3. Inference

a. Exact Inference

i. (Weighted) Model Counting

- Model counting
- Weighted model counting, knowledge compilation (KC)

ii. Variable Elimination and the Like

- Variable elimination (VE)
- Junction tree algorithm (JT)

iii. Lifted Inference

- First-order KC, Lifted VE, lifted JT

b. Approximate Inference

- Direct sampling: Rejection sampling, (lifted) importance sampling
- (Lifted) Markov Chain Monte Carlo sampling

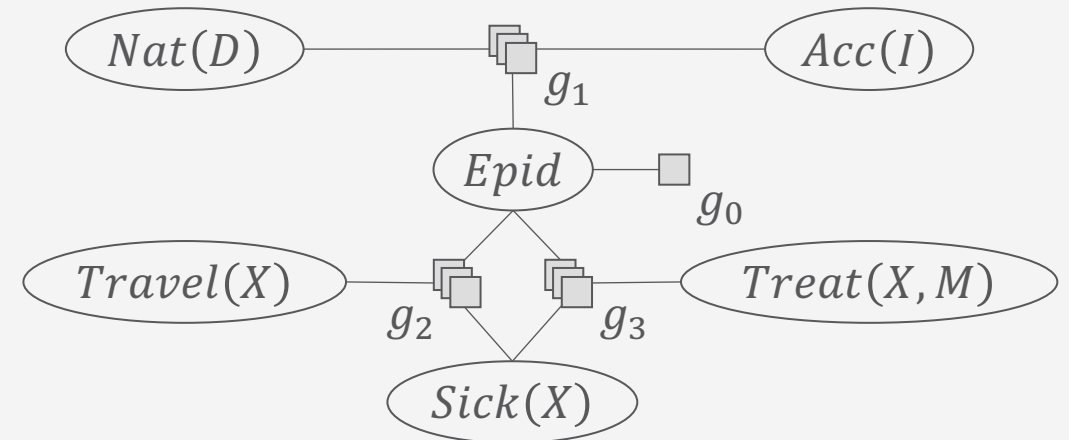
Inference Tasks

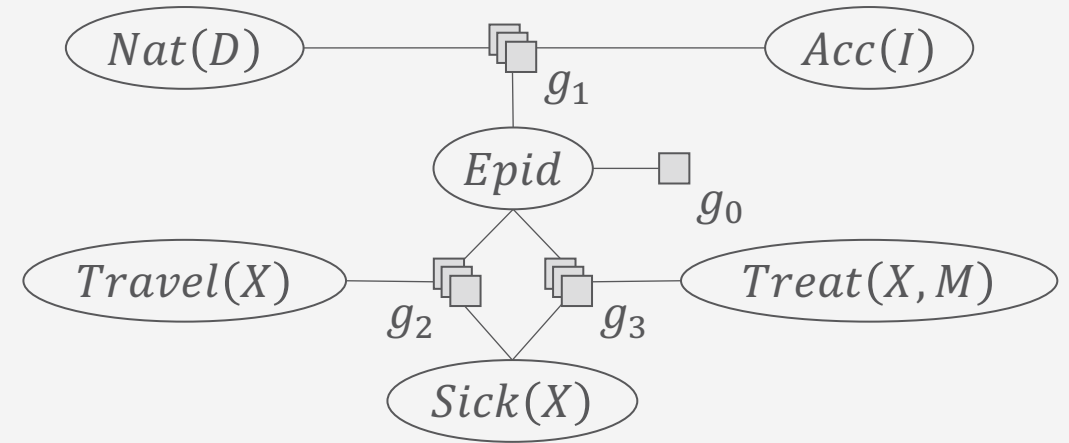
- Query Answering Problem (as before)
 - Compute an answer to a query $P(\mathbf{S}|\mathbf{T})$ given a model G representing the full joint probability distribution P_G
 - Avoid grounding (parts of) G
 - E.g.,
 - $P(\text{Treat}(\text{eve}, m_1))$
 - $P(\text{Travel}(\text{eve}), \text{Epid})$
 - $P(\text{Sick}(\text{eve})|\text{Epid})$
 - $P(\text{Epid}|\text{Sick}(\text{eve}) = \text{true})$
 - $P(\#_E[\text{Epid}(E)])$
 - $P(\#_E[\text{Epid}(E)] = [2,2])$

- Model: either parfactor model or MLN

10 $\text{Presents}(X, P, C) \Rightarrow \text{Attends}(X, C)$

3.75 $\text{Publishes}(X, C) \wedge \text{FarAway}(C) \Rightarrow \text{Attends}(X, C)$





Lifted Variable Elimination

Inference algorithm

Complexity analysis, tractability

What Happens During Variable Elimination Given Relations?

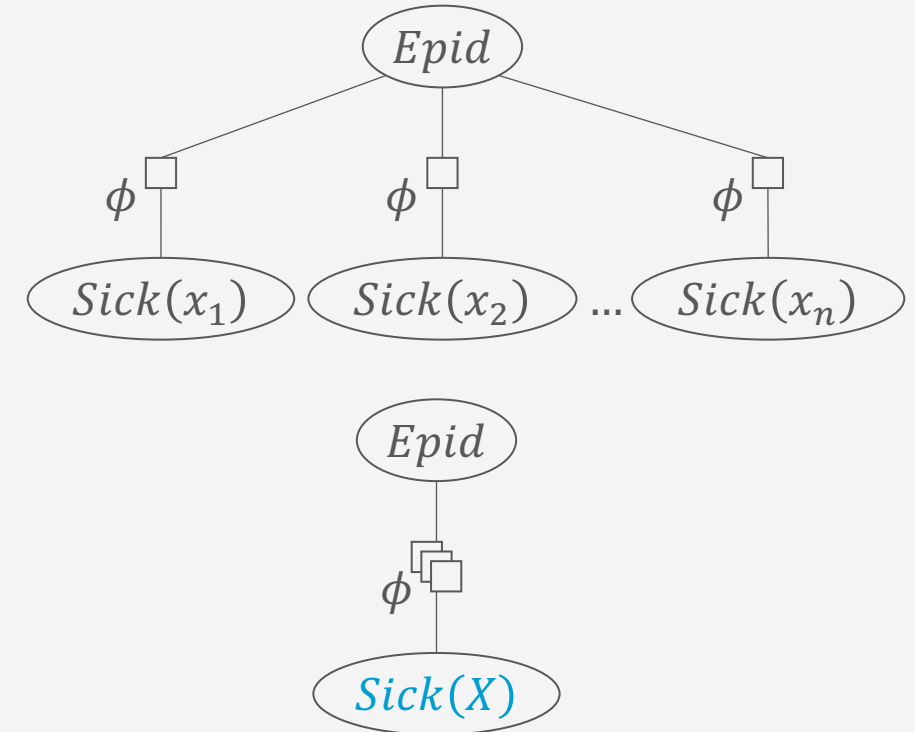
$$P(Epid) \propto \sum_{s_1 \in \text{ran}(Sick(x_1))} \phi(Epid, s_1) \cdot \sum_{s_2 \in \text{ran}(Sick(x_2))} \phi(Epid, s_2) \cdot \dots \cdot \sum_{s_n \in \text{ran}(Sick(x_n))} \phi(Epid, s_n)$$

$$= \underbrace{\phi'(Epid) \cdot \phi'(Epid) \cdot \dots \cdot \phi'(Epid)}_{n \text{ times}} = (\phi'(Epid))^n$$

$$P(Epid) \propto \left(\sum_{s \in \text{ran}(Sick(x))} \phi(Epid, Sick(x) = s) \right)^n = (\phi'(Epid))^n$$

Representative for X in $\phi(Epid, Sick(X))$

of X relative to $Epid$



Lifted Variable Elimination (LVE)

- Outline:

1. Absorb evidence $\mathbf{t}$ in each **parfactor** g covered by $\mathbf{t}$, i.e., $rv(g) \cap \mathbf{t} \neq \emptyset$, **in a lifted way**,
2. Eliminate non-query **PRVs** $\mathbf{U} = \mathbf{R} \setminus rv(\mathbf{S}, \mathbf{t})$ **in a lifted way** in model G

$$\begin{aligned}
 P(\mathbf{S} \mid \mathbf{t}) &= \frac{1}{P(\mathbf{t})} \sum_{\mathbf{u} \in \text{ran}(\mathbf{U})} P_G(\mathbf{S}, \mathbf{t}, \mathbf{U} = \mathbf{u}) \\
 &= \frac{1}{P(\mathbf{t})} \sum_{\mathbf{u} \in \text{ran}(\mathbf{U})} \prod_{g \in G} \phi_g(\underbrace{R_1, \dots, R_k}_{\pi_{rv(f)}(\mathbf{S}, \mathbf{t}, \mathbf{U} = \mathbf{u})})
 \end{aligned}$$

- Factor out **parfactors** from sums if arguments not covered by sum
 - May require manipulation of constraints as at least constants appearing in query are *distinguishable*
3. Divide by $P(\mathbf{t})$ = Normalise $P(\mathbf{S}, \mathbf{t})$

Lifted operators for

- Summing out
- Multiplication
- Absorption of $\mathbf{t}$

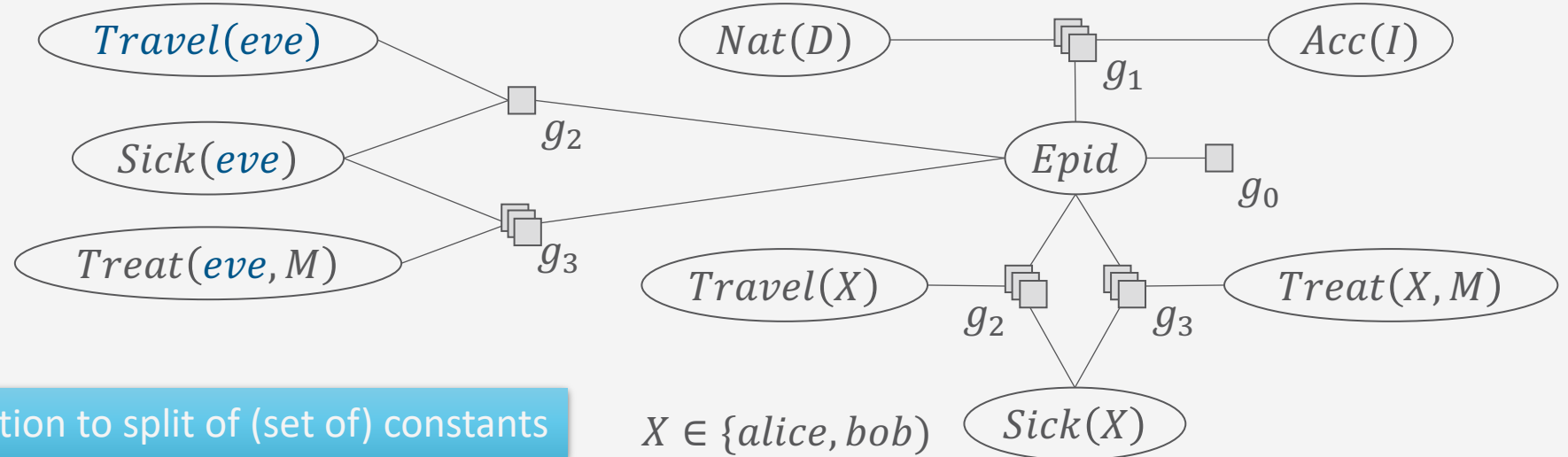
→ Lifting operators of LVE

Lifting operators to enable the main operators above necessary

LVE in Detail

- Example: $P(\textit{Travel}(\textit{eve}))$ in $G = \{g_i\}_{i=0}^3$
 - Pre-processing:
Split all parfactors whose constraint contains constants occurring in query terms: *eve*
 - If parameterised query $P(A|_C)$: split parfactors based on C

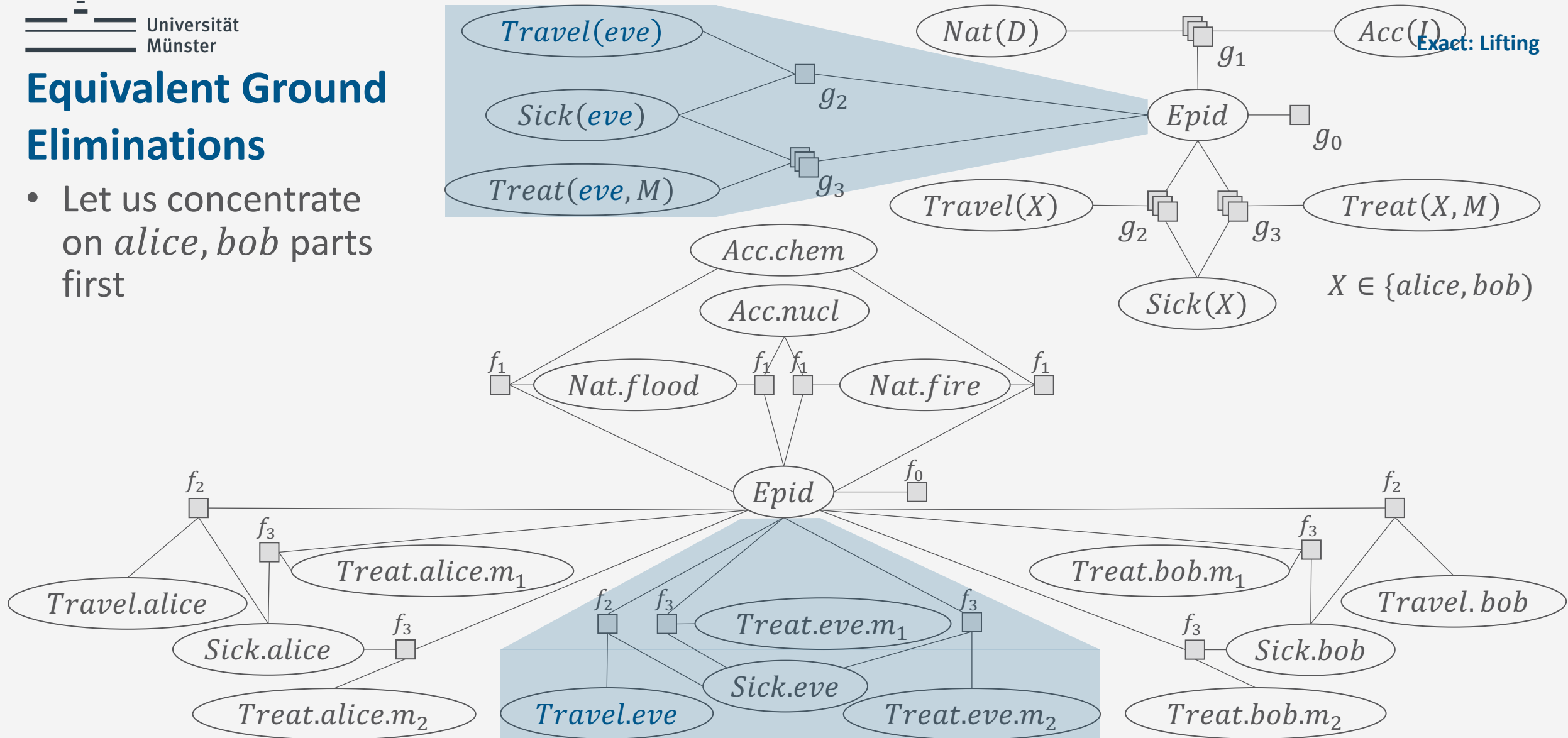
Called shattering



So, we need a formal *split* operation to split of (set of) constants

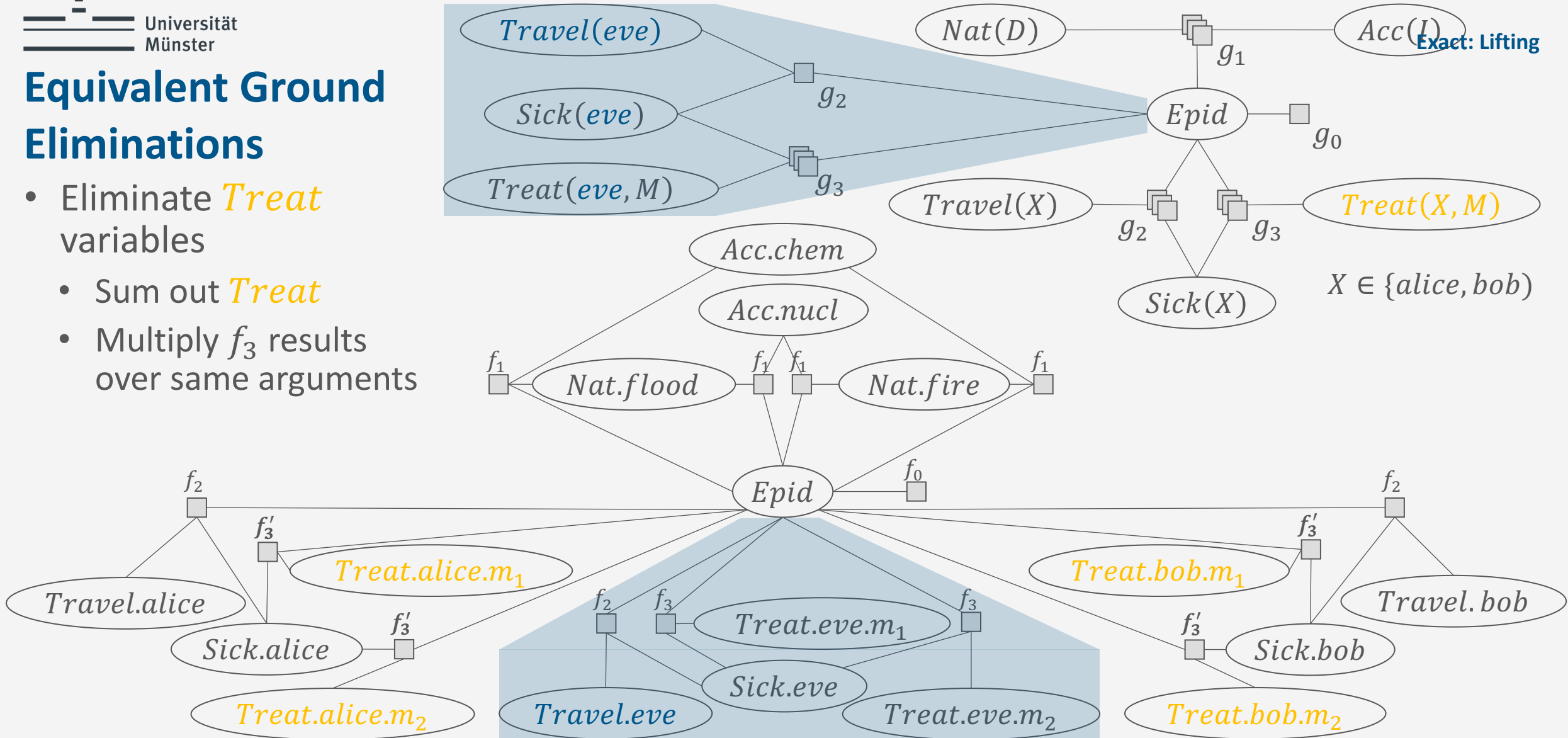
Equivalent Ground Eliminations

- Let us concentrate on *alice, bob* parts first



Equivalent Ground Eliminations

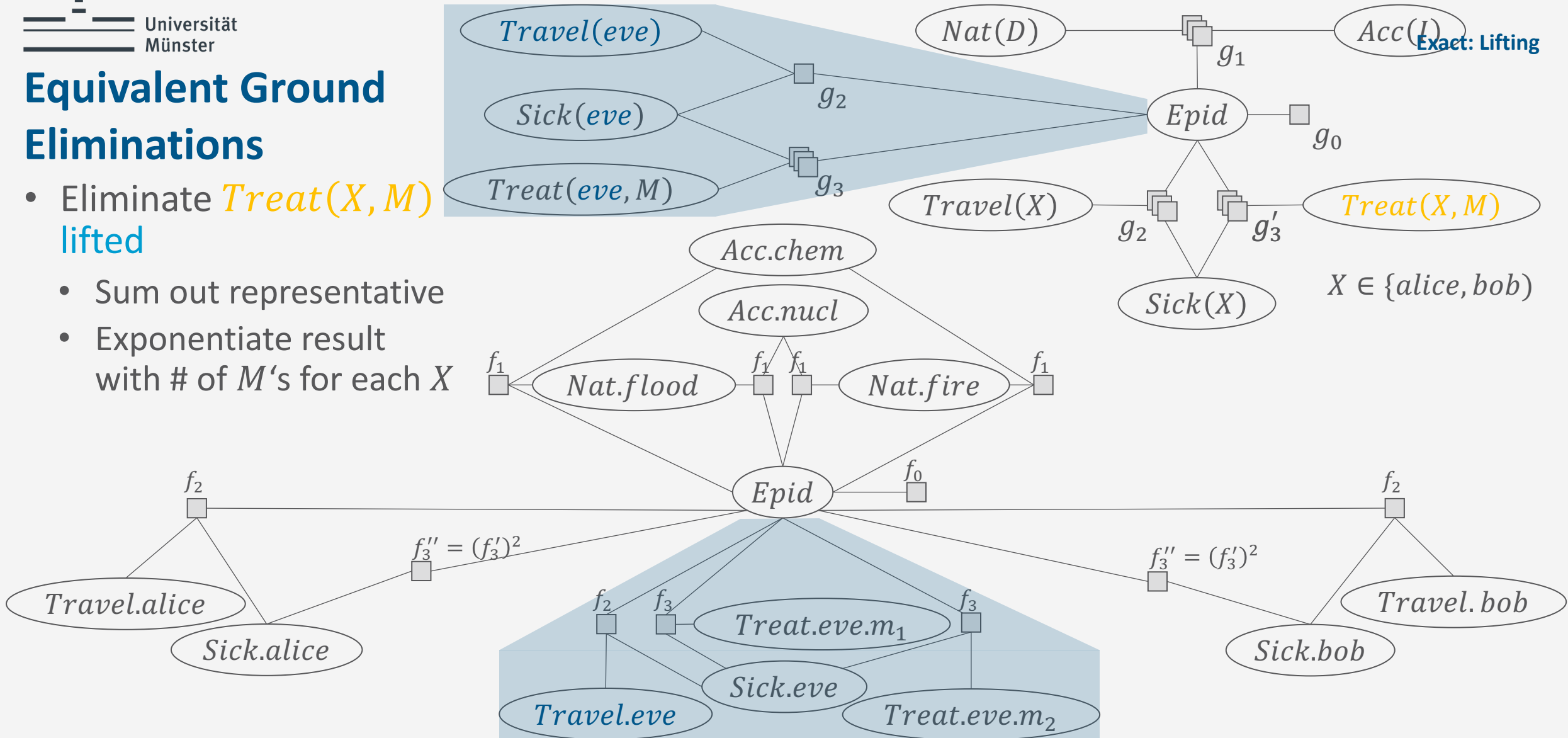
- Eliminate *Treat* variables
- Sum out *Treat*
- Multiply f_3 results over same arguments



Equivalent Ground Eliminations

- Eliminate *Treat*(*X*, *M*) lifted

- Sum out representative
- Exponentiate result with # of *M*'s for each *X*

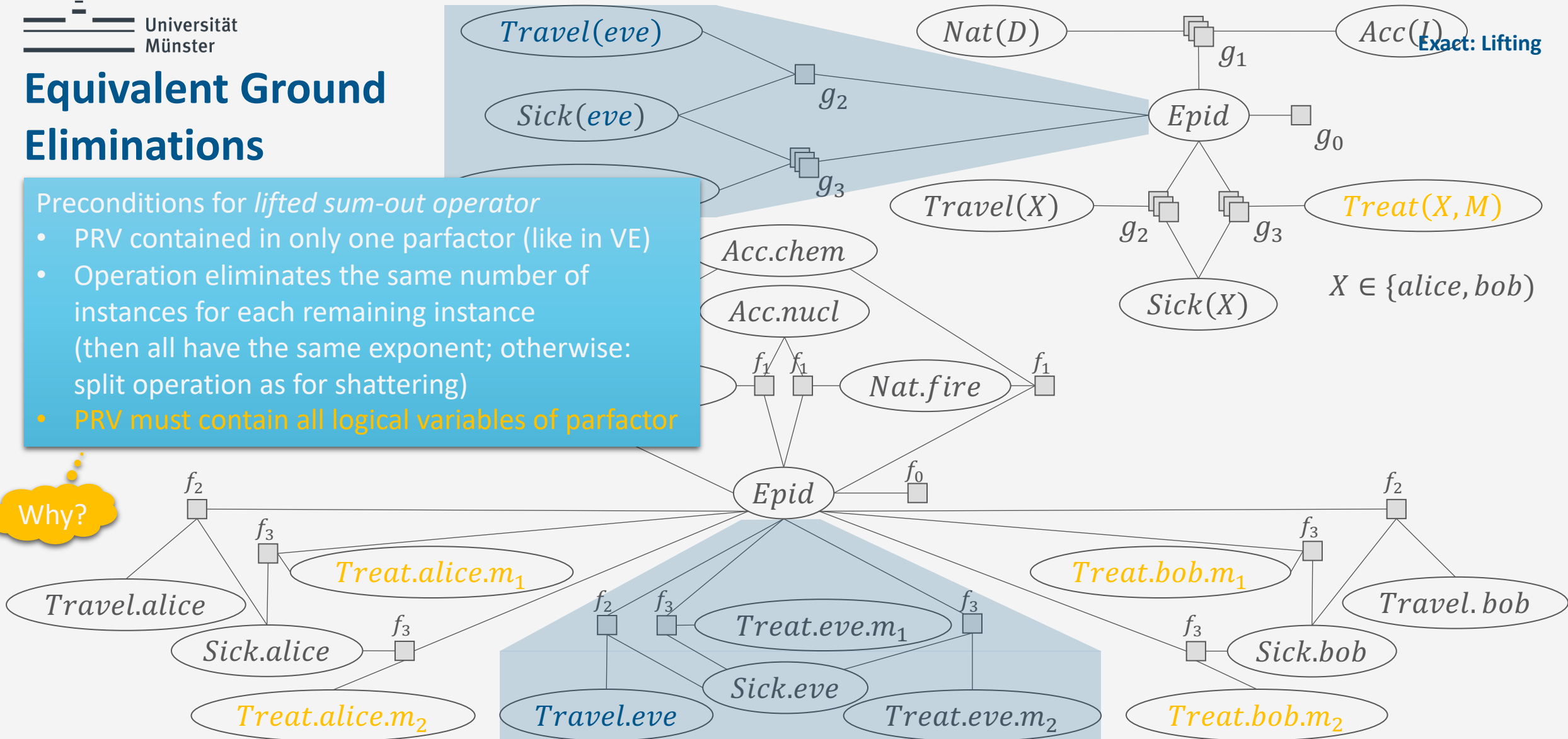


Equivalent Ground Eliminations

Preconditions for *lifted sum-out* operator

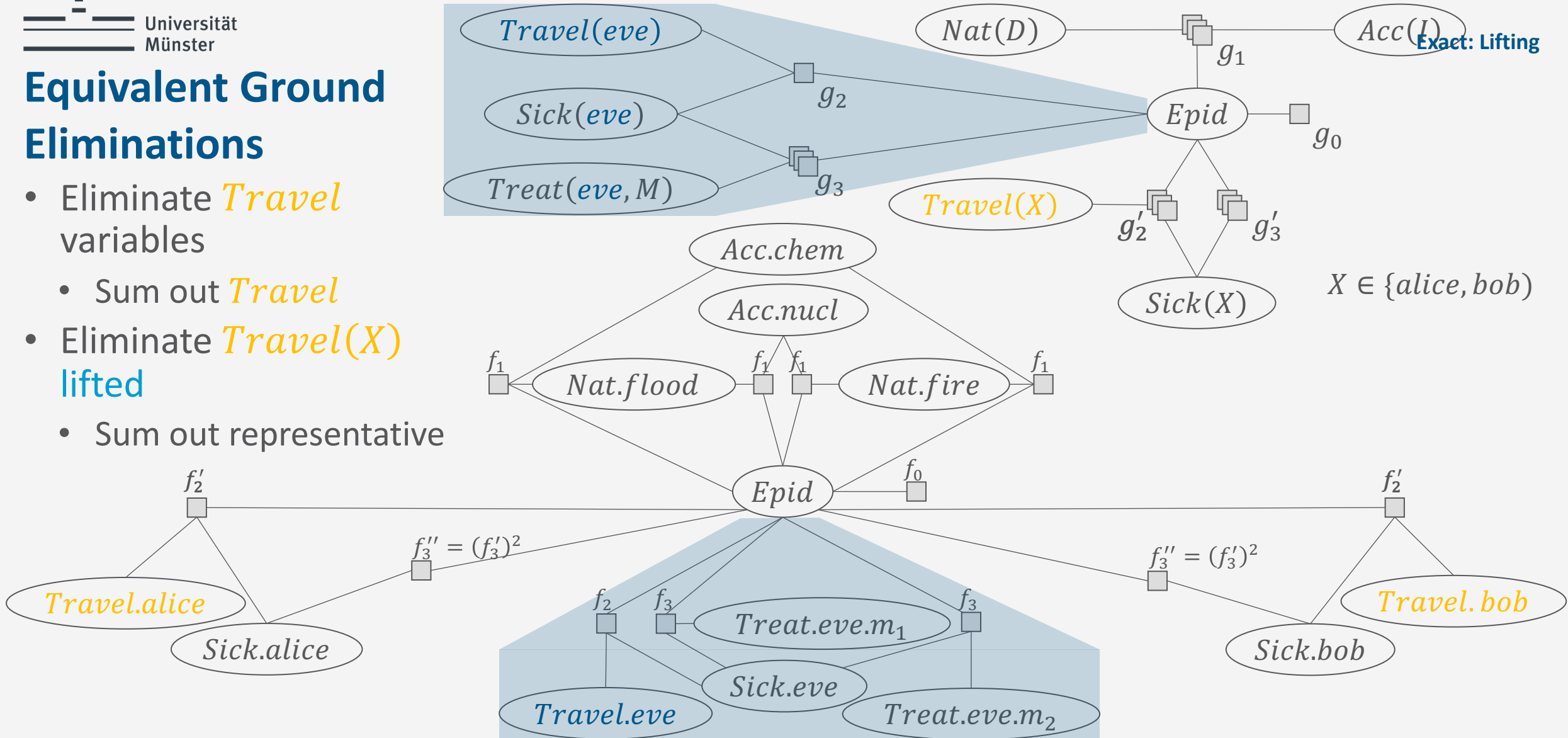
- PRV contained in only one parfactor (like in VE)
- Operation eliminates the same number of instances for each remaining instance (then all have the same exponent; otherwise: split operation as for shattering)
- PRV must contain all logical variables of parfactor

Why?



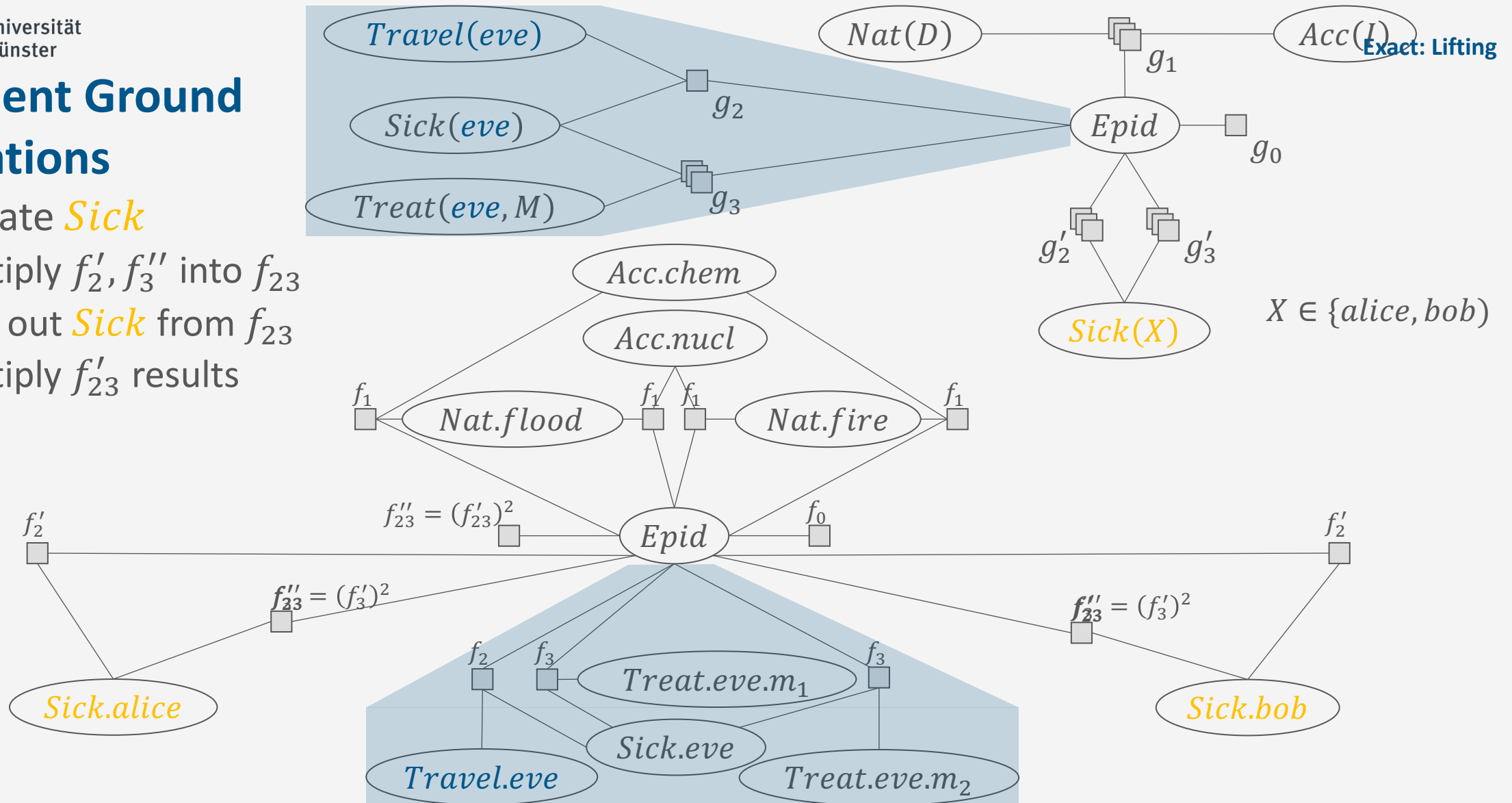
Equivalent Ground Eliminations

- Eliminate *Travel* variables
- Sum out *Travel*
- Eliminate *Travel(X)* lifted
- Sum out representative



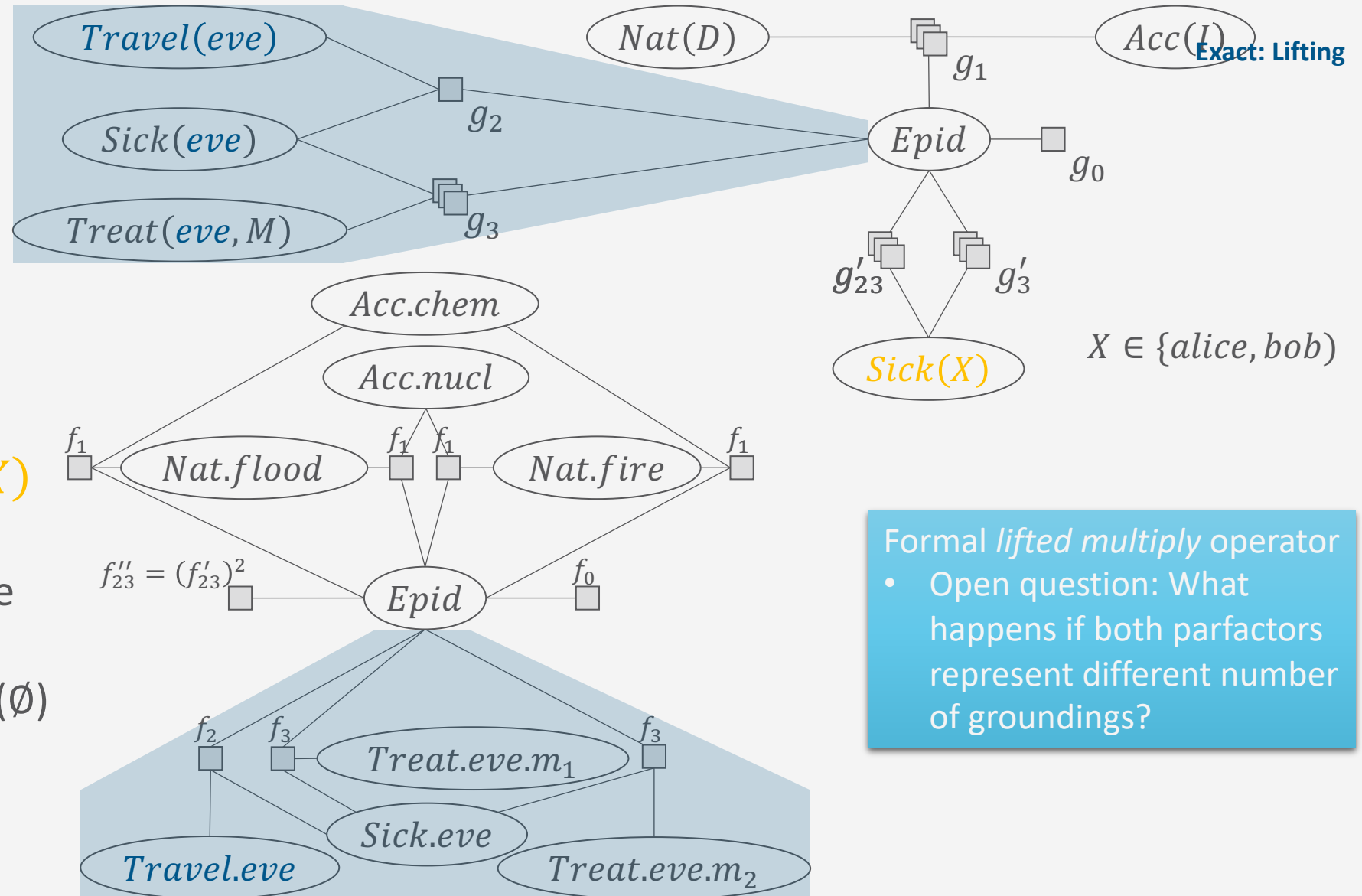
Equivalent Ground Eliminations

- Eliminate *Sick*
 - Multiply f'_2, f'_3 into f_{23}
 - Sum out *Sick* from f_{23}
 - Multiply f'_{23} results



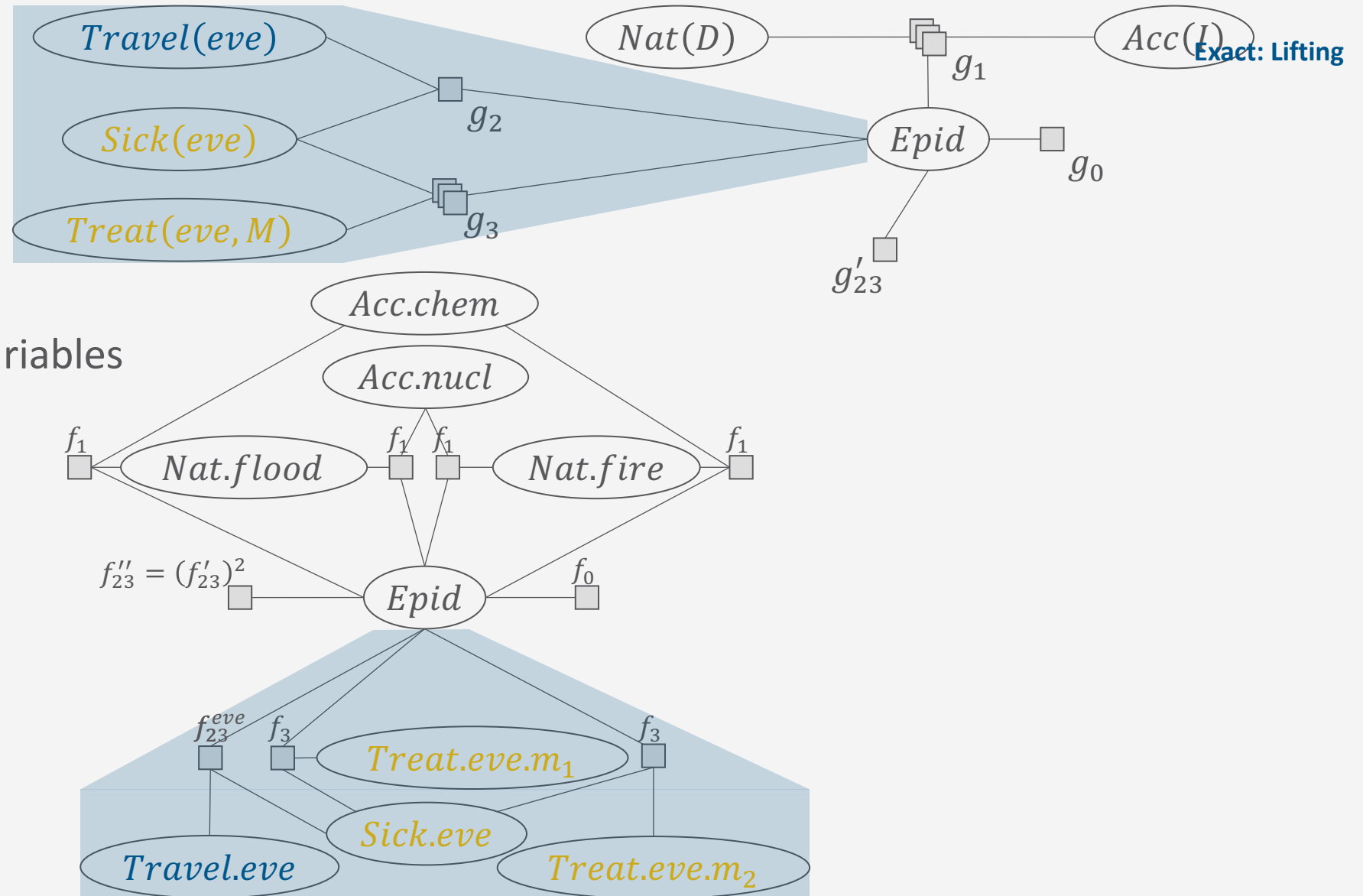
Equivalent Ground Eliminations

- Eliminate *Sick(X)* lifted
- First:
 - Multiply g'_2, g'_3 lifted
- Then, eliminate *Sick(X)* lifted
 - Sum out representative
 - Exponentiate result with # of X 's for *Epid* ($\emptyset$)



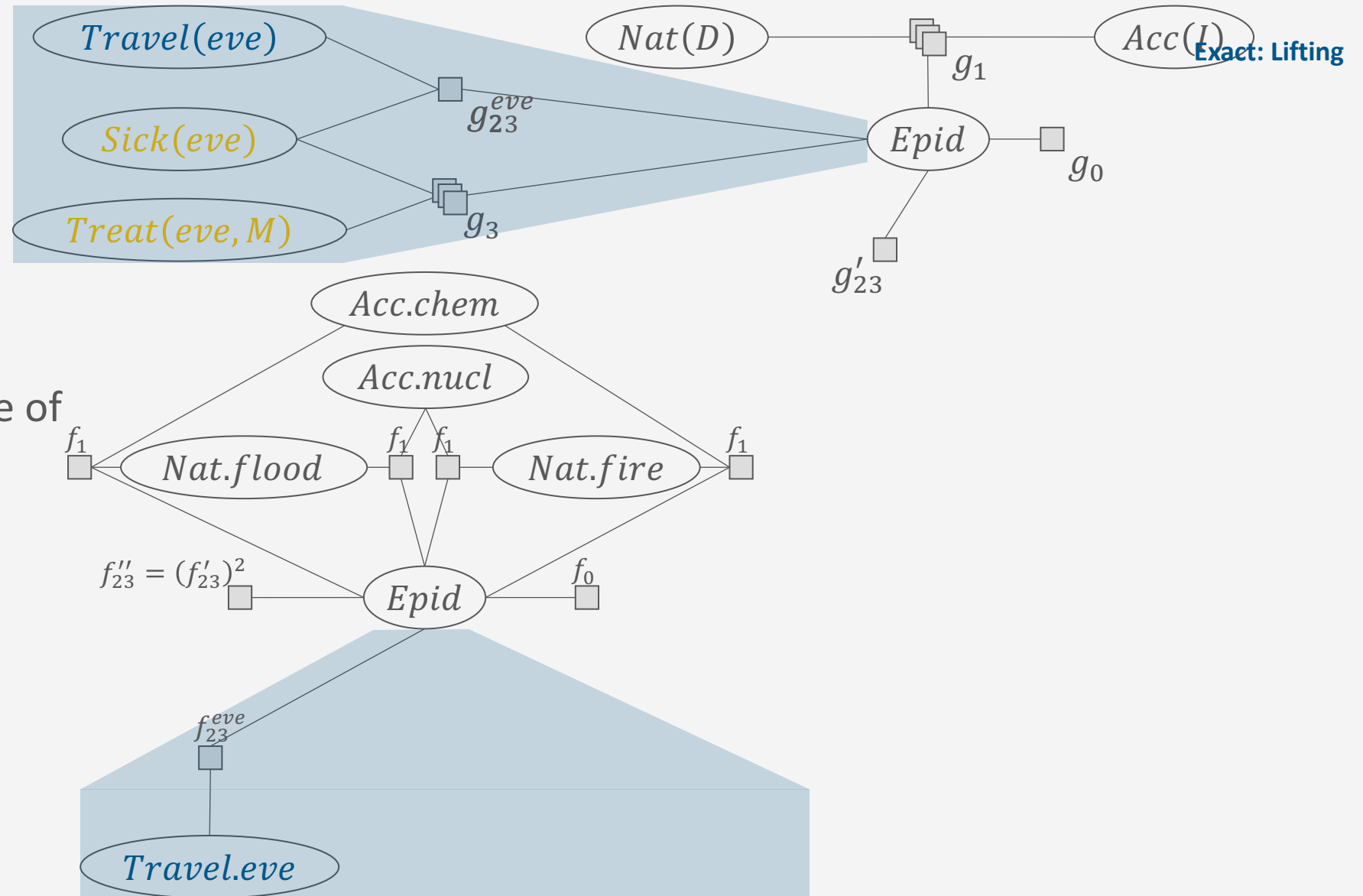
Equivalent Ground Eliminations

- Eliminate *Treat.eve*, *Sick.eve* variables
- Sum out *Treat.eve* variables
- Multiply results over same arguments
- Multiply f_2, f_3'' for *eve*
- Sum out *Sick.eve*



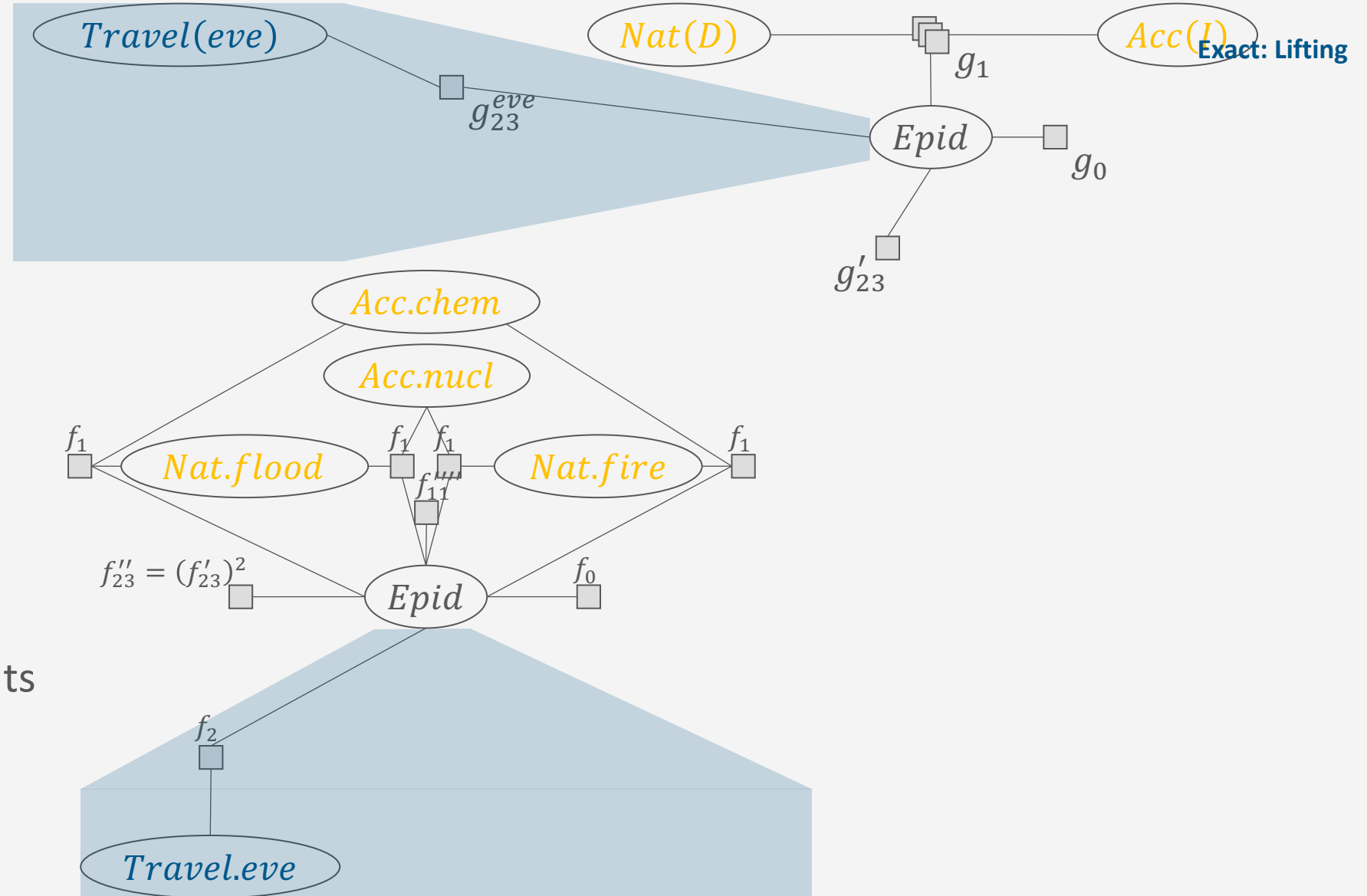
Equivalent Ground Eliminations

- Eliminate $Treat(eve, M)$ lifted, $Sick(eve)$
- Sum out representative of $Treat(eve, M)$
- Exponentiate with 2
- Multiply g_2, g'_3
- Sum out $Sick(eve)$



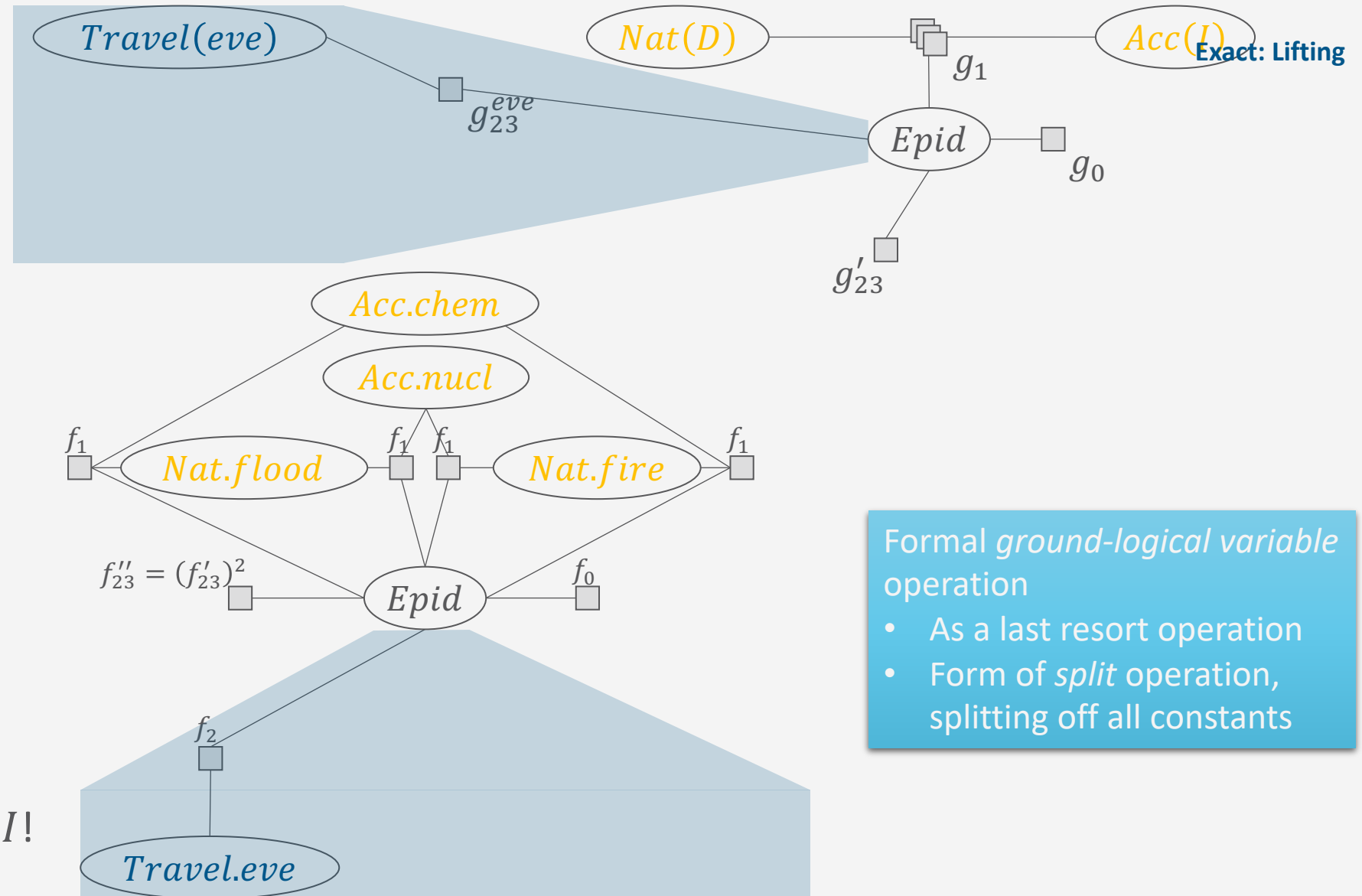
Equivalent Ground Eliminations

- Eliminate *Nat*, *Acc* variables
- Start with *Nat.flood*
 - Multiply f_1, f_1 over *Acc.chem*, *Acc.nucl*
 - Sum out *Nat.flood*
- Same for *Nat.fire*
 - Identical result
- Multiply identical results into f_{11}''
- Sum out *Acc.chem*, *Acc.nucl*



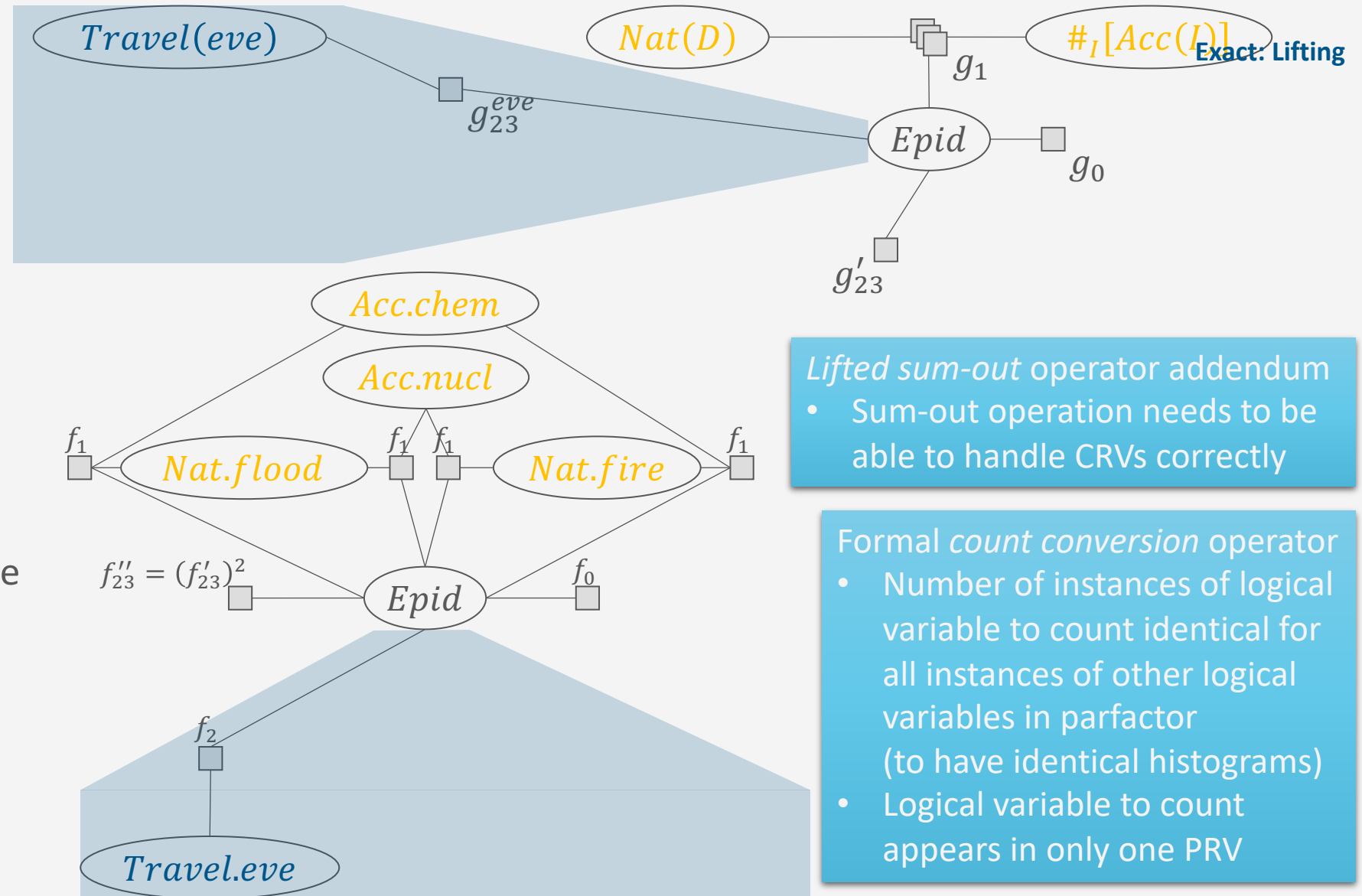
Equivalent Ground Eliminations

- Eliminate $Nat(D)$, $Acc(I)$ lifted
- Problem: Neither contains all logical variables of g_1
- Solution: Ground I ?
 - Eliminate $Nat(D)$
 - Eliminate $Acc(chem)$, $Acc(nucl)$
- But: local symmetries, encode in histograms
- Better solution: Count I !



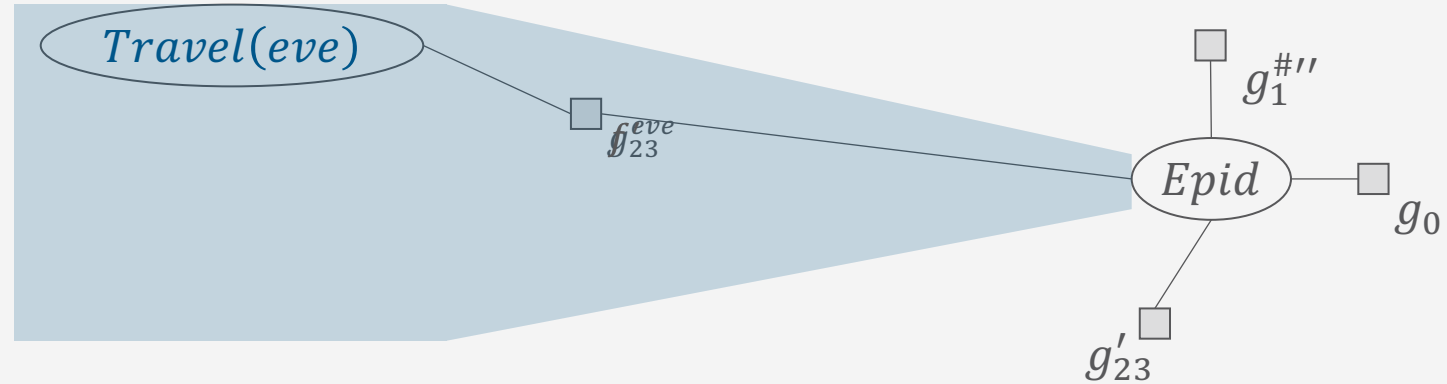
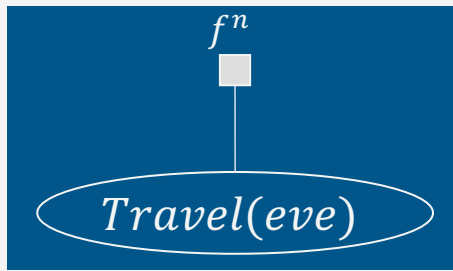
Equivalent Ground Eliminations

- Eliminate $Nat(D)$, $Acc(I)$ lifted
 - Count I in $Acc(I)$
 - CRV $\#_I[Acc(I)]$
- Eliminate $Nat(D)$ lifted
 - Sum out representative
 - Exponentiate with $\#$ of D 's
- Eliminate $\#_I[Acc(I)]$
 - Sum out while considering $Mul(I)$

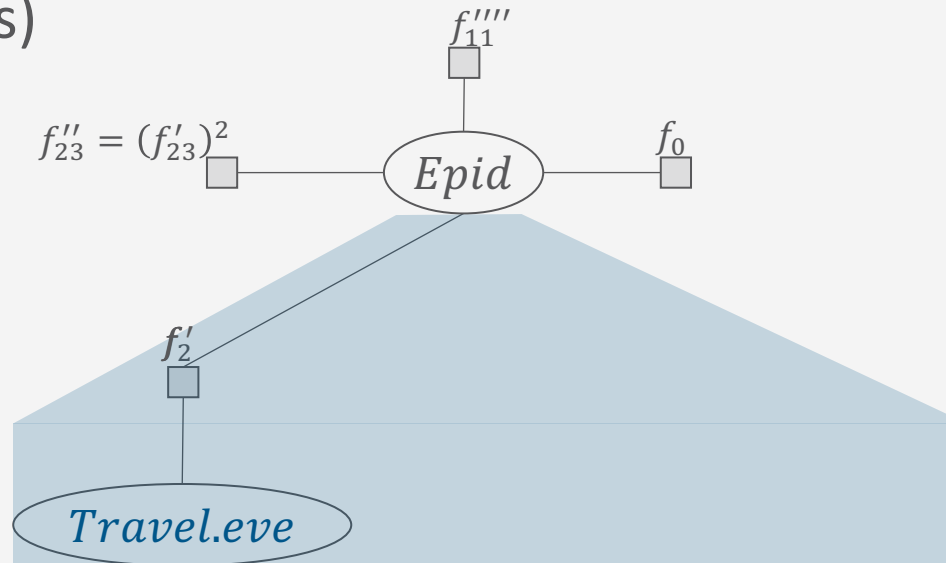


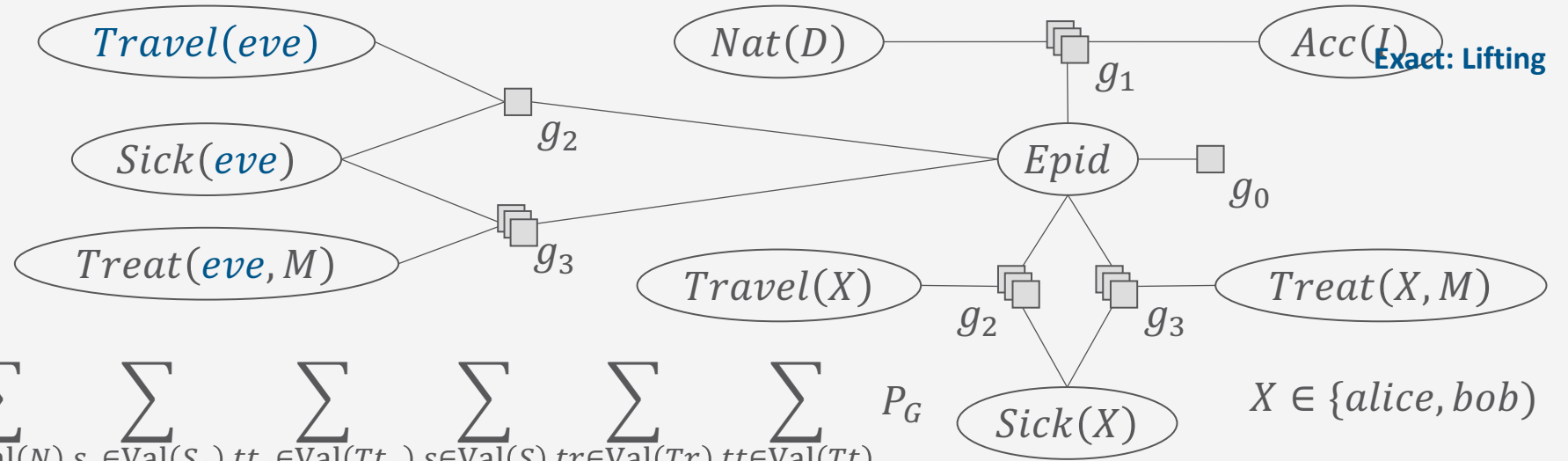
Equivalent Ground Eliminations

- Eliminate *Epid*
 - Identical in both cases
 - Multiply all remaining factors into f
 - Sum out *Epid*
- (Multiply remaining factors)
 - Here only one factor f'
- Normalise result
 - $f^n = P(\text{Travel}(\text{eve}))$



Exact: Lifting





$P(\text{Travel}(\text{eve}))$

$$\propto \sum_{e \in \text{Val}(E)} \sum_{a \in \text{Val}(\#_I[\text{Acc}(I)])} \sum_{n \in \text{Val}(N)} \sum_{s_e \in \text{Val}(S_e)} \sum_{tt_e \in \text{Val}(Tt_e)} \sum_{s \in \text{Val}(S)} \sum_{tr \in \text{Val}(Tr)} \sum_{tt \in \text{Val}(Tt)} P_G$$

$$\propto \sum_{e \in \text{Val}(E)} \phi_0(e) \sum_{a \in \text{Val}(\#_I[\text{Acc}(I)])} \left(\sum_{n \in \text{Val}(N)} \phi_1(e, n, a) \right)^2 \sum_{s_e \in \text{Val}(S_e)} \phi_2(\text{Travel}, e, s_e) \left(\sum_{tt_e \in \text{Val}(Tt_e)} \phi_3(e, s_e, tt_e) \right)^2 \left(\sum_{s \in \text{Val}(S)} \sum_{tr \in \text{Val}(Tr)} \phi_2(tr, e, s) \right)^2 \left(\sum_{tt \in \text{Val}(Tt)} \phi_3(e, s, tt) \right)^2$$

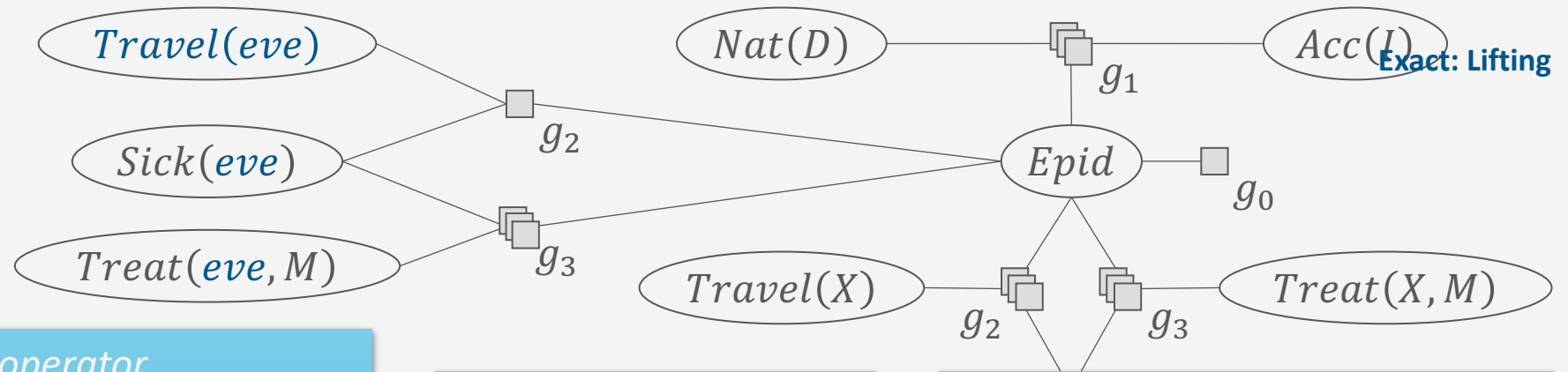
Lifted operators for

- Summing out
- Multiplication
- Absorption of t

→ Lifting operators of LVE

Lifting operators to enable the main operators above necessary

Lifted Operators and Their Preconditions



Preconditions for *lifted sum-out operator*

- PRV contained in only one parfactor (like in VE)
- **PRV must contain all logical variables of parfactor**
- Operation eliminates the same number of instances for each remaining instance (then all have the same exponent; otherwise: split operation as for shattering)

Lifted sum-out operator addendum

- Sum-out operation needs to be able to handle CRVs correctly

So, we need a formal *split* operation to split of (set of) constants

Formal *lifted multiply operator*

- Open question: What happens if both parfactors represent different number of groundings?

Formal *count conversion operator*

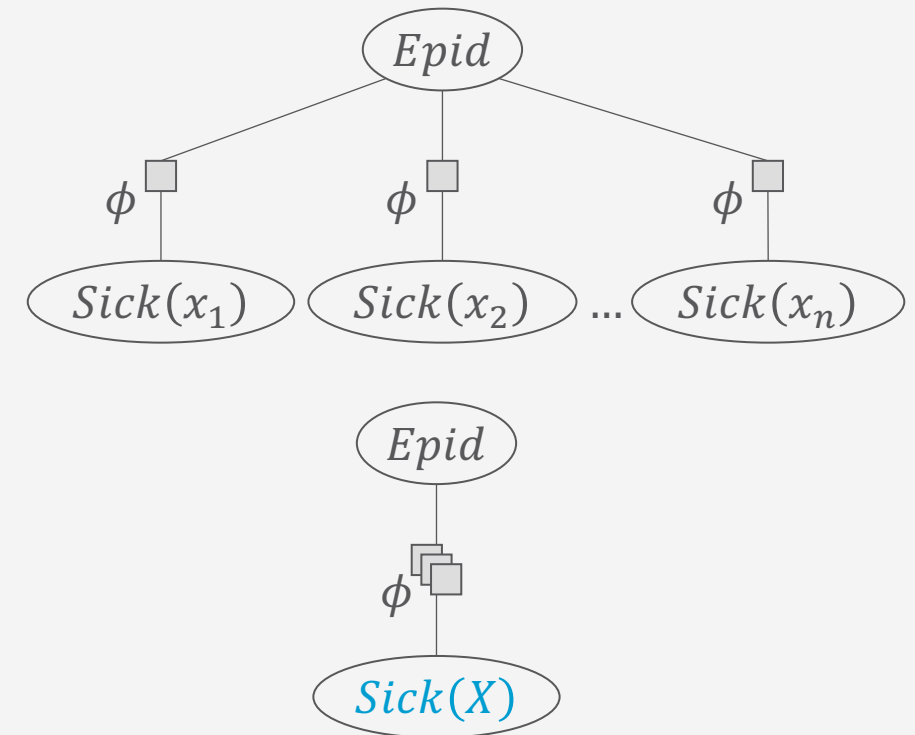
- Number of instances of logical variable to count identical for all instances of other logical variables in parfactor (to have identical histograms)
- Logical variable to count appears in only one PRV

Formal *ground-logical variable operation*

- As a last resort operation
- Form of *split* operation, splitting off all constants

Count Normalisation

- For the different possible groundings of common logical variables X^{com} , the same number of groundings of exclusive logical variables X^{excl} exist
 - X^{excl} contains logical variables that are eliminated during a sum-out operation or the logical variable to count
- Trivial if $X^{com} = \emptyset$:
 - E.g.,
 - $(\mathcal{X}, C_{\mathcal{X}}) = ((X), \{(x_1), \dots, (x_n)\})$
 - $X^{com} = lv(Epid) = \emptyset$
 - $X^{excl} = lv(Sick(X)) \setminus \emptyset = \{X\}$
 - For each possible grounding of *Epid*, which is just one, namely *Epid*, there are n groundings of X
 - One lifted sum-out operation replaces n ground operations



Lifted Summing Out

- Summing out transforms the current model G
 - Removes a PRV from $rv(G)$
- Effect on logical variables :
 - Number of logical variables decreases over the whole LVE run for one query
 - Until only propositional random variables and CRVs (counted logical variables are bound) are left
→ Standard variable elimination
- Preconditions act as a filter on possible sum-out operations

Preconditions for *lifted sum-out operator*

- PRV contained in only one parfactor (like in VE)
- **PRV must contain all logical variables of parfactor**
- Operation eliminates the same number of instances for each remaining instance (then all have the same exponent; otherwise: split operation as for shattering)

Lifted sum-out operator addendum

- Sum-out operation needs to be able to handle CRVs correctly

Lifted Multiplication

- Operator for multiplication as an “enabler” for sum—out operator
 - SO Precondition 1: PRV to sum out may only appear in one parfactor
- Multiply two parfactors
 - Still a join of over arguments and a product of potentials
 - Since two parfactors represent two (different) sets of grounded factors, lifted multiplication has to work as a representative multiplication for those two sets
 - Easy case:
1-to-1 correspondence between groundings of those parfactors
 - *But, what happens if the number of represented factors differ?*
 - 1-to-m correspondence
 - n-to-m correspondence

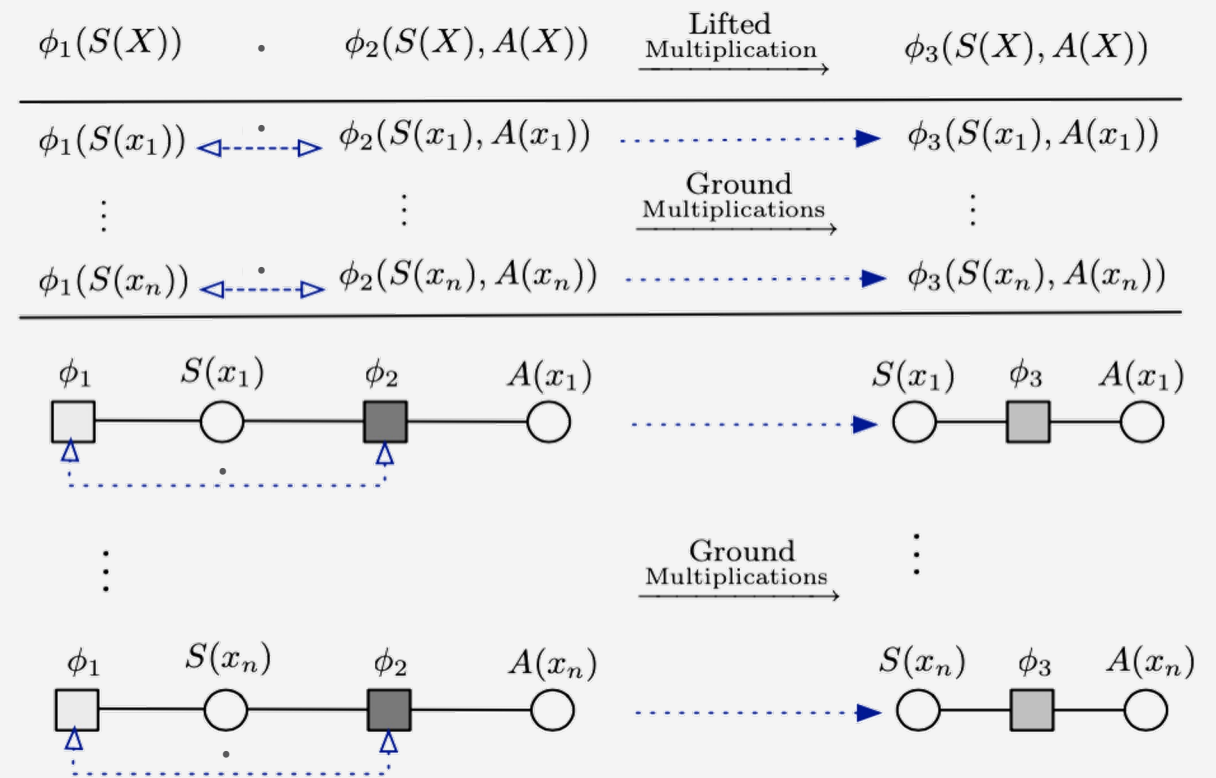
Formal *lifted multiply* operator

- Open question:
What happens if both parfactors represent different number of groundings?

Lifted Multiplication: Trivial Case

- 1-to-1 correspondence between the ground factors of each parfactor
 - E.g., $\phi_1(S(X)) \cdot \phi_2(S(X), A(X))$

Each grounding of X in $\text{gr}(\phi_1(S(X)))$ interacts with one corresponding grounding of X in $\text{gr}(\phi_2(S(X), A(X)))$

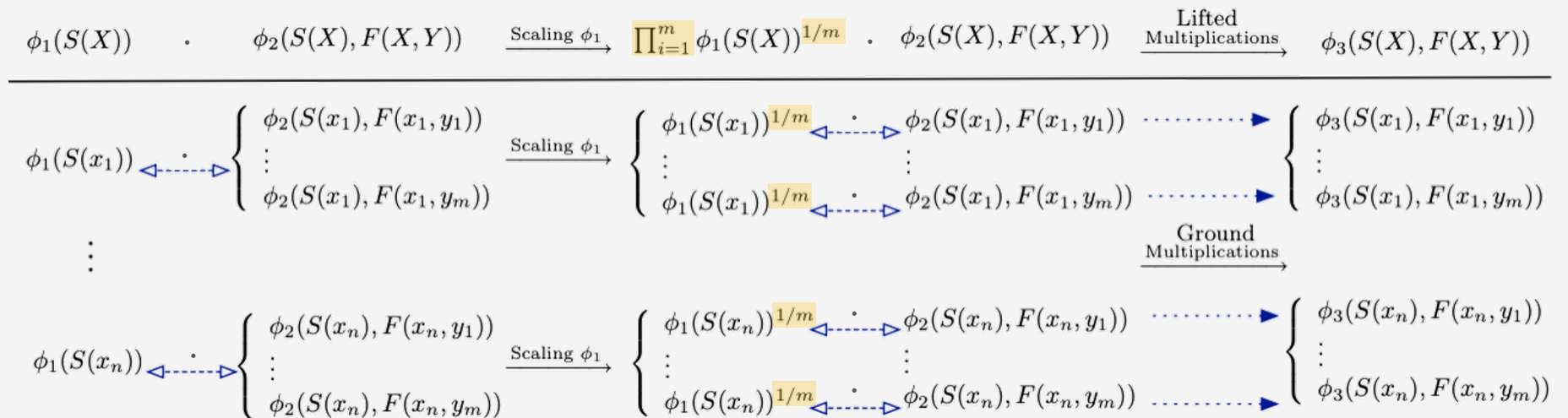


Lifted Multiplication: More General

- 1-to- m correspondence between the ground factors of each parfactor
 - Scaling necessary
 - E.g., $\phi_1(S(X)) \cdot \phi_2(S(X), F(X, Y))$

Distribute $\phi_1(S(X))$ into m factors proportionally

Each grounding of X in $\text{gr}(\phi_1(S(X)))$ interacts with m corresponding groundings of X, Y in $\text{gr}(\phi_2(S(X), F(X, Y)))$



Lifted Multiplication: General Case

- **n -to- m** correspondence between the ground factors of each parfactor
 - **Scaling** necessary in both directions
 - E.g., $\phi_1(S(X), T(X, Z)) \cdot \phi_2(S(X), F(X, Y))$
 - $\text{dom}(X) = \{x_1, \dots, x_k\}, \text{dom}(Z) = \{z_1, \dots, z_n\}, \text{dom}(Y) = \{y_1, \dots, y_m\}$
 - Each grounding of X, Z in ϕ_1 interacts with m groundings of X, Y in ϕ_2
 - Each grounding of X, Y in ϕ_2 interacts with n groundings of X, Z in ϕ_1
 - Scaling:

$$\prod_{i=1}^m \left(\phi_1(S(X), T(X, Z)) \right)^{\frac{1}{m}} \cdot \prod_{i=1}^n \left(\phi_2(S(X), F(X, Y)) \right)^{\frac{1}{n}}$$

Distribute $\phi_1(S(X), T(X, Z))$ into m factors and $\phi_2(S(X), F(X, Y))$ into n factors proportionally

Count Conversion

- Counting a logical variable **binds a logical variable**, i.e., removes logical variable from the logical variables of the parfactor
 - E.g.,
 - $g_1 = \phi_1(\text{Epid}, \text{Nat}(D), \text{Acc}(I)) \rightarrow lv(g_1) = \{D, I\}$
 - $g'_1 = \phi'_1(\text{Epid}, \text{Nat}(D), \#_I[\text{Acc}(I)]) \rightarrow lv(g'_1) = \{D\}$
 - Helps with Precondition 2 of summing out!**
 - SO Precondition 2: PRV to sum out has to contain all logical variables of parfactor
- Operator count—convert
 - Count a logical variable $\rightarrow$ convert a PRV into a (P)CRV
 - Works as an “enabler” for sum—out operator
 - Preconditions for count—convert as well

Formal *count conversion* operator

- Number of instances of logical variable to count identical for all instances of other logical variables in parfactor (to have identical histograms)
- Logical variable to count appears in only one PRV

Generalised Counting

- Count conversion as discussed here, first introduced by Milch et al. (2008)
- Generalised counting by Nima Taghipour et al. (2013)
 1. Count logical variables that appear in more than one PRV
 - E.g., $\phi(Q(X), R(X), S(Y), T(Y))$
 $\rightarrow \phi(\#_X[Q(X), R(X)], S(Y), T(Y))$
 2. Merge CRVs with counted logical variables of the same domain
 - E.g., $\phi(\#_X[Q(X), R(X)])_{C^X}$ and $\phi(\#_Y[Q(Y), R(Y)])_{C^Y}$ with $gr(X|_{C^X}) = gr(Y|_{C^Y})$
 $\rightarrow \phi(\#_X[Q(X), R(X)])_C$
 3. Merge-count a PRV and a CRV with an inequality constraint
 - E.g., $\phi(\#_X[Q(X)], R(Y))_C$ with C encoding $X \neq Y$
 $\rightarrow \phi(\#_X[Q(X), R(X)])_C$

Lifted Absorption

- Remember:
- Observations for groundings of a PRV can be
 - One of the range values
 - Not available (N/A)
- Compactly encode evidence with PRVs and parfactors
 - Within each group:
 - instances are indistinguishable again

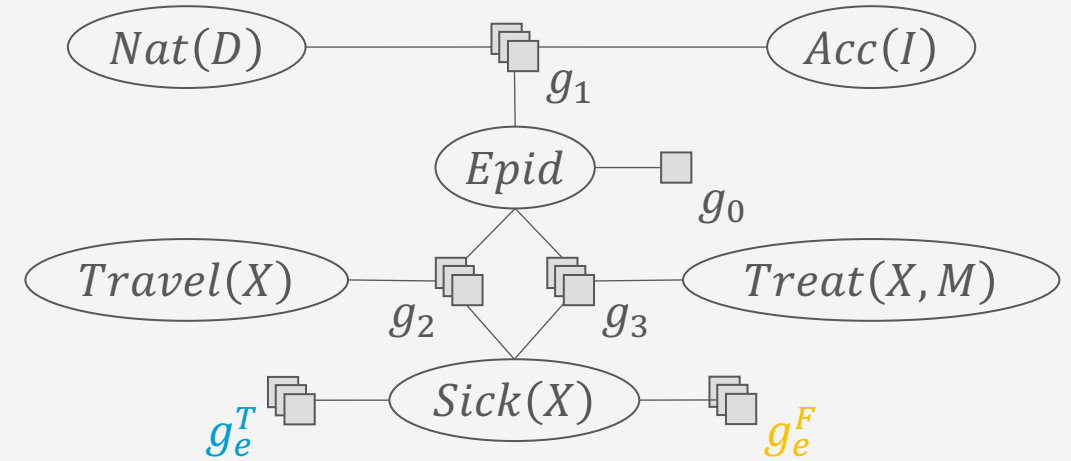
→ Absorb evidence for each group at once using the parfactors

Observations for $Sick(X)$

$Sick(x_1) = Sick(x_2) = \dots = Sick(x_{10}) = \text{true}$
 $Sick(x_{11}) = Sick(x_{12}) = \dots = Sick(x_{20}) = \text{false}$

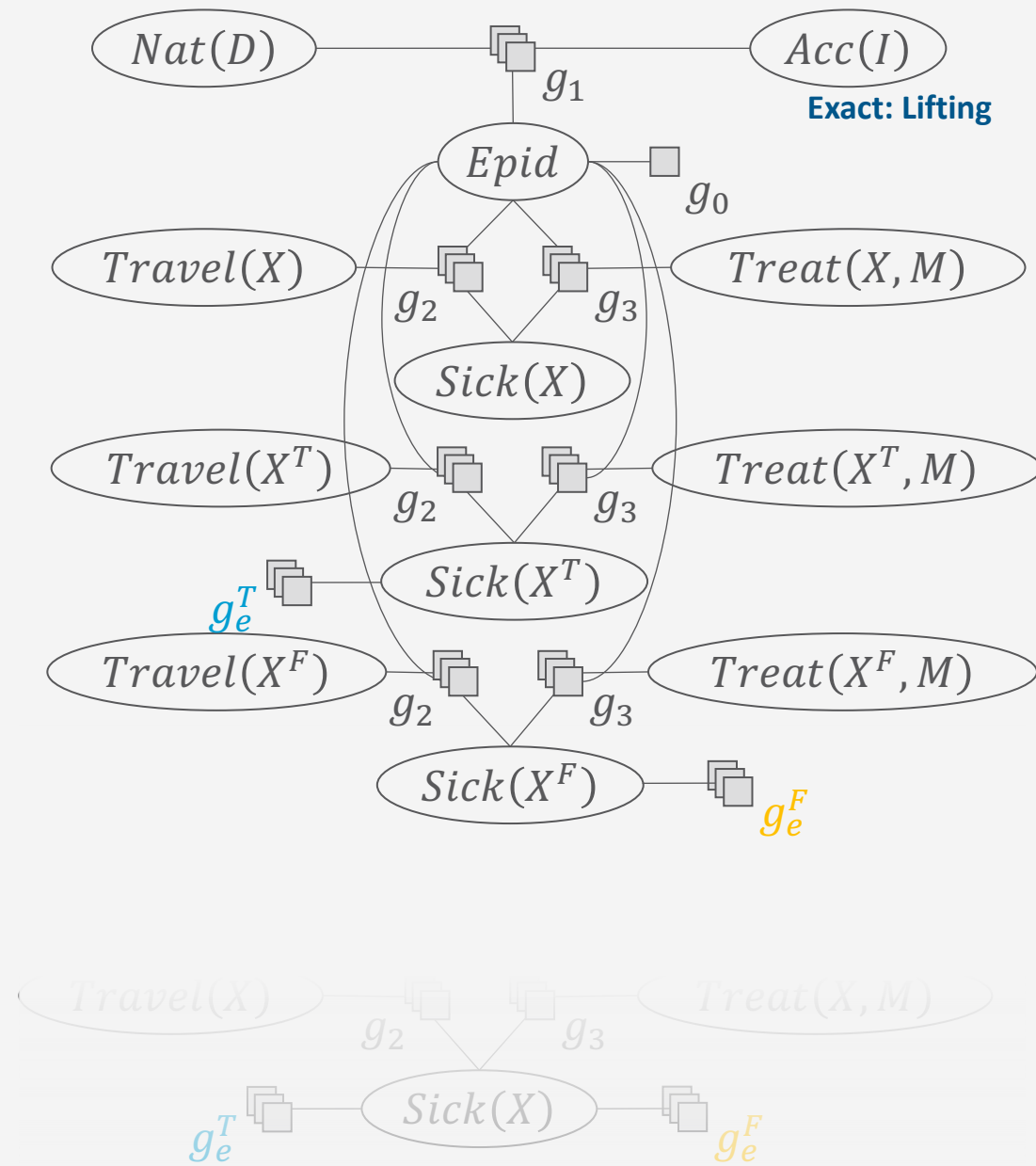
$Sick(X)$	ϕ_e^T
false	0
true	1

$Sick(X)$	ϕ_e^F
false	1
true	0



Lifted Absorption: Shattering on Evidence

- As observations are seldom for all constants in a constraint, parfactors have to be split based on the constants that occur in the observations
 - Only then: absorb applicable evidence in each parfactor individually
 - E.g., given evidence parfactors g_e^T , g_e^F , every parfactor containing $Sick(X)$ has to be split on the constraints: g_2, g_3
- After shattering, absorb each evidence parfactor g_e in each applicable parfactor g_i
 - Possible to interleave shattering and absorption



Lifted Absorption: Example

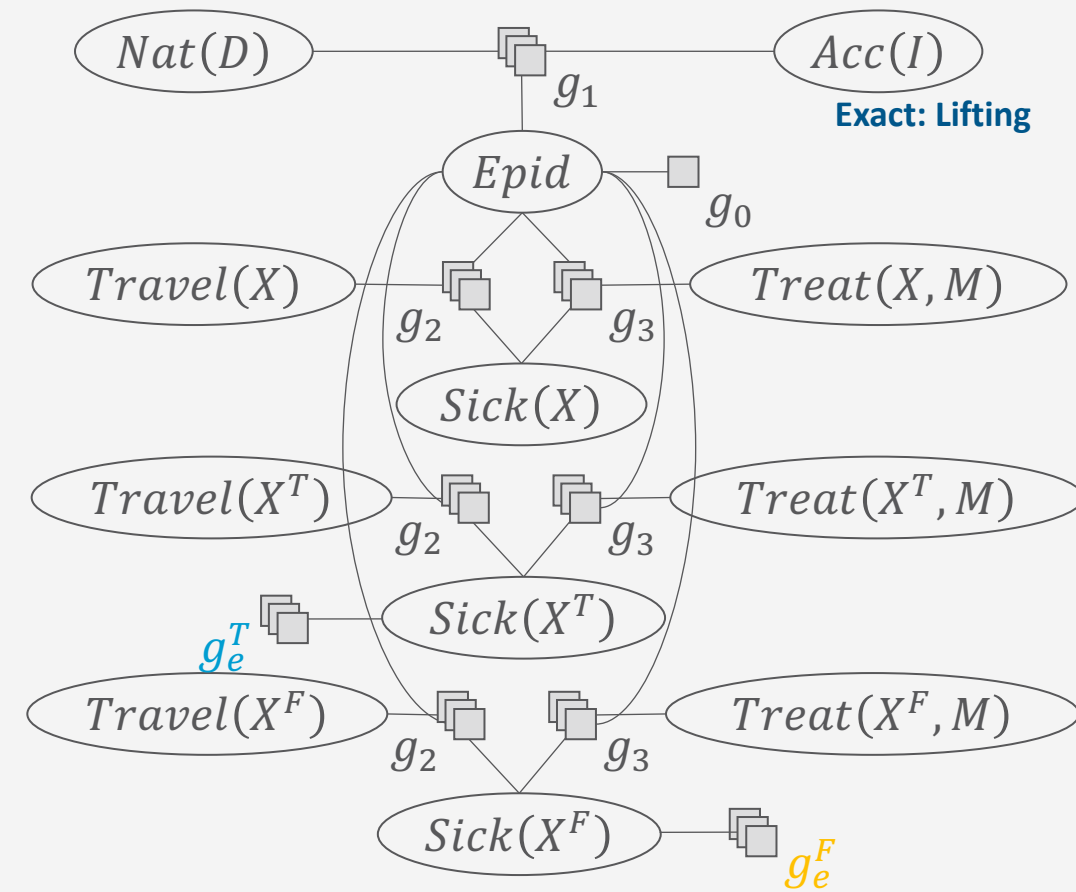
- Absorb g_e^T in g_2 :

$Travel(X)$	$Epid$	$Sick(X)$	ϕ_2
false	false	false	5
false	false	true	0
false	true	false	4
false	true	true	6
true	false	false	4
true	false	true	6
true	true	false	2
true	true	true	9

$Sick(X)$	ϕ_e^T
false	0
true	1

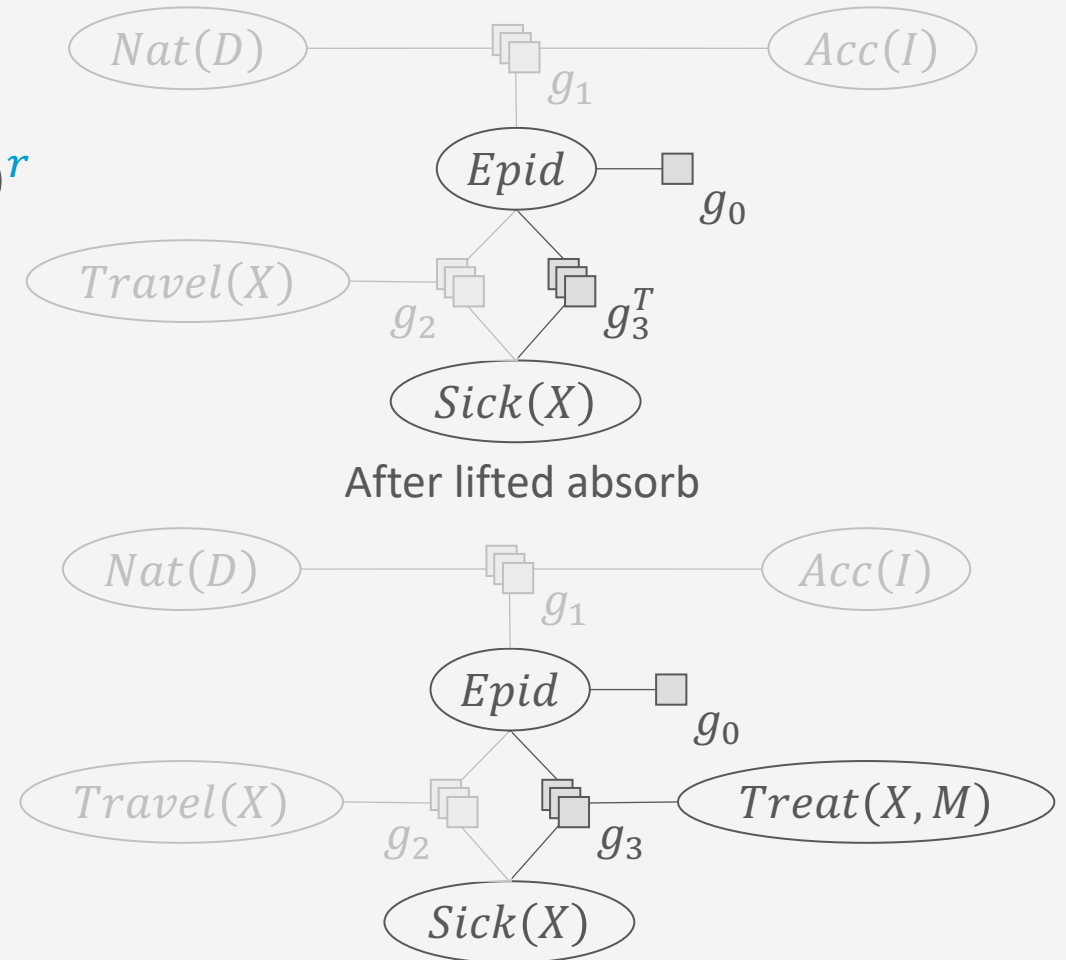
$Travel(X)$	$Epid$	ϕ_2^T
false	false	0
false	true	6
true	false	6
true	true	9

- Same for g_e^T in g_3 , g_e^F in g_2 , g_e^F in g_3



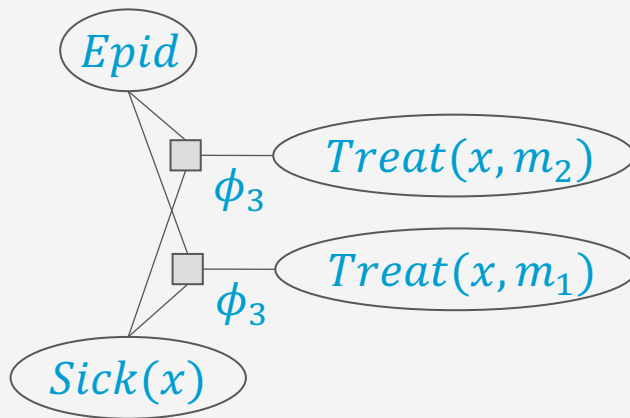
Lifted Absorption: Eliminating a Logical variable

- Output: $g' = \phi'(\mathcal{A}')|_{C'}$
 $\phi'(\dots, a_{i-1}, a_{i+1}, \dots) = \phi(\dots, a_{i-1}, e, a_{i+1}, \dots)^r$
 - with $e = o$ if $A_i = R(Y)$
- E.g., $Treat(X, M) = \text{true} \forall (x, m) \in T$
 - Parfactor g_3 contains $Treat(X, M)$
 - Output: $\phi'(Epid, Sick(X))$
 - Absorbing $Treat(X, M) = \text{true}$ eliminates M
 - $r = \text{ncount}_{M|X}(C) = 2$
 - Potentials in selected lines have to be raised to the power of 2

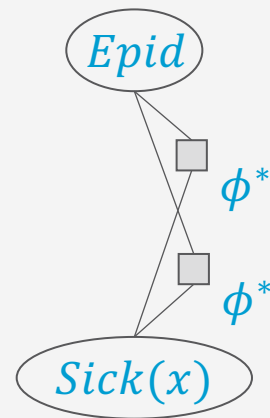


Lifted Absorption: Eliminating a Logical variable

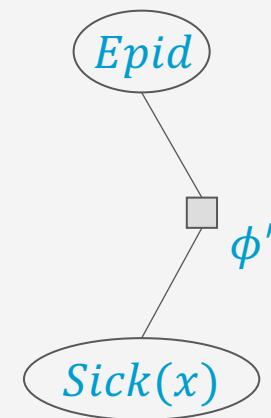
- Equivalent ground case:
 - Absorb $Treat(x, m) = \text{true}$ in r propositional factors for each x
 - Output: $\phi^*(Epid, Sick(x))$ r times (one for each x)
 - Multiply all $\phi^*(Epid, Sick(x))$ into one factor $\phi'(Epid, Sick(x))$, i.e., raise to the power of r



Initial model



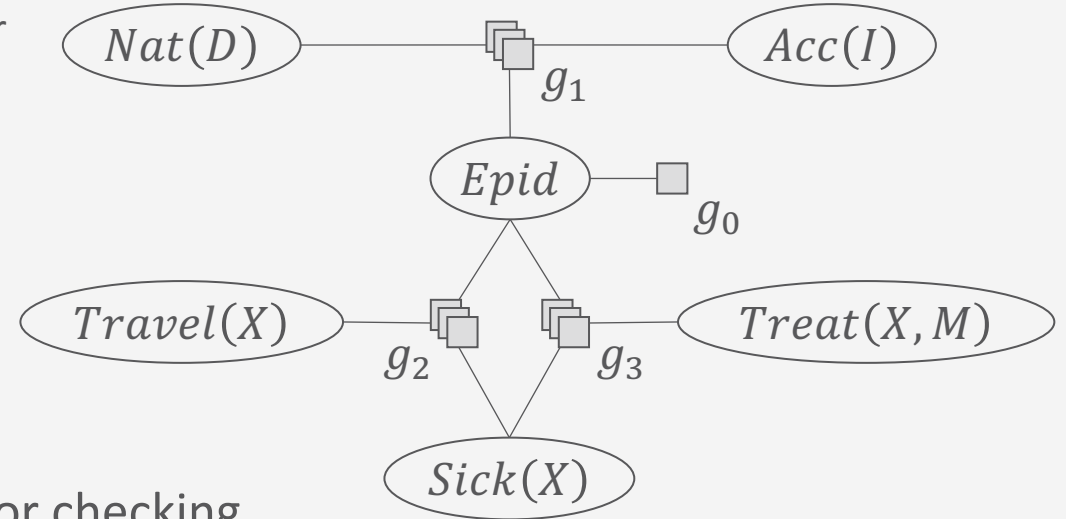
After absorb



After multiply

Types of Shattering

- So far considered: **Pre-emptive shattering**
 - Recursively shattering the model on evidence, query terms, and itself before starting with any calculations
 - Shattering a model on *itself*: Ensure that all sets of constants for logical variables occurring in constraints are either identical or disjoint
 - Allows for introducing one logical variable for each set of constants and T constraints except when an inequality is encoded
 - Avoids splitting during LVE and makes PRV comparisons easier
- **On-demand shattering**
 - Splitting on constraints only if the application of an LVE operator requires it
 - In initial example calculation for $P(\text{Travel}(\text{eve}))$:
Eliminate $\text{Treat}(X, M)$ before splitting of $\text{Sick}(\text{eve})$
 - Does not change complexity of the problem
 - May be hard to determine when to shatter + extra work for checking



LVE: Algorithm

- Assumption:
 - Pre-emptive shattering
 - Ground query terms
 - Set of propositional random variables, instances (groundings) of PRVs,
- Inputs:
 - Model $G = \{g_i\}_{i=1}^n$
 - Query terms Q
 - Evidence e encoded in evidence parfactors $\{g_e\}_{e=1}^E$
- Output:
 - Parfactor $g = \phi(Q)$
 - Encodes the a-posteriori probability distribution of Q given e : $P(Q|e)$

LVE: Algorithm

$\text{LVE}(G, Q, \{g_e\}_{e=1}^E)$

$G \leftarrow$ Shatter G on Q , $\{g_e\}_{e=1}^E$, and on itself

$G \leftarrow$ Absorb $\{g_e\}_{e=1}^E$ in G

while G contains non-query terms **do**

if a PRV A fulfils the preconditions of sum-out **then**

$G \leftarrow$ Apply sum-out to A in G

else

$G \leftarrow$ Apply an enabling operator (multiply, count-convert, expand, count-normalise, split, ground) on some parfactors in G

$g \leftarrow$ Multiply all parfactors in G into one parfactor

$g \leftarrow$ Normalise the potentials in g

return g

How to choose?

G may contain several parfactors $\phi_i(Q)$

LVE: Heuristics

- Important for an implementation
 - Cannot search all possible permutations of all possible operator applications
- Preconditions of lifted operators already restrict possible elimination order
- One possible greedy heuristics (as used in the upcoming implementation):
 - Choose sum-out operations over any other operation
 - Explicitly written down in algorithm
 - Only consider multiplication if the arguments of the two parfactors are the same or ground
 - Avoid scaling
 - Choose operation that results into the smallest parfactor(s) to be added to G
 - If same size: choose at random
 - May result in sub-optimal application order or unnecessary applications
 - E.g., if a grounding is unavoidable, the heuristics may lead to various count conversions being applied before grounding as the result of a count conversion is usually smaller in size than the result of grounding the same logical variable

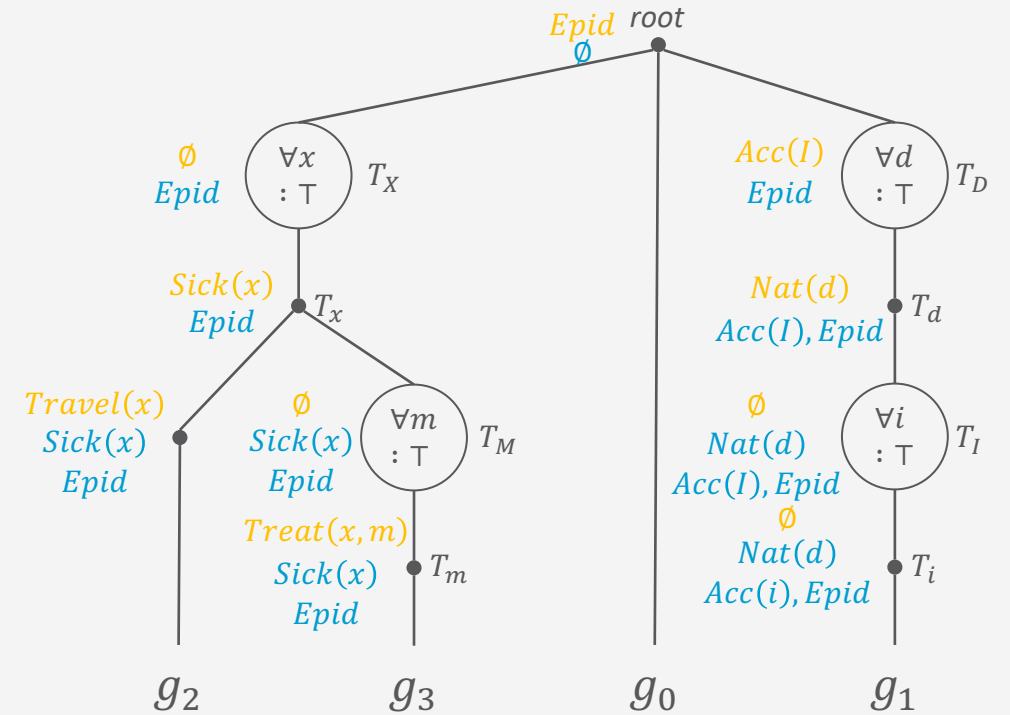
Interim Summary

- LVE lifted operators
 - Eliminate PRVs: lifted summing out, lifted absorption
 - Enable elimination: lifted multiplication, count conversion, splitting
 - Other based on splitting: expand, ground, count-normalise
- Shattering = splitting on query terms, evidence, model constraints
 - Pre-emptive, on-demand
- LVE algorithm
 - Heuristics

Formal Analysis

Complexity, runtimes

Tractability, completeness



Complexity of Probabilistic Inference in PRMs

- *Informally*, LVE complexity still worst case size of an interim result *times* number of eliminations
- Use a so-called first-order dtree (**FO dree**), to get worst case size of an interim result characterised by so-called **lifted width**
 - In dtree representation of VE for $gr(G)$, duplicate subtrees whenever a lifted sum-out applicable in G
 - In FO dtree of LVE for G , representative subtree for lifted sum-out (compactly encode duplicate subtrees)

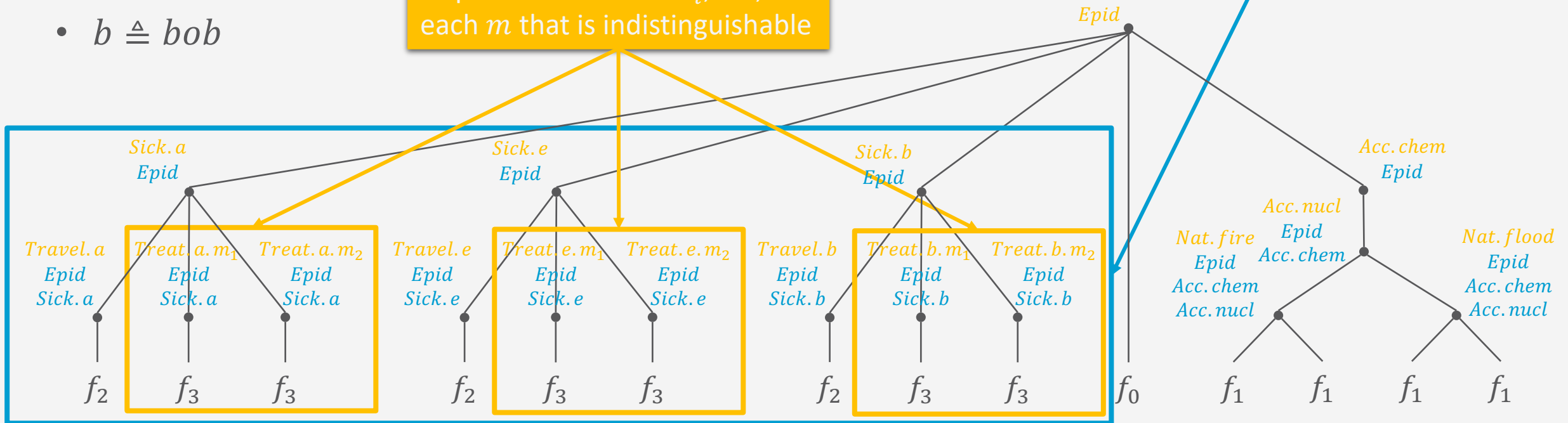
Dtree with Repeating Structures

- Decomposition of $\text{gr}(G)$
 - $a \triangleq \text{alice}$
 - $e \triangleq \text{eve}$
 - $b \triangleq \text{bob}$

Aim for a node that basically says:
 $\forall x$ in this submodel

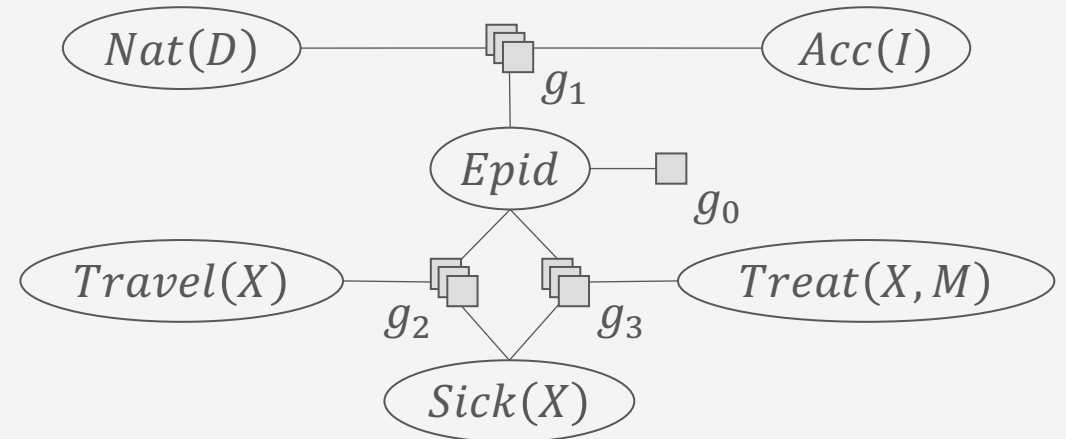
Duplicated for *alice*, *bob*, i.e., for each x that is indistinguishable

Duplicated for each m_i , i.e., for each m that is indistinguishable



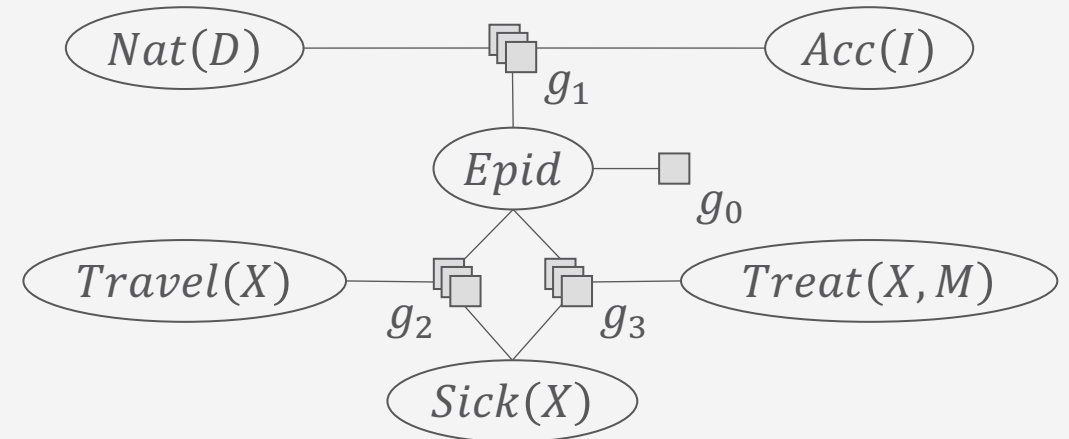
Decomposition into Partial Groundings

- Introduce a new inner node: **DPG node** denoted $\forall x : C$
 - DPG = Decomposition into partial groundings
 - Replaces a logical variable with a representative constant
 - The resulting model/subtree is identical for each constant represented
 - Allows for considering the resulting model without the grounded logical variable for further decomposition (top-down)
 - E.g., submodel below *Epid* in the graph
 - Logical variable X appears in each parfactor
 - Grounding X leads to copies of the same submodel
 - Replace X with representative $x \rightarrow$ partial grounding
 - Whatever you do to x applies to all constants represented
 - Represent that $\forall x : C, C$ a constraint, the subtree would be identical



FO Dtree Construction

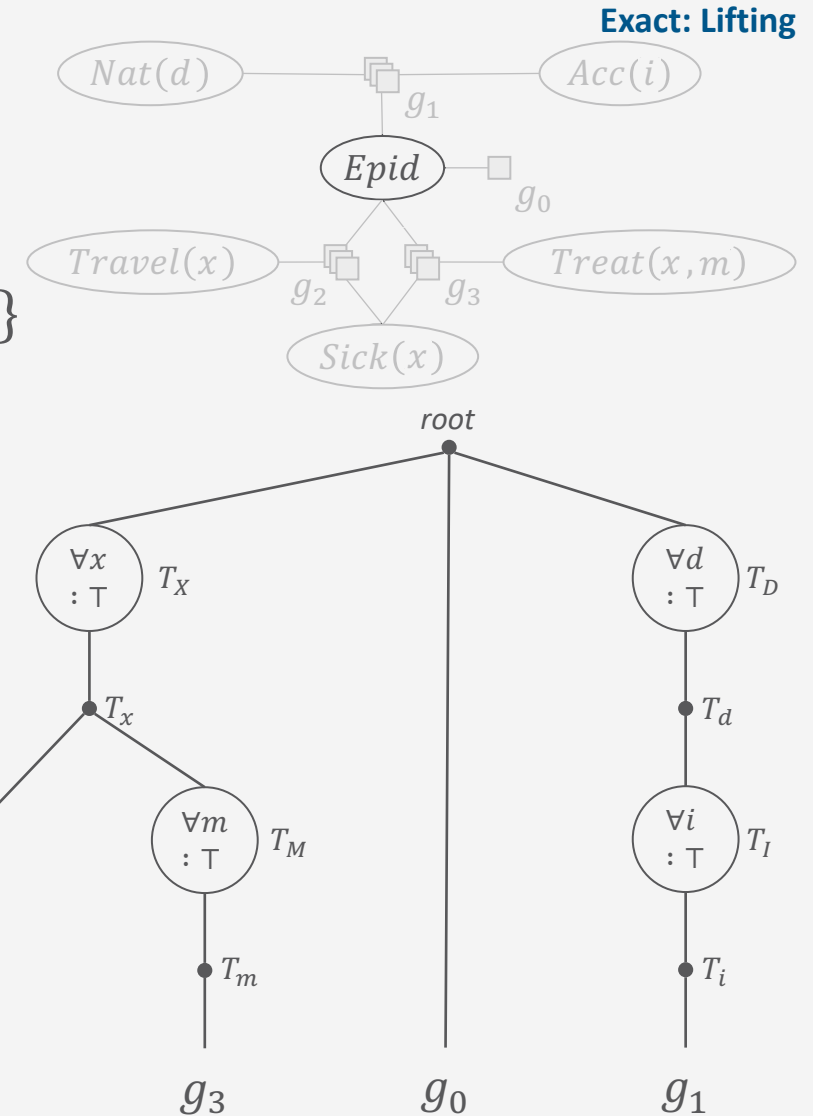
- Recursively, starting with G as the current model G' at the root
 - Check if there exists logical variable X that allows for a DPG in G'
 - If so, make current node a DPG node T_X for X , replace X with representative x , i.e., apply $\theta = \{X \rightarrow x\}$ to G' , add child node T_x with $G' = G\theta$ as current model
 - If parent node is a DPG node $T_{x'}$ as well, with current node being $T_{x'}$, add new DPG node T_X as child of $T_{x'}$
 - Otherwise: Partition G' on logical variables (if exist) or random variables into $\{G'_i\}_{i=1}^n$
 - Add a child node for each G'_i with G'_i as current model
- Until
 - All logical variables replaced by representatives and
 - Only one parfactor per partition



FO Dtree: Example

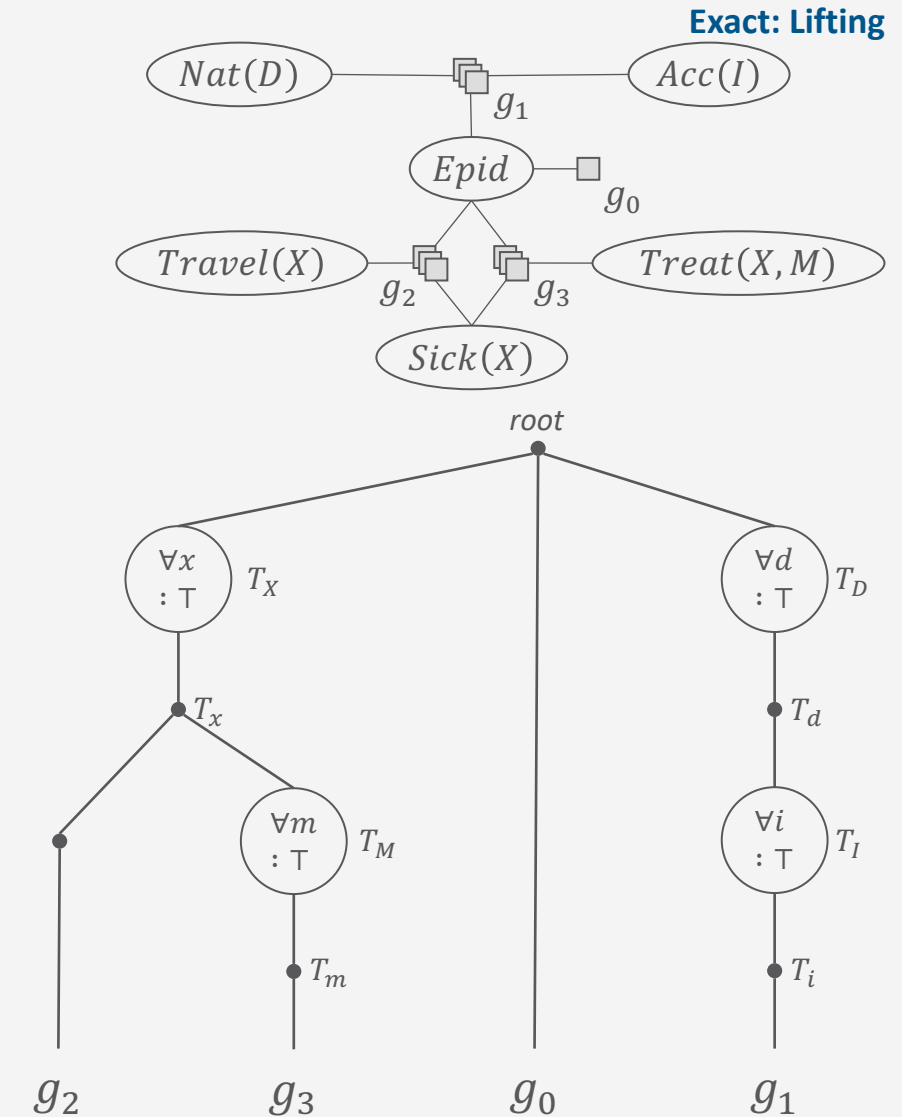
Root: In G , no logical variable for a DPG

- Partition based on, e.g., $X \rightarrow G_0 = \{g_2, g_3\}, G_1 = \{g_1\}, G_2 = \{g_0\}$
- Left: $G_0 = \{g_2, g_3\}$
 - DPG with $X \rightarrow$ Replace X with x
 - No logical variable for DPG
 - Partition based on $M \rightarrow G_{01} = \{g_2\}, G_{02} = \{g_3\}$
 - Left: $G_{01} = \{g_2\}$
 - No logical variables and only one parfactor left
 - Right: $G_{02} = \{g_3\}$
 - DPG with $M \rightarrow$ Replace M with m
 - No logical variables and only one parfactor left
- Right: $G_1 = \{g_1\}$
 - DPG with $D \rightarrow$ Replace D with d
 - DPG with $I \rightarrow$ Replace I with i
 - No logical variables and only one parfactor left



FO Dtree Definition

- An FO dtree has three node types
 - DPG node T_X
 - Represents a DPG (top-down)
 - Given by a tuple (X, x, C) with X a logical variable, x a representative constant, and C a constraint
 - In this lecture: $C = \top$
 - Denoted $(\forall x : C)$ in graphical representation of the tree
 - VE node T
 - Represents a partitioning
 - All inner nodes that are not DPG nodes
 - Leaf node L
 - Contains a parfactor, grounded with representative constants



FO Dtree Properties (as before)

- Cutset**

$$\text{cutset}(T) = \left(\bigcup_{T_i, T_j \in \text{Ch}(T)} \text{rv}(T_i) \cap \text{rv}(T_j) \right) \setminus \text{acutset}(T)$$

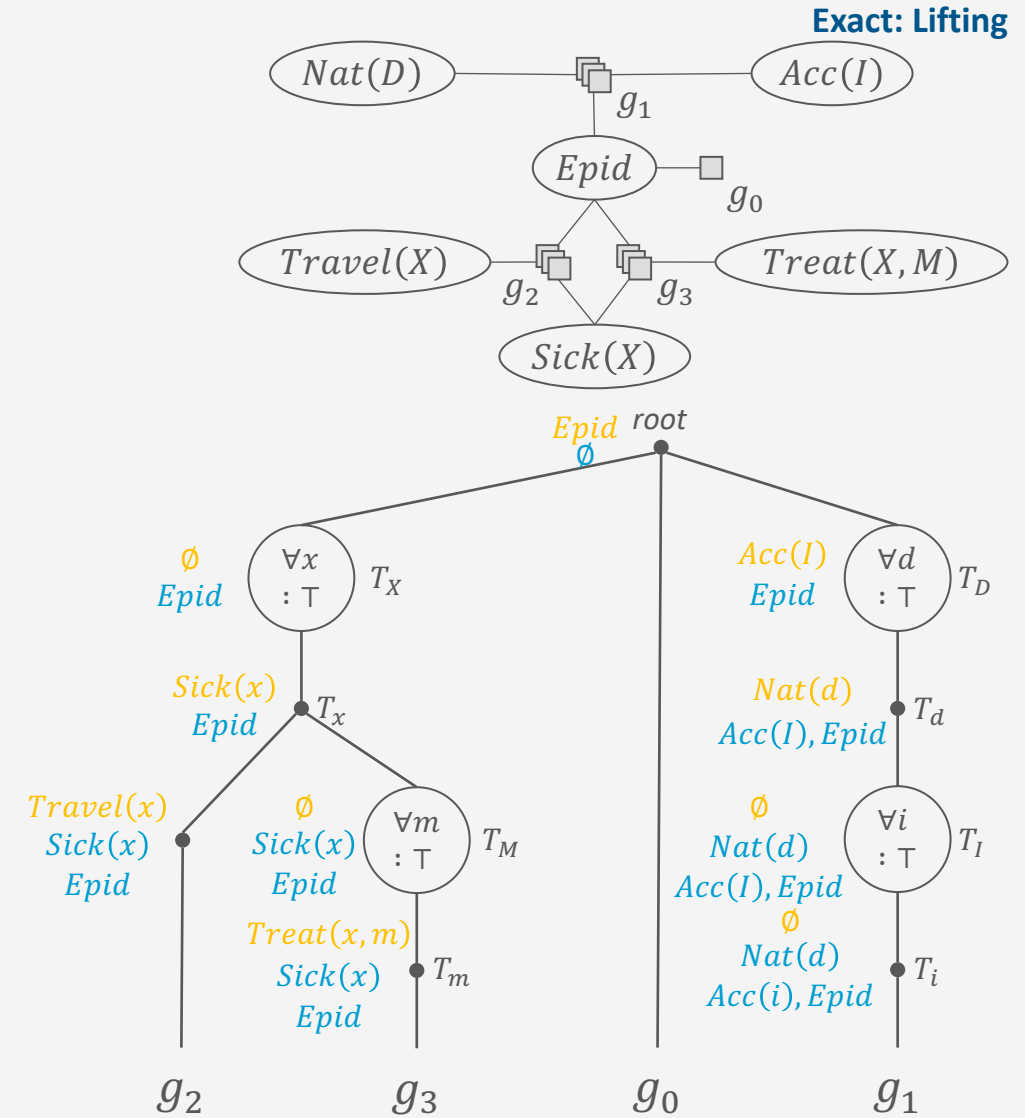
- Ancestor cutset: $\text{acutset}(T) = \bigcup_{T' \in \text{Anc}(T)} \text{cutset}(T')$
- Definitions of Ch if DPG nodes T_X involved
 - Θ_X all grounding substitutions of X
 - $\text{Ch}(T_X) = \{T_{x\theta} \mid T_x \text{ is child of } T_X \wedge \theta \in \Theta_X\}$
 - $\text{Ch}(T_y) = \{T_{x\theta} \mid T_x \text{ is child of } T_y \wedge \theta \in \Theta_X\}$

- Context**

$$\text{context}(T) = \text{rv}(T) \cap \text{acutset}(T)$$

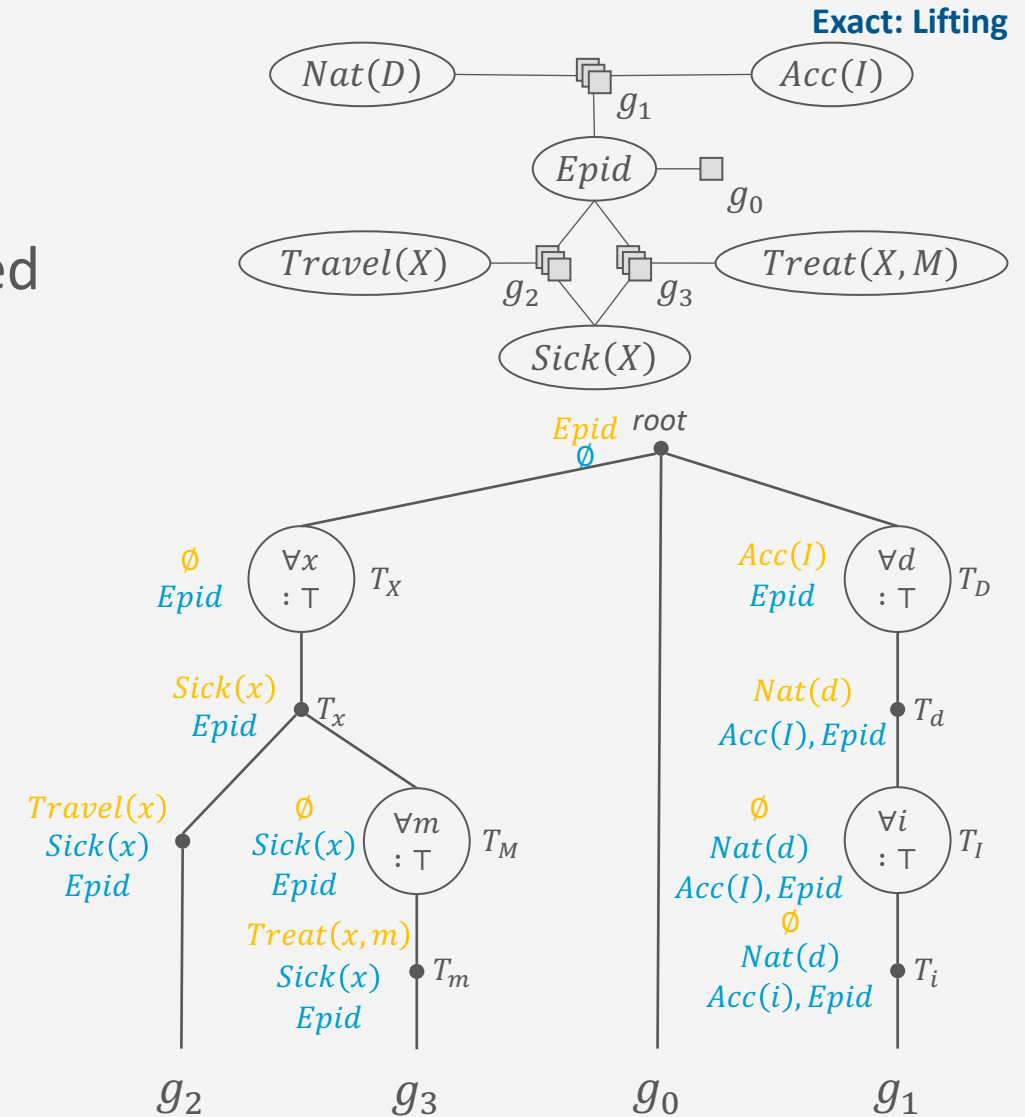
- Cluster**

$$\text{cluster}(T) = \text{cutset}(T) \cup \text{context}(T)$$



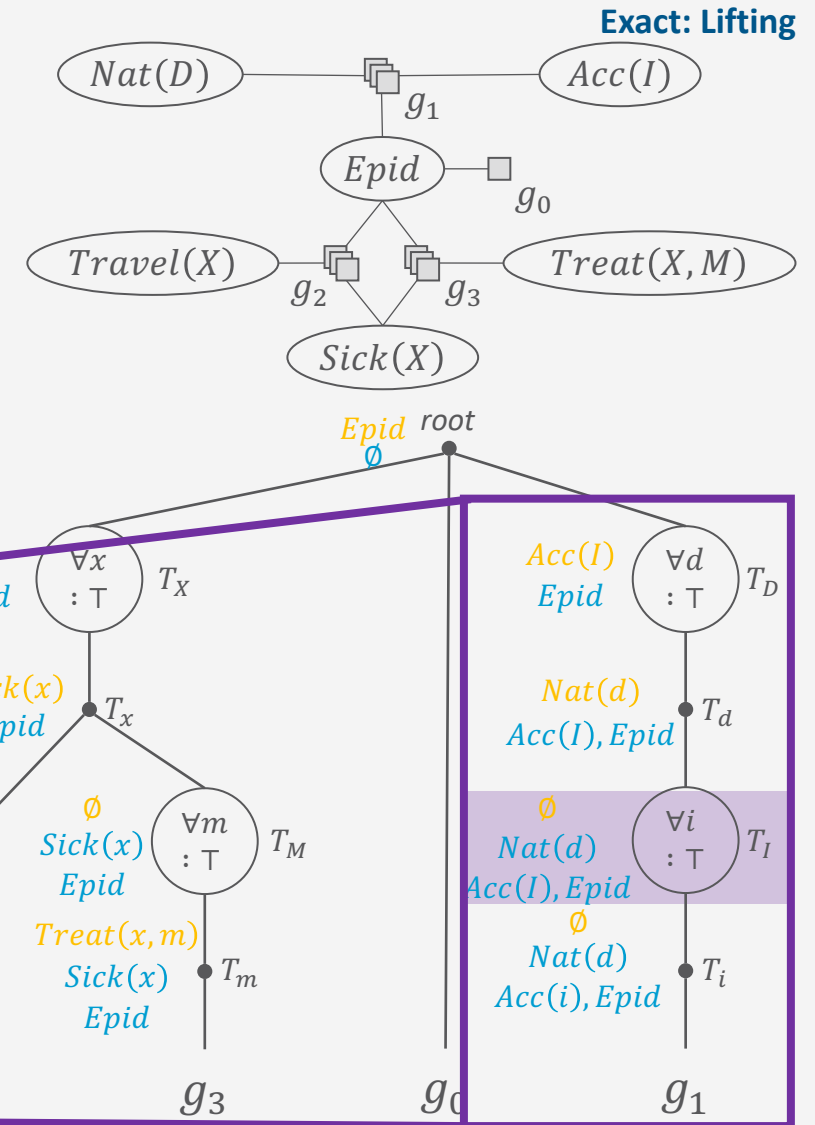
FO dtree: Bottom-up Interpretation

- If only lifted summing out and multiplication involved
 - VE node: (Multiplication), elimination
 - DPG node: Exponentiation
 - Cutset and context interpretation
 - Cutset of DPG child VE node:
PRV summed out in representative summing out
 - Cutset of DPG node:
Exponentiation for logical variable of DPG node
 - Context:
All other PRVs involved at this point in operation



FO dtree: Bottom-up Interpretation

- If DPG logical variable occurs in the context of its DPG node:
Count conversion necessary!
- E.g., $Acc(I)$ in context of DPG node T_I
- Shows “only” that PRV not directly eliminable
 - Occurs when eliminating $Nat(d)$ at T_d
 - No direct interpretation in terms of LVE operations
- Rework FO dtree to represent calculations in count-converted model
 - E.g., consider model with $\#_I[Acc(I)]$ instead of $Acc(I)$

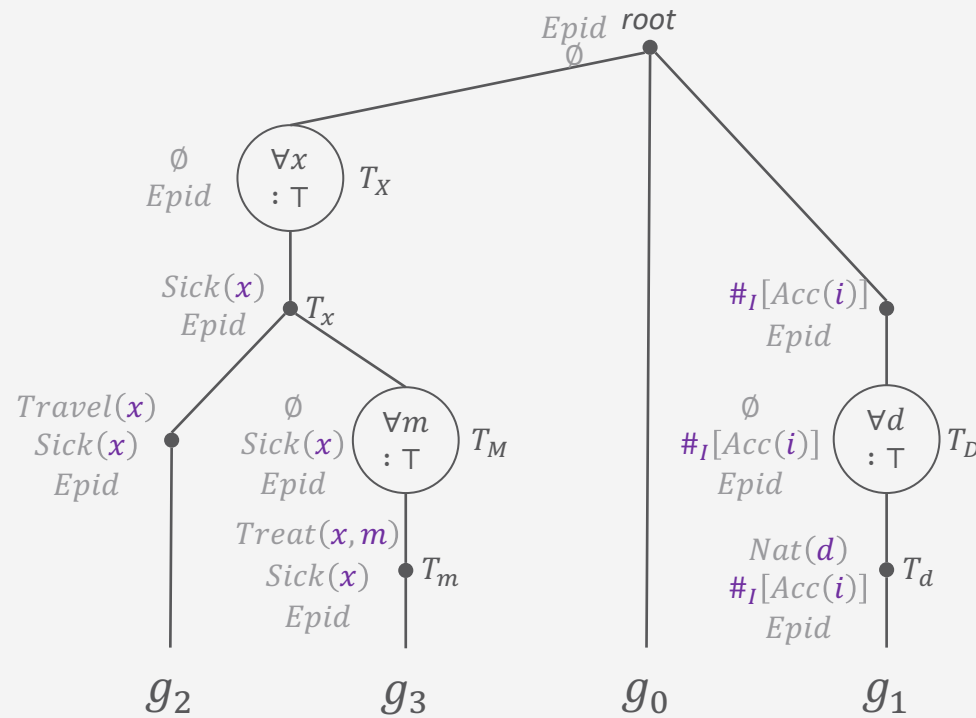


Liftability

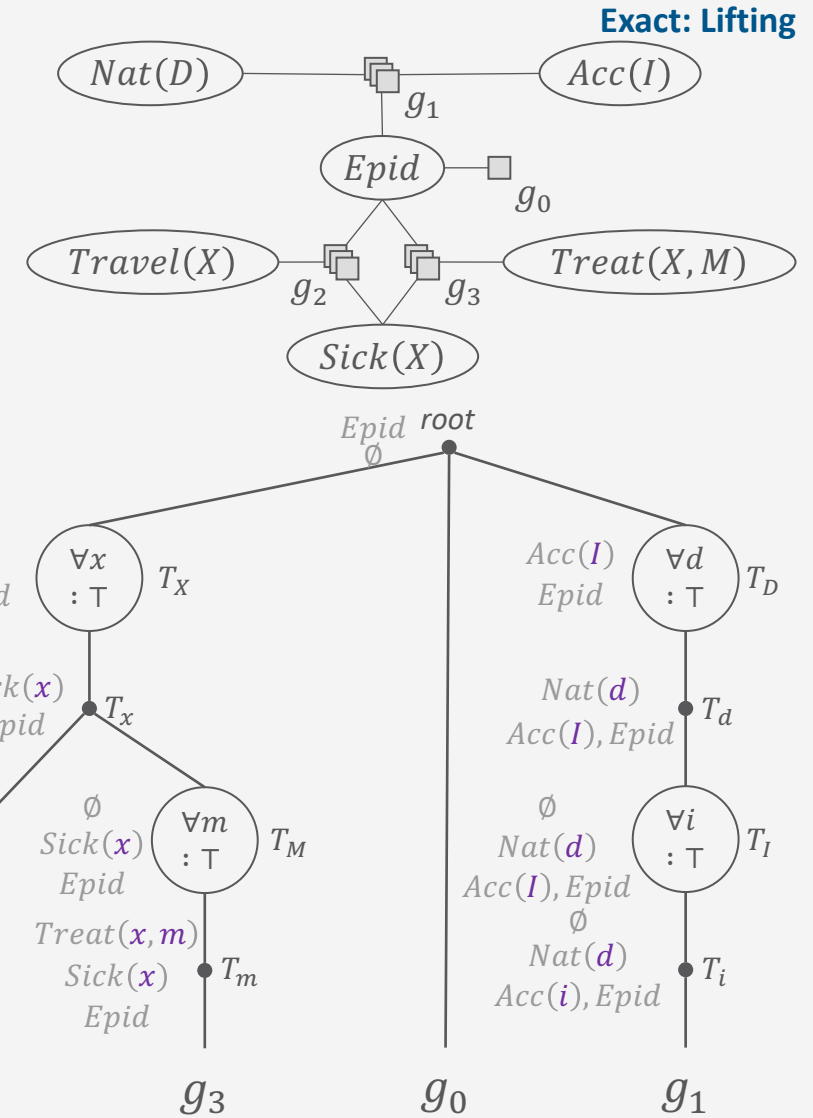
- Given an FO dtree T for a model G
 - If the clusters of T only consist of *PRVs with representative constants* and *PRVs with one logical variable*, then G has a lifted solution
 - Lifted solution: no groundings necessary; only lifted calculations (sum—out, multiply, count—convert)
 - PRVs only with representative constants $\rightarrow$ lifted summing out possible
 - PRVs with one logical variable $\rightarrow$ count conversion necessary (and possible)
 - Called countable
 - T is called **liftable**
 - Apply the count conversions to the countable logical variables, transforming $G \rightarrow$ resulting FO dtree T' called **counted**
- For complexity analysis: Concentrate on models with liftable (counted) FO dtrees
 - Otherwise: the worst case is grounding G and performing VE

Liftability: Example FO Dtree

- Only PRVs with representative constants or one logical variable in the clusters
→ liftable
- If counting I and reworking the FO dtree
→ counted, liftable



counted, liftable

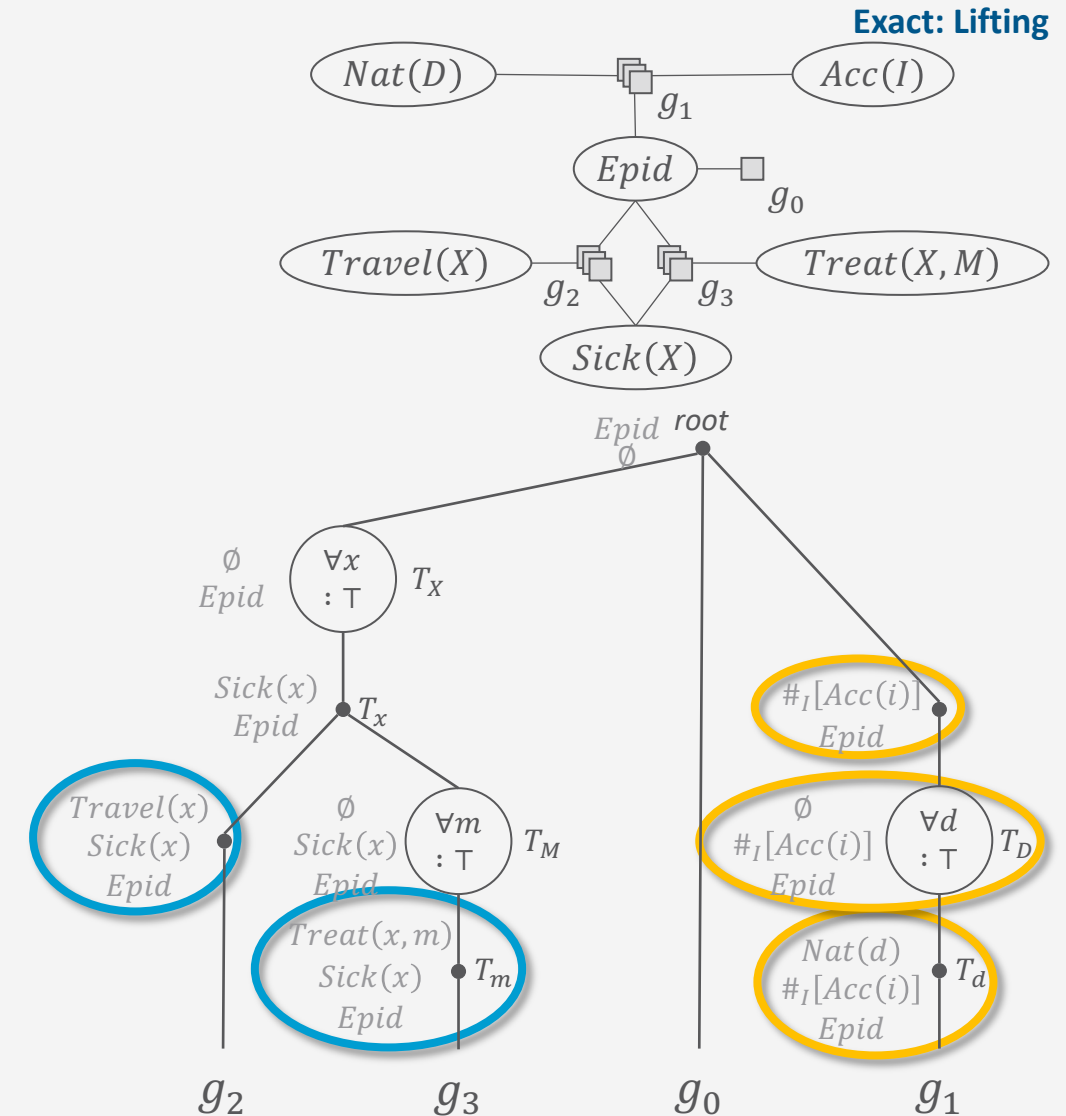


Exact: Lifting

liftable

Lifted Width

- Lifted notion of tree width
 - Bound worst-case parfactor size
- Given liftable, counted FO dtree T
 - Clusters for each node in T
- Lifted width $w_T = (w_g, w_{\#})$
 - w_g largest **ground width**
 - Largest number of PRVs with representative constants in any cluster
 - $w_{\#}$ largest **counting width**
 - Largest number of CRVs in any cluster
 - E.g., $w_T = (w_g, w_{\#})$ with $w_g = 3$, $w_{\#} = 1$



Worst-case Parfactor Size

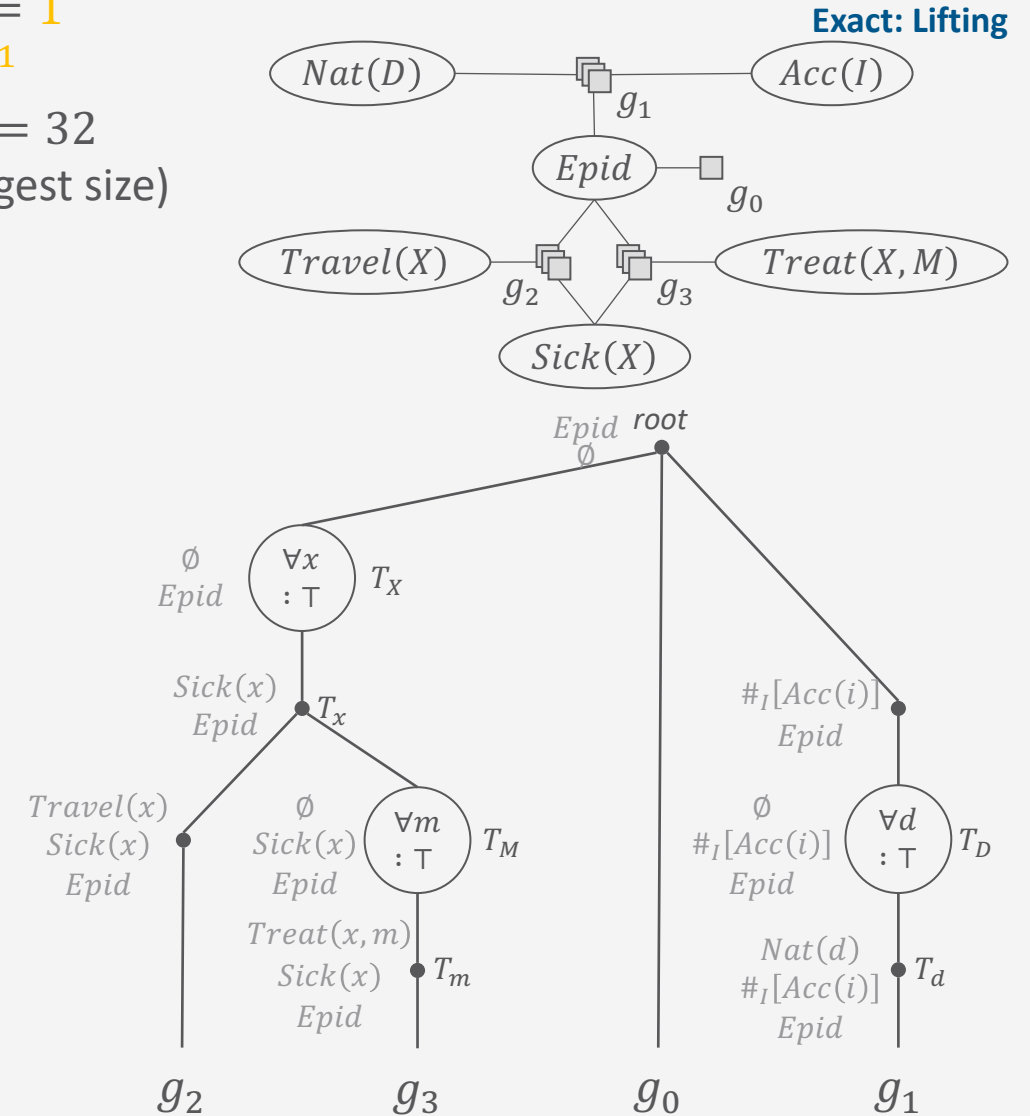
- Given lifted width $w_T = (w_g, w_\#)$
- Worst-case parfactor size:
 - Worst case: $w_g + w_\#$ variables in one parfactor
 - Worst-case range size for the w_g PRVs:

$$r = \max_{A \in \text{erv}(G)} |\text{ran}(A)|$$
 - Worst-case range size for the $w_\#$ CRVs:

$$\binom{n_\# + r_\# - 1}{n_\# - 1} \leq n_\#^{r_\#}$$
 - $n_\#$ largest domain size of any counted logical variable
 - $r_\#$ largest range size of any of the PRVs in the CRVs
 - Number of mappings by w_g and $w_\#$:

$$r^{w_g} \cdot (n_\#^{r_\#})^{w_\#} = r^{w_g} \cdot n_\#^{r_\# w_\#}$$

E.g., with $w_g = 3, w_\# = 1$
 $2^3 \cdot |\text{dom}(I)|^{2 \cdot 1}$
 $= 2^3 \cdot 2^2 = 8 \cdot 4 = 32$
 • $32 > 12$ (actual largest size)

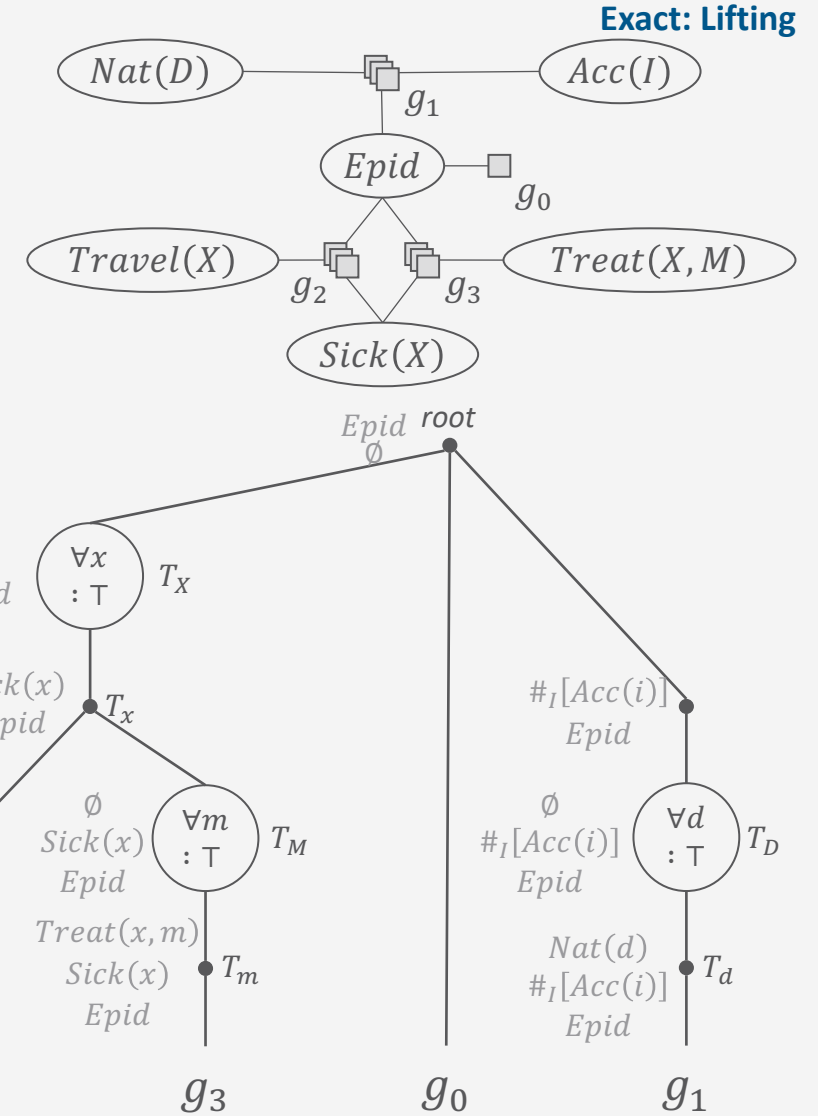


Complexity

- Worst-case parfactor size $r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}}$
- Complexity of lifted operations
 - Multiplication (goes through each line of each parfactor):
 $O(r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$
 - Summation (goes through each line):
 $O(r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$
 - Exponentiation (goes through each line):
 $O(\log_2(n) \cdot r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$
 - n largest overall domain size
 - Count conversion (goes through each line of parfactor):
 - Multiplication and exponentiation:
 $O(\log_2(n_{\#}) \cdot r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$
 - Bounded by $O(\log_2(n) \cdot r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$

E.g., with $w_g = 3, w_{\#} = 1$
 $r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}} = 32$

- $\log_2(|dom(X)|) \cdot 32$
- $\log_2(|dom(I)|) \cdot 32$



Complexity

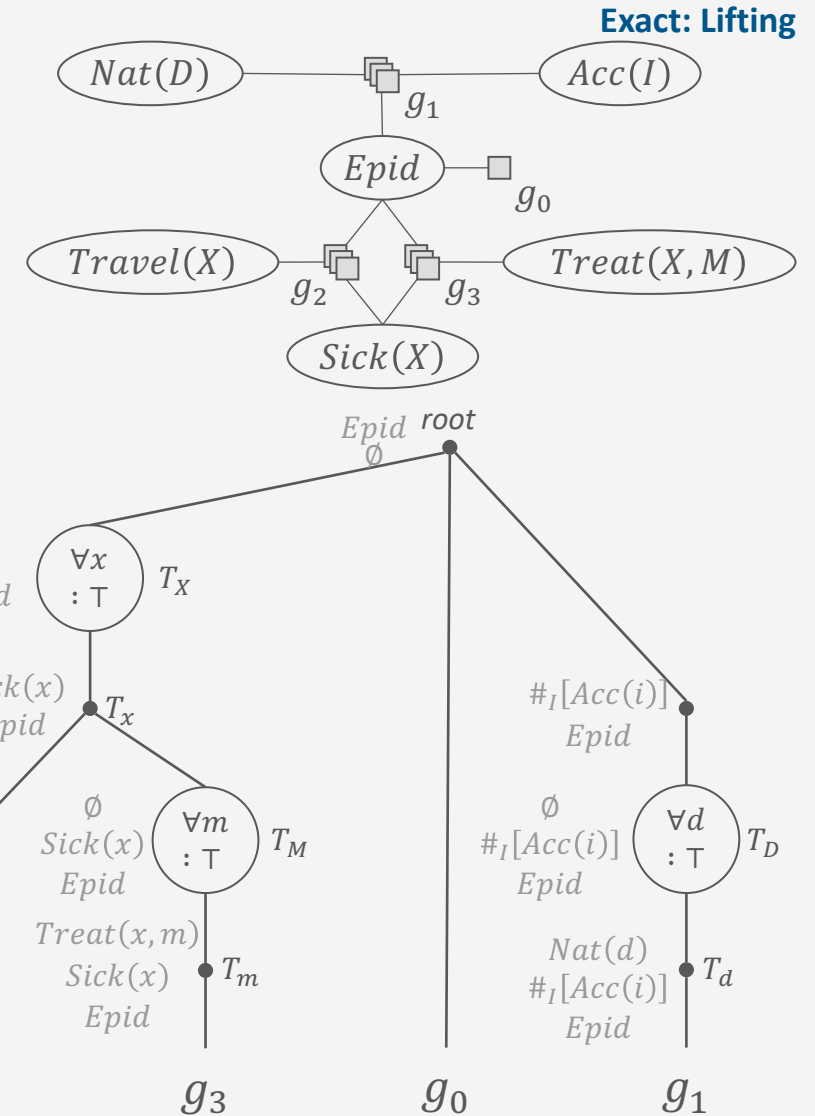
- Worst-case parfactor size $r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}}$
- Complexity of lifted operations
 - Bounded by $O(\log_2(n) \cdot r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$
- Complexity of LVE given a liftable FO dtree T

$$O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}})$$

- n_T : number of inner nodes in T
- w_g : bounded from below by $\max_{g \in G} |\text{rv}(g)|$

E.g., with $w_g = 3, w_{\#} = 1$
 $r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}} = 32$

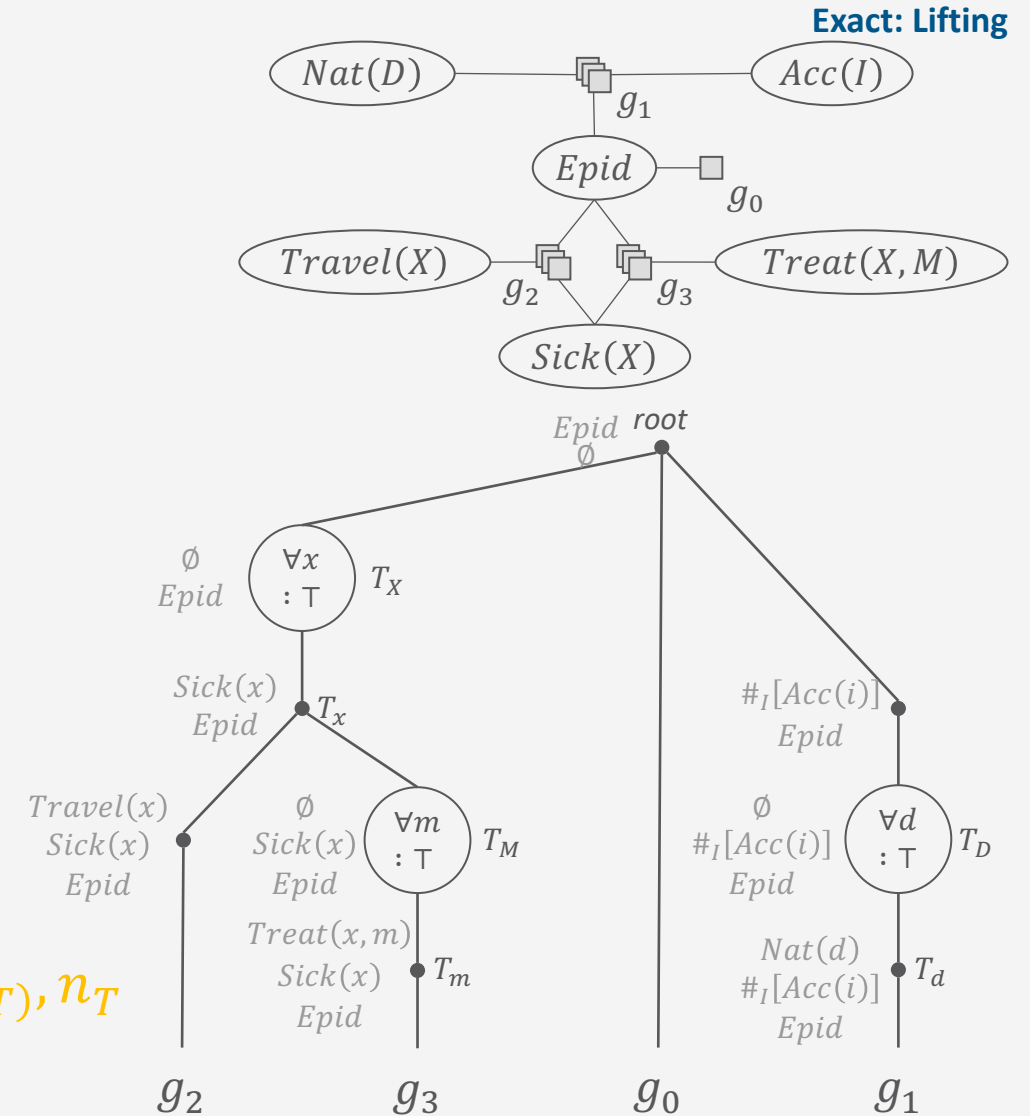
- $\log_2(|\text{dom}(X)|) \cdot 32$
- $9 \cdot \log_2(|\text{dom}(X)|) \cdot 32$



Comparison

- Complexity of LVE given a liftable, counted FO dtree T for a counted model G :

$$O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r \cdot w_{\#}})$$
 - $n_T = |\text{rv}(G)| + |\text{lv}(G)|$
- Complexity of VE: $O(n_{\text{gr}(T)} \cdot r^w)$
 - $n_{\text{gr}(T)} = |\text{gr}(\text{rv}(G))|$
- If no count conversions involved, i.e., $w_{\#} = 0$,
 - $n^{r \cdot 0} = 1 \rightarrow O(n_T \cdot \log_2(n) \cdot r^{w_g})$
 - $w = w_g$
 - Difference in $\log_2(n)$ for lifted computations and $n_{\text{gr}(T)}, n_T$
 - More noticeable if domain sizes increase ($n_{\text{gr}(T)} \gg n_T$)

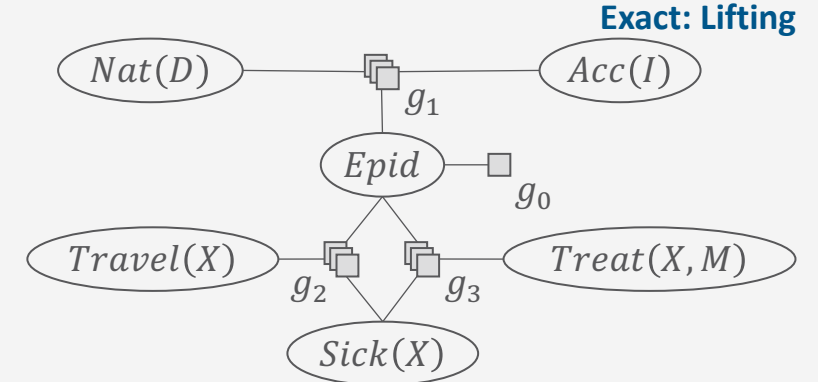


Comparison

- If count conversions involved, i.e., $w_{\#} > 0$,
 - $w \gg (w_g + w_{\#})$
 - CRV with counted logical variable of domain size n appears grounded in a factor
 - With one count conversion, $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}})$ vs. $O(n_{\text{gr}(T)} \cdot r^{n+c})$
 - c the number of random variables also occurring in the cluster
 - E.g., with $c = 2$:

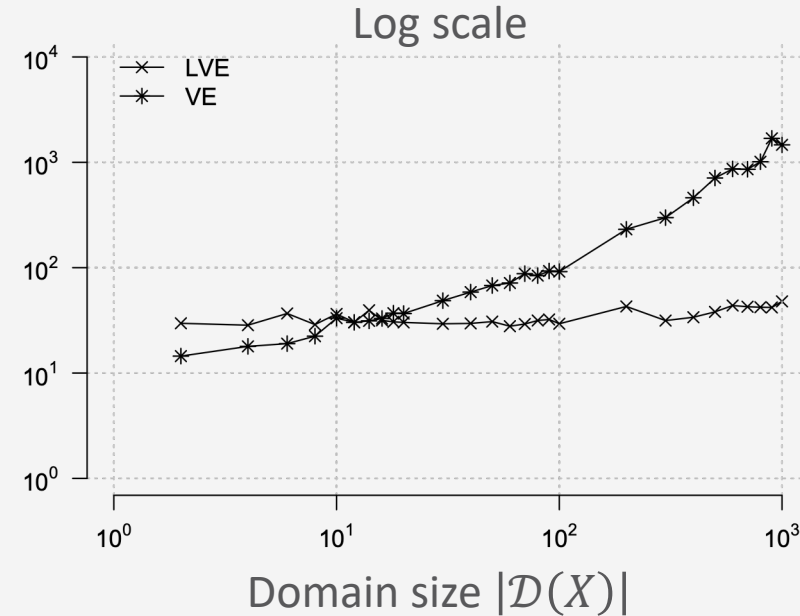
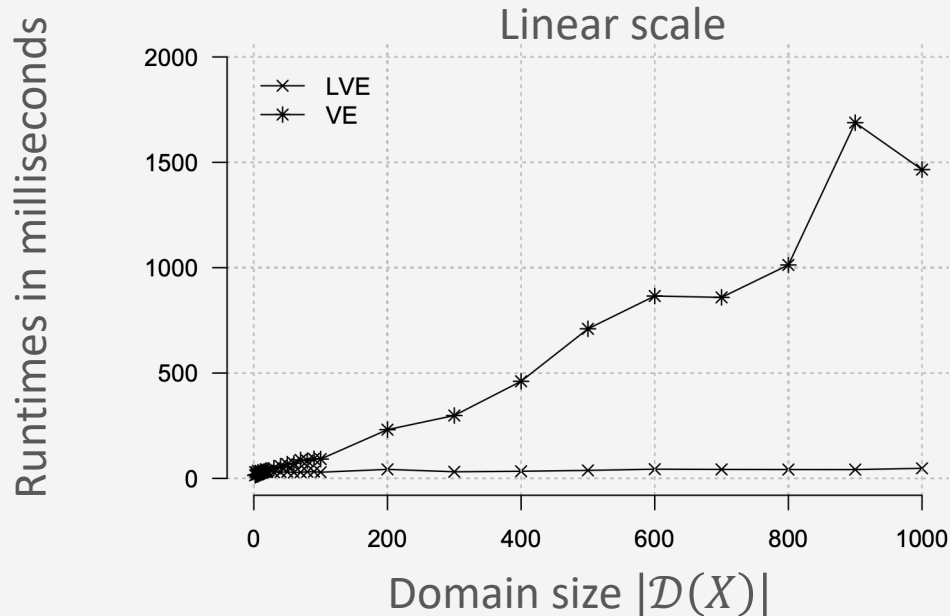
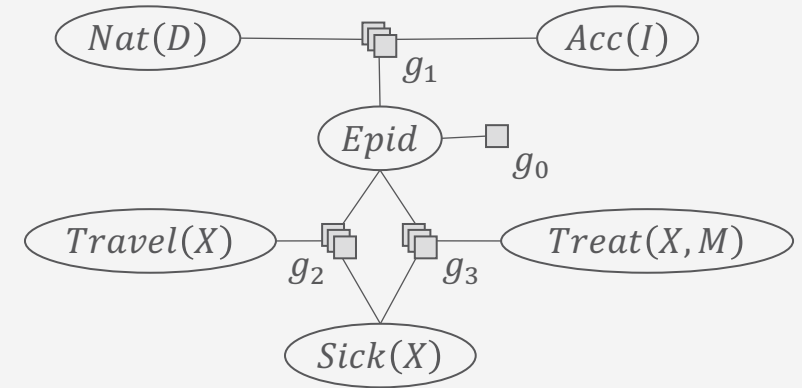
In the lifted case, domain size n no longer occurs in an exponent whereas it does in the propositional case thanks to count conversion

$$\begin{array}{c}
 \phi_1(E, \text{Nat}(D), \text{Acc}(I)) \xrightarrow{\#} \phi_1^{\#}(E, \text{Nat}(D), \#_I[\text{Acc}(I)]) \xrightarrow{\Sigma} \phi_1(E, \#_I[\text{Acc}(I)]) \\
 \hline
 \left. \begin{array}{c} \phi_1(E, \text{Nat}(d_1), \text{Acc}(i_1)) \\ \vdots \\ \phi_1(E, \text{Nat}(d_1), \text{Acc}(i_n)) \end{array} \right\} \xrightarrow{\cdot} \phi_1^1(E, \text{Nat}(d_1), \text{Acc}(i_1), \dots, \text{Acc}(i_n)) \xrightarrow{\Sigma} \phi_1^1(E, \text{Acc}(i_1), \dots, \text{Acc}(i_n)) \\
 \left. \begin{array}{c} \vdots \\ \phi_1(E, \text{Nat}(d_m), \text{Acc}(i_n)) \\ \vdots \\ \phi_1(E, \text{Nat}(d_m), \text{Acc}(i_n)) \end{array} \right\} \xrightarrow{\cdot} \phi_1^1(E, \text{Nat}(d_m), \text{Acc}(i_1), \dots, \text{Acc}(i_n)) \xrightarrow{\Sigma} \phi_1^1(E, \text{Acc}(i_1), \dots, \text{Acc}(i_n))
 \end{array}$$



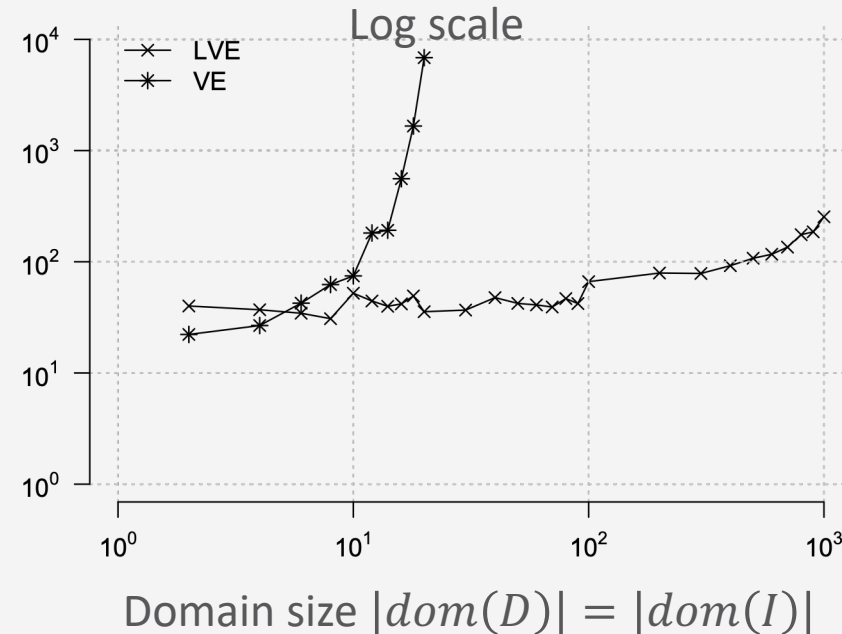
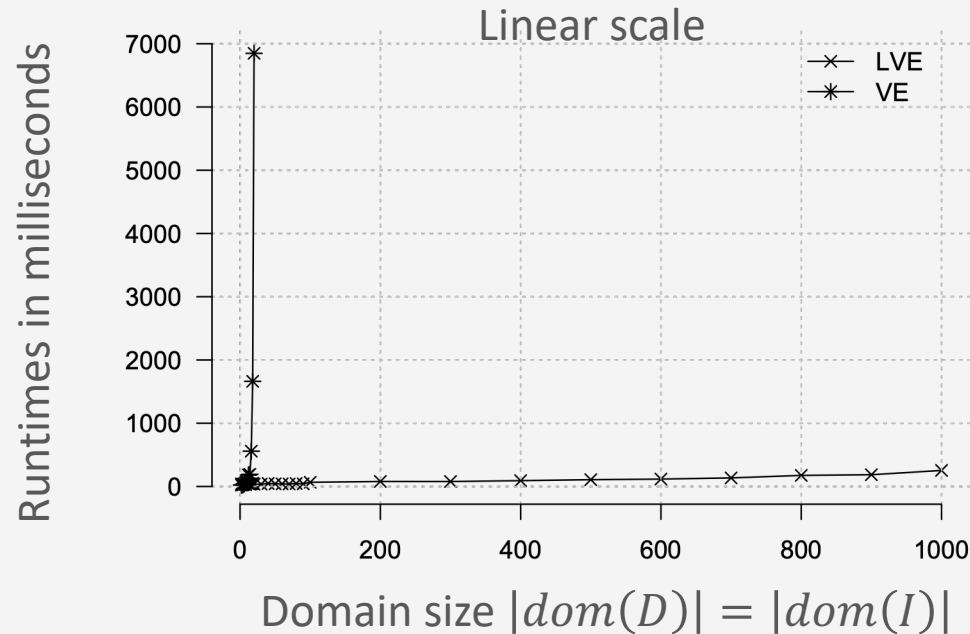
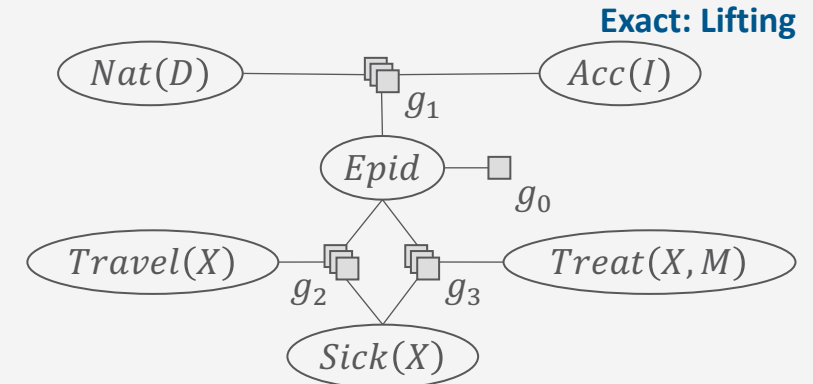
Runtimes: Increasing Domain Sizes

- Running example model with all domain sizes 2, except $|\text{dom}(X)| \in \{2, 4, \dots, 20, 30, \dots, 100, 200, \dots, 1000\}$
- Query: $P(\text{Travel}(x_1))$



Runtimes: Increasing Counted Domain Size

- One count conversion, i.e., $w_{\#} = 1$,
- $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}})$ vs. $O(n_{gr(T)} \cdot r^{n+c})$
- With domain size of D and I (in g_1) increasing



Tractability

- A query answering problem is **tractable**
 - if it is solved by an efficient algorithm, running in time **polynomial in the number of random variables**
- Assume that the number of random variables is *characterised by domain size n* and
 - In LVE, n does not occur in the exponent: $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$
 - Solving a query answering problem is tractable under liftability
 - Runtime still exponential in other terms ($w_g, w_{\#}, r_{\#}$)
- More general results by
Mathias Niepert and Guy Van den Broeck. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *AAAI-14 Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.
 - Tractability through Exchangeability

Completeness

- Class of models $\mathcal{M}$
 - Set of all possible models given some *model characteristic*
- An algorithm is **complete** for a class of models $\mathcal{M}$ iff
 - **No groundings necessary in all models of $\mathcal{M}$**
 - All models allow for a liftable FO dtree
 - Then, class called **liftable**
- Existing liftable classes
 - $\mathcal{M}^{2lv}$:
 - Two logical variables per parfactor max

$$g(A(X, Y), B(X, Y))$$

$$g(A(X, Y), C(X), C(Y)), X \neq Y$$

$$g(A(X, Y), D(X), E(Y))$$
 - $\mathcal{M}^{1prv}$:
 - One logical variable per PRV (arbitrarily many logical variables per parfactor)

$$g(A(X), B(Y), C(Z))$$
 - Holds for various lifted algorithms
 - E.g., LVE, LJ, FOKC

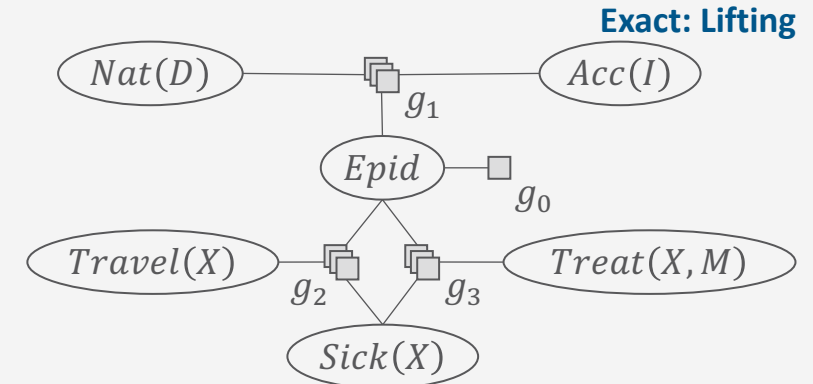
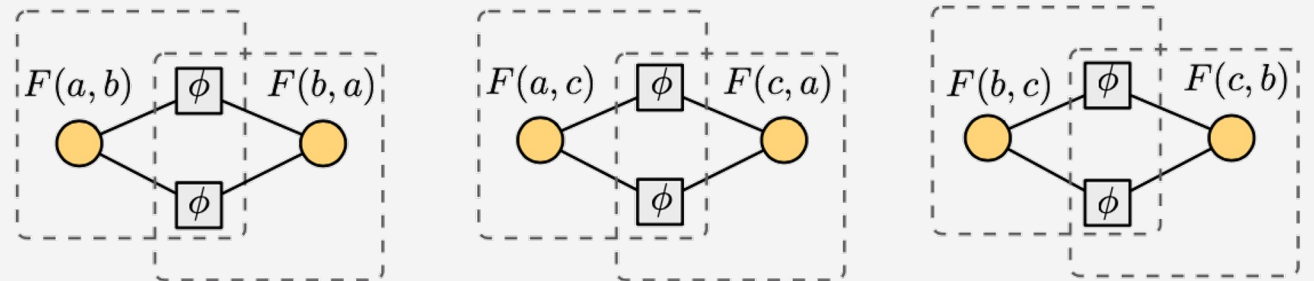
Completeness

- LVE is complete for $\mathcal{M}^{1prv}$ with generalised counting
 - $\mathcal{M}^{1prv}$: One logical variable per PRV
- Proof:
 - Fact: Only PRVs with one logical variable to eliminate
 1. Perform count conversion on all logical variables in the model; possible scenarios in each parfactor
 - A. Logical variable is the only one with a particular domain $\rightarrow$ Standard count conversion applies
 - B. Logical variable occurs in several PRVs without inequality constraints $\rightarrow$ Generalised Counting 1 applies
 - C. Logical variable occurs in several PRVs with inequality constraints $\rightarrow$ After count-converting PRVs of Scenario B, Generalised Counting 3 applies
 - Afterwards: No uncounted logical variables remain
 2. Multiply all parfactors into one large parfactor and merge CRVs (Generalised Counting 2)
 3. Eliminate all merged CRVs (possible since the different CRVs do not overlap after Step 2)
 4. Eliminate all propositional random variables

- Generalised counting by Nima Taghipour et al. (2013)
 1. Count logvars that appear in more than one PRV
 - E.g., $\phi(Q(X), R(X), S(Y), T(Y))$
 $\rightarrow \phi(\#_X[Q(X), R(X)], S(Y), T(Y))$
 2. Merge CRVs with counted logvars of the same domain
 - E.g., $\phi(\#_X[Q(X), R(X)])_{C^X}$ and $\phi(\#_Y[Q(Y), R(Y)])_{C^Y}$ with
 $gr(X)_{C^X} = gr(Y)_{C^Y}$
 $\rightarrow \phi(\#_X[Q(X), R(X)])_C$
 3. Merge-count a PRV and a CRV with an inequality constraint
 - E.g., $\phi(\#_X[Q(X)], R(Y))_C$ with C encoding $X \neq Y$
 $\rightarrow \phi(\#_X[Q(X), R(X)])_C$

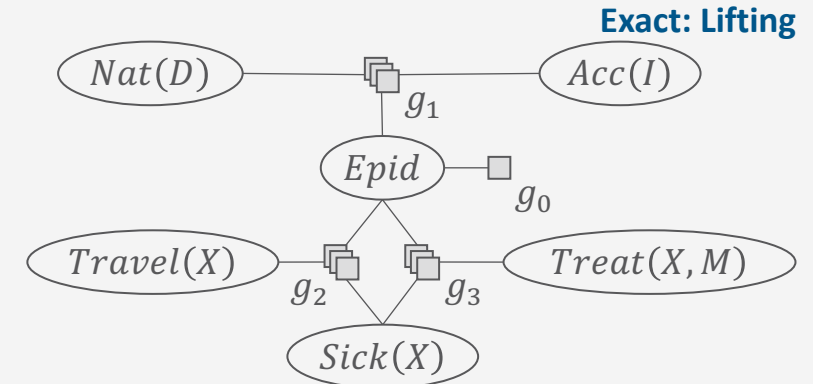
Completeness

- LVE is complete for $\mathcal{M}^{2lv}$
 - $\mathcal{M}^{2lv}$: Maximum of two logical variables per parfactor
- Requires another operator: **Group Inversion**
 - For the case $\phi(F(X, Y), F(Y, X))|_C$, C encodes $X \neq Y$
 - Cannot sum out $F(X, Y)$ independently of $F(Y, X)$ as they refer to same grounded random variables
 - Sums out PRVs $\{A_1, \dots, A_k\}$ from $\phi(\mathcal{A})|_C$ at once where
 - $lv(A_1) = \dots = lv(A_k) = lv(\mathcal{A})$
 - C encodes $X_i \neq X_j$ for each pair of logical variables X_i, X_j , $\text{dom}(X_i) = \text{dom}(X_j)$



Completeness

- LVE is complete for $\mathcal{M}^{2lv}$
 - $\mathcal{M}^{2lv}$: Maximum of two logical variables per parfactor
- Proof idea:
 - Fact 1: Each parfactor has two logical variables X, Y at most
 - Fact 2: Once PRVs with two logical variables are eliminated, model is in $\mathcal{M}^{1prv}$
 1. Multiply all parfactors together that share PRVs with two logical variables
 - Preserves the number of logical variables per parfactor, namely, two
 2. Eliminate each PRV with two logical variables in each parfactor; possible scenarios
 - A. Only PRVs with two logical variables and no inequality constraint → Eliminate using summing out
 - B. PRVs with two logical variables with an inequality constraint → Eliminate using group inversion
 - Afterwards: Only PRVs with one logical variable and propositional random variables remain (Fact 2)
 - 3. Count logical variables in all parfactors, multiply the parfactors and merge CRVs, eliminate CRVs and propositional random variables (compare proof for completeness of $\mathcal{M}^{1prv}$)



Completeness

- Models with other constellations may be computed without groundings but not all possible models
 - E.g., for lifted variable elimination, models with three logical variables

$$g(A(X, Y, Z), B(X, Y), C(X)) \rightarrow \textit{liftable}$$

$$g(F(X, Y), F(Y, Z), K(X, Z)) \rightarrow \textit{not liftable}$$

→ Not complete for class $\mathcal{M}^{3lv}$, i.e., models with three logical variables per parfactor

- Completeness results assume a **liftable class of queries $\mathcal{Q}$** and a **liftable class of evidence $\mathcal{E}$**

Interim Summary

- (FO) dtrees
 - Cutset, context, cluster $\rightarrow$ (lifted) tree width
 - Lifiable models
- Complexity
 - No longer exponential in domain sizes given liftable model $\rightarrow$ tractability
- Completeness
 - No groundings for
 - Models with two logical variables per parfactor
 - Models with one-logical variable PRVs and propositional random variables

Lifting Queries, Junction Trees, Most Probable Assignments

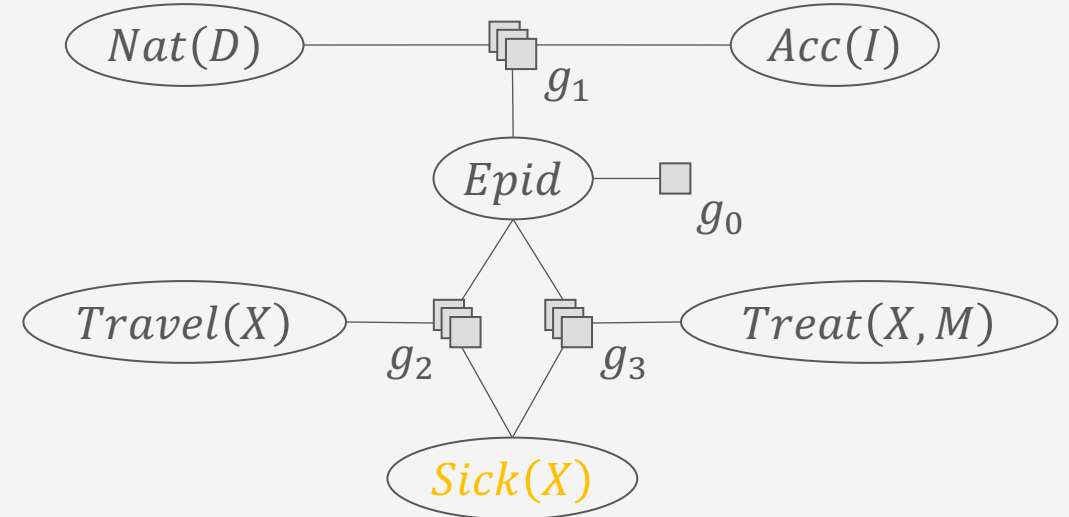
Parameterised queries

Lifted Junction Tree Algorithm (LJT)

Lifted MPE / MAP

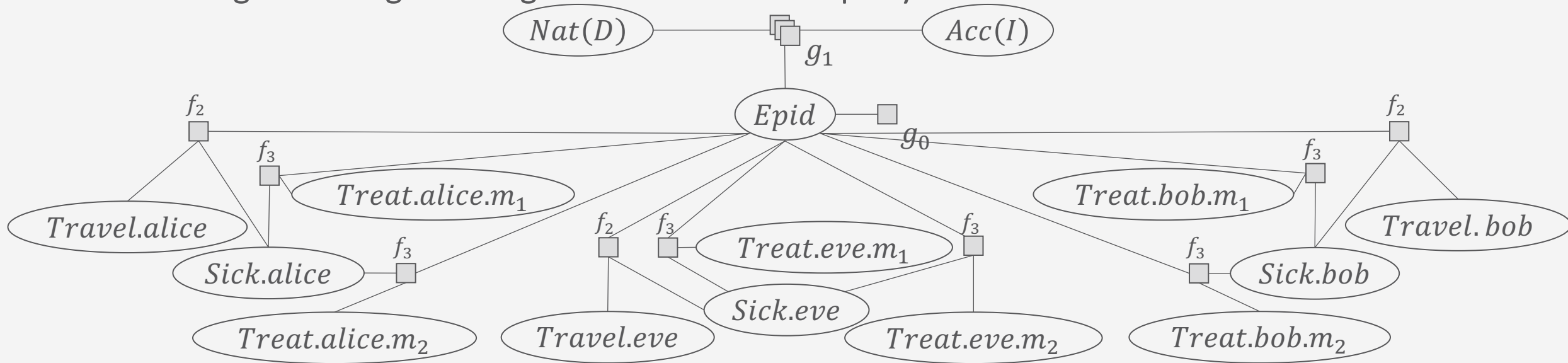
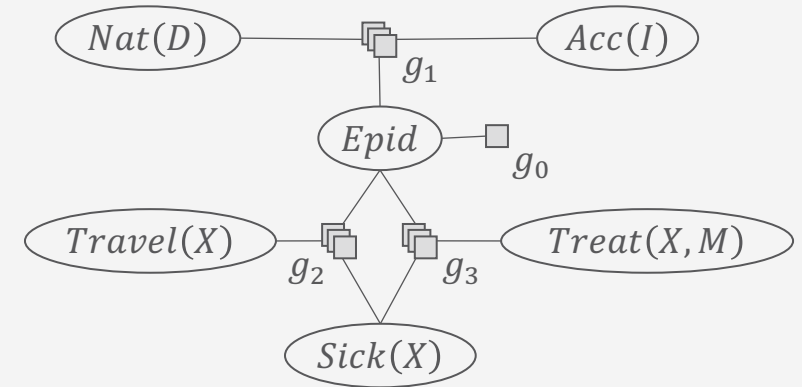
What About Parameterised Queries?

- Allow logical variables in query terms: $P(A_{|C}|T)$
 - Represents a conjunctive query $P(\text{gr}(A_{|C})|T)$
- E.g., $P(\text{Sick}(X)_{|T})$ for $P(\text{Sick}(\text{alice}), \text{Sick}(\text{eve}), \text{Sick}(\text{bob}))$



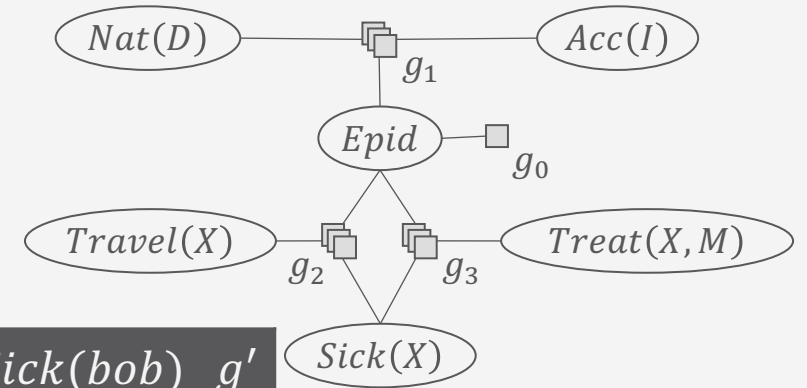
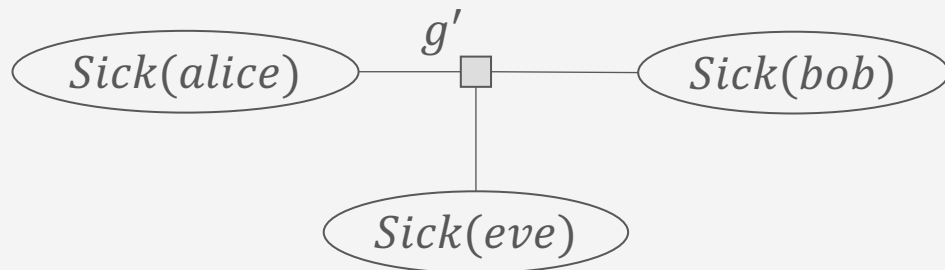
Indistinguishable Query Terms

- Indistinguishable instances in query:
 - $P(\text{Sick}(\text{alice}), \text{Sick}(\text{eve}), \text{Sick}(\text{bob}))$
- Standard LVE:
 - Shattering leads to groundings w.r.t. constants in query



... And Their Effect

- Query: $P(\text{Sick}(\text{alice}), \text{Sick}(\text{eve}), \text{Sick}(\text{bob}))$
- After shattering, eliminate all non-query terms
 - Identical computations in eliminations
 - Large intermediate results
 - Symmetries in result
 - Encode with CRV



$\text{Sick}(\text{alice})$	$\text{Sick}(\text{eve})$	$\text{Sick}(\text{bob})$	g'
false	false	false	1
false	false	true	2
false	true	false	2
false	true	true	3
true	false	false	2
true	false	true	3
true	true	false	3
true	true	true	4

$\#_X[\text{Sick}(X)]$	g
[0,3]	1
[1,2]	2
[2,1]	3
[3,0]	4

LVE for Parameterised Queries

- To avoid grounding, take PRV representation of query terms and apply LVE as before
 - Shatter model on constraint of query terms $\rightarrow$ maximum of two groups per parfactor and logical variable
 - Eliminate all non-query terms
 - If logical variable prevents application of operator $\rightarrow$ count or ground logical variable from query
 - After all non-query terms eliminated, count or ground remaining logical variables to make logical variables explicit
 - Otherwise only in representative form but not a joint over all groundings
- At the end, the result contains the logical variables counted (or grounded)
 - If counting the logical variables of $A_{|c}$ is not possible, then LVE needs to ground them to ensure a distribution over $A_{|c}$

Normalisation

- Histogram h may encode multiple assignments $\{\mathbf{a}_i\}_{i=1}^{\text{Mul}(h)}$
 - $\text{Mul}(h)$ assignments have the potential $\phi(h)$
 - Incorporate into normalisation (just like we needed to incorporate that into SUM—OUT)
- To get the probability of one assignment $\mathbf{a}$ behind histogram h in parfactor $\phi(\#_X[R(X)])$:

$$p_{\mathbf{a}|h} = \frac{\phi(h)}{\sum_{h \in \text{ran}(\#_X[R(X)])} \text{Mul}(h) \cdot \phi(h)}$$

- Probability of $\text{Mul}(h)$ assignments

$$p_h = \text{Mul}(h) \cdot p_{\mathbf{a}|h}$$

- Distribution:

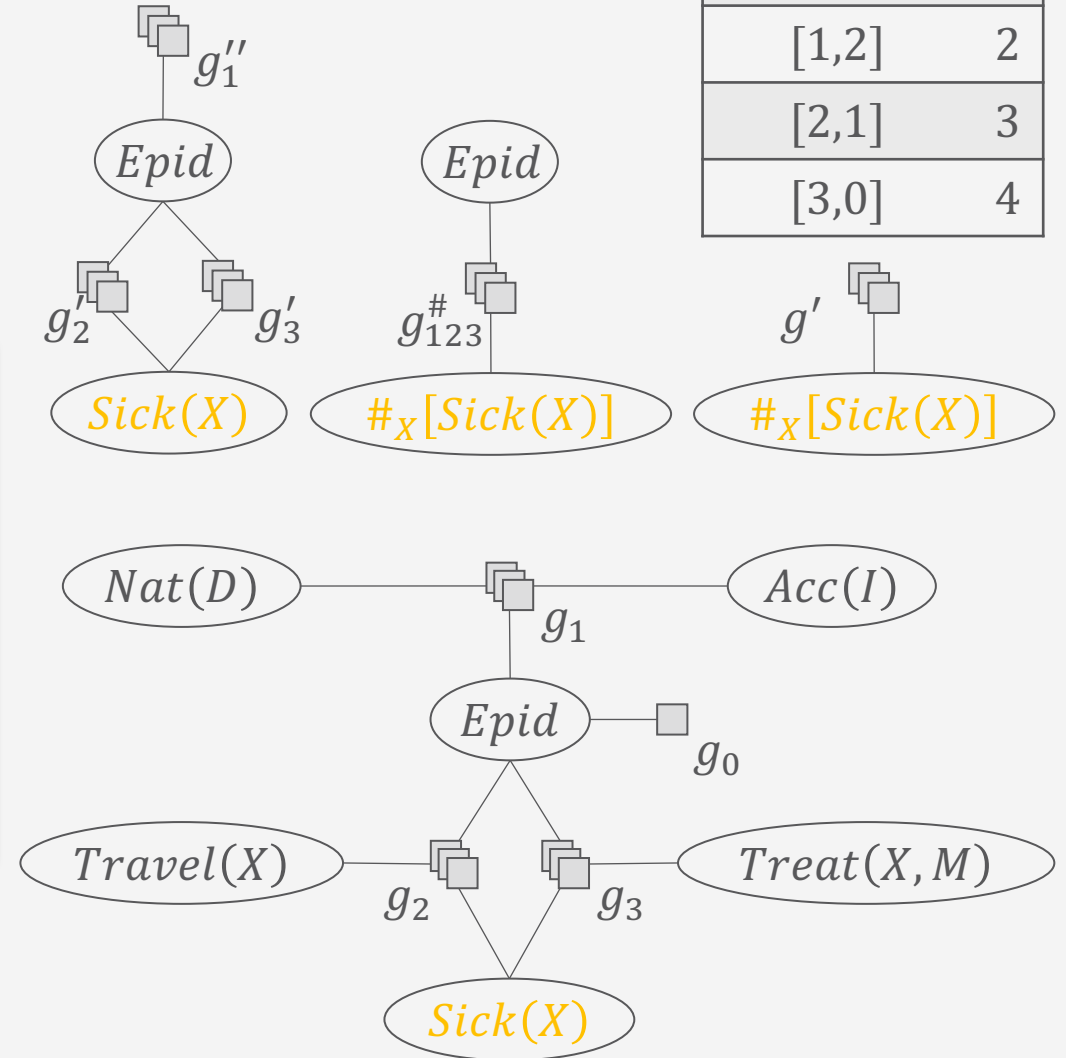
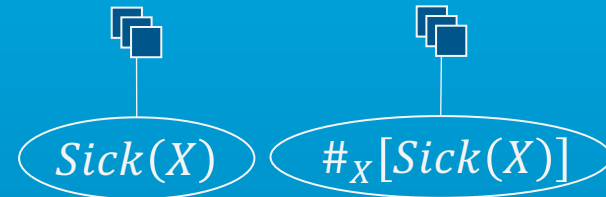
$$\sum_{h \in \text{ran}(\#_X[R(X)])} p_h = \sum_{h \in \text{ran}(\#_X[R(X)])} \text{Mul}(h) \cdot p_{\mathbf{a}|h} = 1$$

$\#_X[\text{Sick}(X)]$	g
[0,3]	1
[1,2]	2
[2,1]	3
[3,0]	4

Example

- Query: $P(Sick(X))$
- No shattering necessary with T constraints
- Elimination:
 - Eliminate $Treat(X, M)$
 - Eliminate $Travel(X)$
 - Count-convert $Acc(I)$
 - Eliminate $Nat(D)$
 - Eliminate $\#_I[Acc(I)]$
 - Eliminate $Epid$
 - Multiply parfactors (fulfil precondition 1)
 - Count-convert X (fulfil precondition 2)
 - Sum out $Epid$

Here, count conversion as part of elimination, but if logical variables remaining after elimination, count conversions afterwards (trivially possible):



$\#_X[Sick(X)]$	ϕ'
[0,3]	1
[1,2]	2
[2,1]	3
[3,0]	4

Example

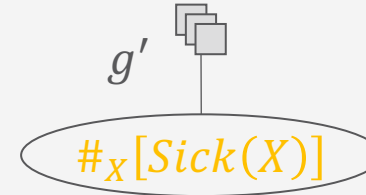
- Query: $P(\text{Sick}(X))$
- Elimination: Finished
- Normalisation:

$$p_{a|h} = \frac{\phi(h)}{\sum_{h \in \text{ran}(\#_X[R(X)])} \text{Mul}(h) \cdot \phi(h)} = \frac{\phi(h)}{1 \cdot 1 + 3 \cdot 2 + 3 \cdot 3 + 1 \cdot 4}$$

- Probability distribution:

$$\sum_{h \in \text{ran}(\#_X[R(X)])} \text{Mul}(h) \cdot p_{a|h}$$

$$= 1 \cdot \frac{1}{20} + 3 \cdot \frac{2}{20} + 3 \cdot \frac{3}{20} + 1 \cdot \frac{4}{20} = \frac{1 + 6 + 9 + 4}{20} = \frac{20}{20} = 1$$



$\#_X[\text{Sick}(X)]$	ϕ'
[0,3]	1
[1,2]	2
[2,1]	3
[3,0]	4

	single assignment	all assignments
	$\rightarrow 1/20$	$\rightarrow 1/20$
	$\rightarrow 2/20$	$\rightarrow 6/20$
	$\rightarrow 3/20$	$\rightarrow 9/20$
	$\rightarrow 4/20$	$\rightarrow 4/20$

Exact: Lifting

Splits Affecting Query Logical Variables

- Logical variables X in query terms may be split (or grounded) in result
 - If splits of the model affect the query logical variables
- Prominent case: evidence; three cases given query PRV $R(X)_{|C}$, evidence PRV $E(Y)_{|C_e}$
 1. Overlap of instances (i.e., $R(X) = E(Y)$): $\text{gr}(R(X)_{|C}) \cap \text{gr}(E(X)_{|C_e}) \neq \emptyset$
 - Split C on the overlap with C_e , i.e., $C /_X C_e \rightarrow$ instances of C_e will be absorbed
 - Result has $R(X)_{|C}$ partitioned into $C /_X C_e \setminus C_e$ (absorbed instances: probability 1 of observed value)
 2. Overlap of constants (Z shared logical variables): $\text{gr}(\pi_Z(X_{|C})) \cap \text{gr}(\pi_Z(Y_{|C_e})) \neq \emptyset$
 - Split C on the overlap with C_e , i.e., $C /_Z C_e$
 - Answer has $R(X)_{|C}$ partitioned into $C /_X C_e$ in the result (different evidence applies)
 - No overlap (more of a non-case): $\text{gr}(R(X)_{|C}) \cap \text{gr}(E(Y)_{|C_e}) = \emptyset \wedge$ no shared logvars Z
 - $R(X)_{|C}$ is not partitioned in the result because of evidence (maybe partitioned for other reasons)

Splits Affecting Query Logical Variables: Examples

- Given query $P \left(Sick(X)_{|(X, \{x_i\}_{i=1}^{20})} \right)$:

1. Overlap of instances:

evidence $\phi_e(Sick(X))_{|(X, \{x_i\}_{i=1}^{10})}$

- Result: $\phi(\#_X[Sick(X)])_{|(X, \{x_i\}_{i=1}^{20})}$

2. Overlap of constants:

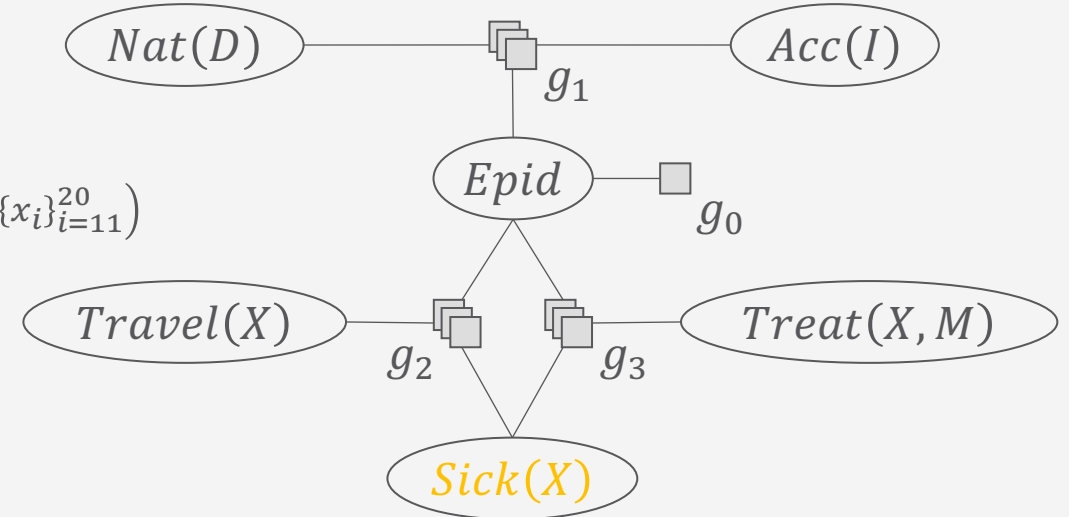
evidence $\phi_e(Travel(X))_{|(X, \{x_i\}_{i=1}^{10})}$

- Result: $\phi(\#_{X'}[Sick(X')], \#_{X''}[Sick(X'')])_{|((X', X''), \{x_i\}_{i=1}^{10} \times \{x_i\}_{i=1}^{20})}$

3. No overlap:

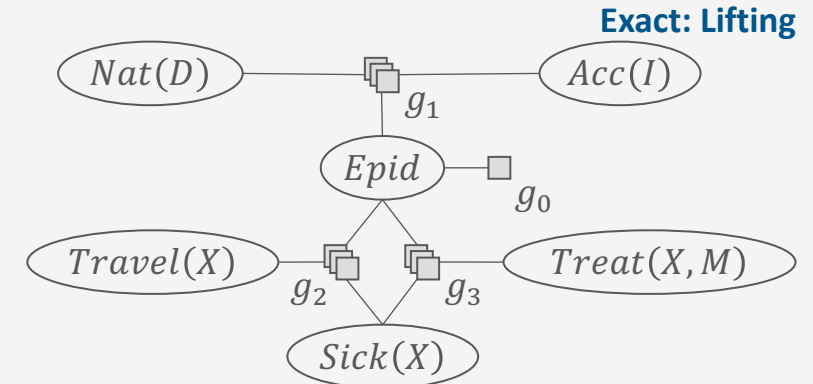
evidence $\phi_e(Nat(D))_{|(D, \{d_i\}_{i=1}^2)}$

- Result: $\phi(\#_X[Sick(X)])_{|(X, \{x_i\}_{i=1}^{20})}$



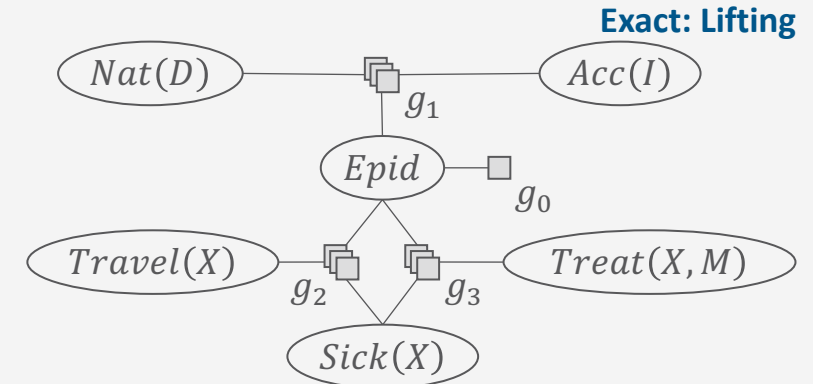
Completeness Beyond Models: Queries

- Queries Q :
 - Class of one ground query term Q *liftable*
 - As argued on earlier slide, one query term does not influence complexity and cannot cause groundings
 - Class of sets of ground query terms Q *not liftable*
 - Proof by counter example
 - $P(\text{Sick}(\text{eve}), \text{Travel}(\text{alice}), \text{Treat}(\text{bob}, m_1))$ grounds X
 - LVE no longer polynomial in domain size
 - Class of query terms Q containing at most one constant for each logical variable in $lv(G)$ *liftable*
 - Argument: Splits do not lead to a set of parfactors whose size depends on the domain size of logical variables
 - Examples: $P(\text{Travel}(\text{eve}))$, $P(\text{Travel}(\text{eve}), \text{Sick}(\text{eve}))$, $P(\text{Travel}(\text{eve}), \text{Nat}(\text{chem}))$



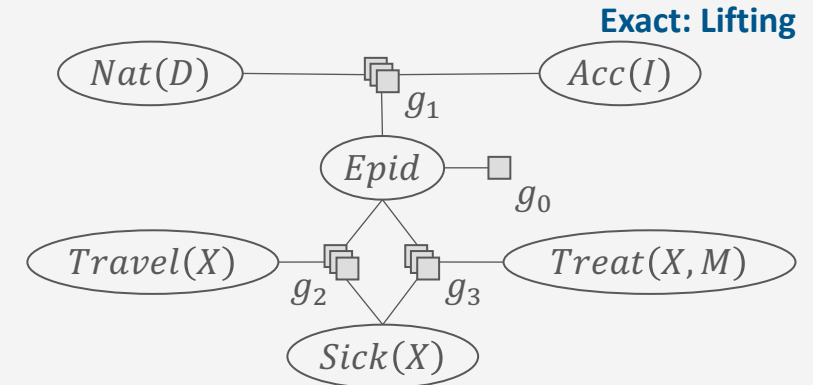
Completeness Beyond Models: Queries

- Queries Q :
 - Class of all parameterised queries *not liftable*
 - Proof by counter example, using constraints or logical variables :
 - $P(Sick(X'), Travel(X''))_{|((X', X''), \{alice, eve\} \times \{eve, bob\})}$
 - Query $P(B(X, Y))$ in model $g(A(X), B(X, Y), C(Y))$
 - Parameterised query terms with only one parameter per term and one subset of constants per domain *liftable*
 - Proof along the lines of proving completeness for $\mathcal{M}^{1prv}$
 - Example: $P(Sick(X), Travel(X))_{|T}$
- Corollary
 - CRVs compactly represent the result of liftable queries



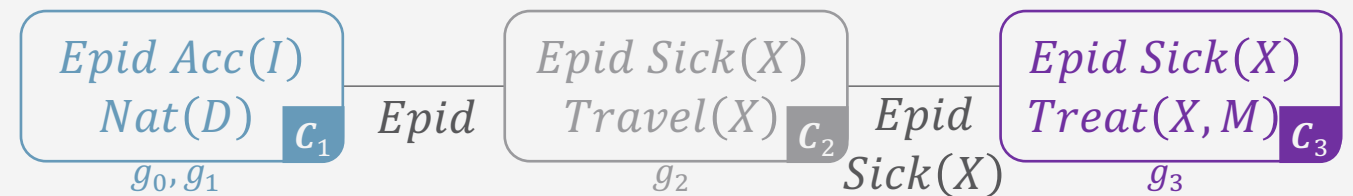
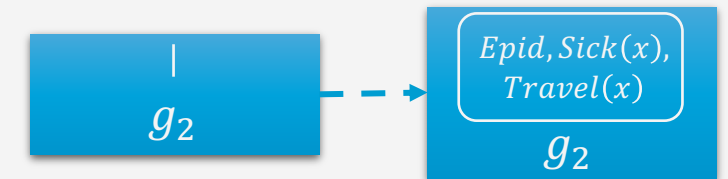
Completeness Beyond Models: Evidence

- Evidence $\mathcal{E}$:
 - *Liftable* class: Evidence on propositional random variables
 - Example: $Epid = \text{true}$
 - *Liftable* class: Evidence on instances of PRVs with one logical variable
 - Example: $Sick(X) = \text{true}, \text{dom}(X) = \{alice, eve, bob\}$
 - General evidence on PRVs with two logical variables *not liftable* in all cases
 - Lifted calculations possible for some cases but not for all
 - Proof by reduction to #2SAT problem



Lifted Junction Tree Algorithm

- Repeated structures of dtree also occur in jtree, identical messages sent
→ lift jtree to first-order version
- First-order junction tree (FO jtree)
 - Nodes = Parclusters = sets of PRVs with separators between them
 - Jtree properties also apply to FO jtrees
 - Local models of parfactors
 - Construct using clusters of FO dtree
 - Keep constraints from DPG nodes
 - Re-substitute representative constants with logical variables again



Lifted Junction Tree Algorithm

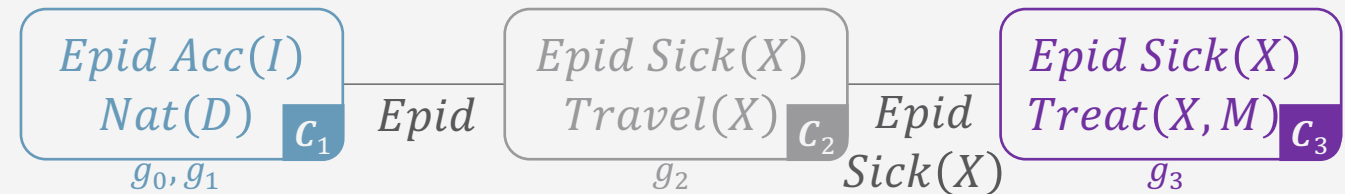
- Lifted Junction Tree Algorithm
 - Inputs
 - Model G
 - Query variables $\{Q_i\}_{i=1}^m$
 - Evidence parfactors $\{g_e\}_{e=1}^E$
 - Procedure (as before)
 1. Construct FO jtree J
 2. Enter evidence
 3. Pass messages
 4. Answer queries
 - Use LVE operators for steps 2-4
- Analogous complexity results

Preconditions:

1. $\forall B \in rv(G \setminus \{g\}) : gr(B|_C) \cap gr(A_i|(x, c_x)) = \emptyset$
2. $\forall X \in \{X \mid |\pi_X(C_x)| > 1\} : X \in lv(A_i)$
3. $X^{excl} = lv(A_i) \setminus (X \setminus lv(A_i))$ count-normalised w.r.t. X^{con}

Exact: Lifting

- Algorithm-induced groundings possible
 - Even in liftable models!
 - For message calculation, non-separator PRVs are eliminated
 - Have to fulfil sum-out preconditions
 - Precondition 2 may **not** be fulfilled
 - Example: Add logical variable E to PRVs $Epid, Sick(X), Treat(X, M)$
- Merge nodes to avoid grounding



Lifted MPE and MAP

- As before, if random variables are indistinguishable, then the same assignment has the maximum value → Assign in groups
 - Formally, computations are identical → can reuse result

$$\begin{aligned}\text{MPE}_F(\mathbf{e}) &= \arg \max_{\mathbf{r} \in \text{ran}(\mathbf{R})} P(\mathbf{r} \mid \mathbf{e}) = \arg \max_{\mathbf{r} \in \text{ran}(\mathbf{R})} P(\mathbf{r}, \mathbf{e}) \\ \text{MAP}_F(\mathbf{U} \mid \mathbf{e}) &= \arg \max_{\mathbf{u} \in \text{ran}(\mathbf{U})} \sum_{\mathbf{v} \in \text{ran}(\mathbf{V})} P(\mathbf{u}, \mathbf{v} \mid \mathbf{e}) = \arg \max_{\mathbf{u} \in \text{ran}(\mathbf{U})} \sum_{\mathbf{v} \in \text{ran}(\mathbf{V})} P(\mathbf{u}, \mathbf{v}, \mathbf{e})\end{aligned}$$

- Replace lifted sum-out operator with a lifted max-out operator

Interim Summary

- Parameterised queries
 - PRVs as query variables allow for avoiding query-induced groundings
 - Count conversion of query logical variables to yield compact distribution over groundings
 - Lifiable queries / evidence
- LJT for multi-query answering
 - FO jtree: first-order version of jtree, built from FO dtree
 - Procedure same as before (construction, evidence entering, message passing, query answering)
 - Algorithm-induced groundings possible → avoid by merging nodes
 - Complexity results analogous to LVE / JT
- Lifted MPE + MAP
 - Most probable assignment for groups of indistinguishable random variables
 - Lifted max-out operator

First-order Knowledge Compilation

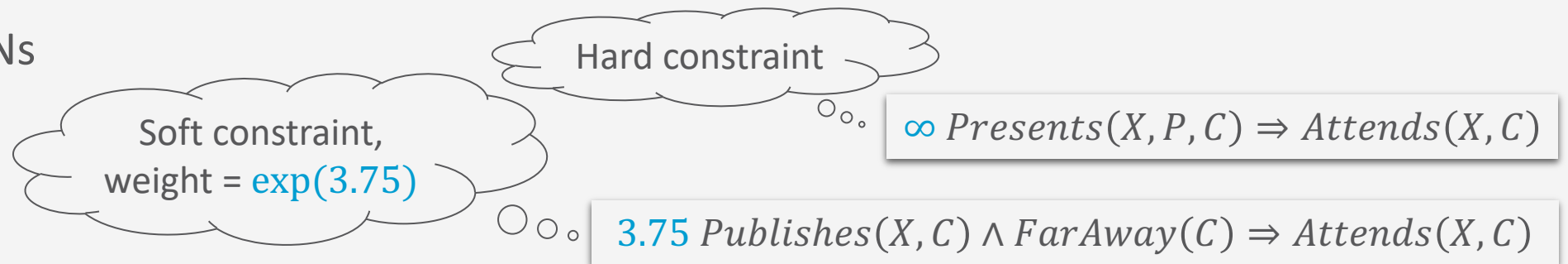
Lifting knowledge compilation for MLNs

$$10 \text{ } \mathit{Presents}(X, P, C) \Rightarrow \mathit{Attends}(X, C)$$

$$3.75 \text{ } \mathit{Publishes}(X, C) \wedge \mathit{FarAway}(C) \Rightarrow \mathit{Attends}(X, C)$$

Remember: Markov Logic Networks (MLNs)

- Again: First-order view on probabilistic modelling
- Use logical formulas to specify potential functions
 - Weights for each formula induce a full joint again
 - Lower weight if constraint does not hold all the time
 - ∞ weight for hard constraints, in practice either a high weight or only worlds considered that satisfy hard constraints
- Next slides
 - Syntax of MLNs
 - Semantics of MLNs



Local Symmetries and Structure

- Consider potential function as given by the table on the right
 $\phi(\text{Travel}(X), \text{Epid}, \text{Sick}(X))$
- Only two weighted formulas (w, ψ) necessary
 - $(\ln 2, \neg \text{travel}(X) \vee \neg \text{epid} \vee \neg \text{sick}(X))$
 - $(\ln 7, \text{travel}(X) \wedge \text{epid} \wedge \text{sick}(X))$
- If potential of 1 instead of 2, would reduce to
 - $(\ln 7, \text{travel}(X) \wedge \text{epid} \wedge \text{sick}(X))$
 - Assignments that do not make the formula true automatically get weight of $0 = \ln 1$
- If external knowledge existing, provide FOL formulas directly
 - E.g., $(\ln 2, \text{epid} \wedge \text{sick}(X) \Rightarrow \neg \text{travel}(X))$

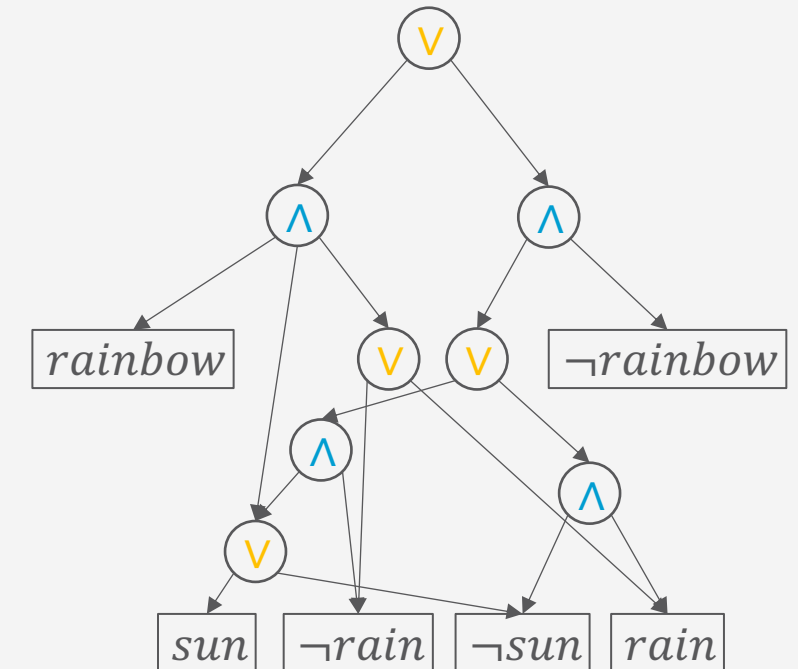
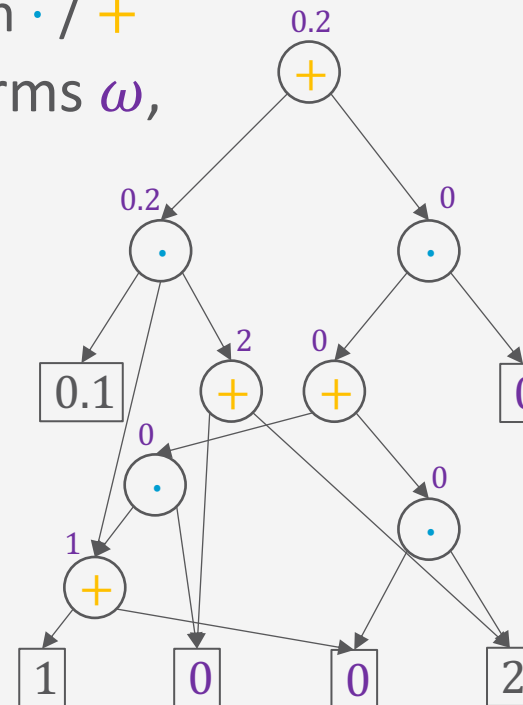
Use for efficient inference

<i>Travel(X)</i>	<i>Epid</i>	<i>Sick(X)</i>	ϕ
false	false	false	2
false	false	true	2
false	true	false	2
false	true	true	2
true	false	false	2
true	false	true	2
true	true	false	2
true	true	true	7

Remember: Circuit for Weighted Model Counting – Knowledge Compilation

- Input: Propositional sentences, weight function for literals
- Build smooth dDNNF circuit ← Reuse for different queries
 - Replace conjunctions / disjunctions with $\cdot$ / $+$
- For probabilities of worlds or query terms ω , condition on truth values
 1. Compute $WMC(\varphi, weight)$
 2. Compute $WMC(\varphi \wedge \omega, weight)$
 3. Divide the two counts

$$P(\omega = \{sun, rain, rainbow\}) = \frac{WMC(\varphi \wedge \omega, weight)}{WMC(\varphi, weight)} = \frac{0.2}{525.4} = 0.00038$$



Propositional → First-order

- If input theory is in FOL-DC ((function-free) first-order logic with domain constraints), one could ground the theory given domains and build a circuit for the grounded theory
 - FOL-DS includes intensional conjunctions and disjunctions ($\forall, \exists$)
 - Leads to repeated structures in circuit
- Combine repeated structures using new inner node types for intensional conjunctions and disjunctions ($\forall, \exists$)
- We are not going into every detail of FOKC;
 - For complete description, analysis, and discussion, see the PhD thesis by Guy Van den Broeck

Weighted First-order Model Counting

- Define a weighted first-order model counting problem using a weighted first-order model count (**WFOMC**)

$$WFOMC(\Delta, w_T, w_F) = \sum_{\substack{\omega = \omega_T \cup \omega_F \\ \omega \in \Omega_\Delta}} \prod_{l \in \omega_T} w_T(pred(l)) \prod_{l \in \omega_F} w_F(pred(l))$$

- Δ a theory in FOL-DC
- w_T a weight function for predicates being positive
- w_F a weight function for predicates being negative
- Ω_Δ the set of worlds (i.e., models in logics) of Δ
- $pred(l)$ a function mapping a literal l to its predicate
- Query can be answered by computing

$$P(q_i|e) = \frac{WFOMC(\Delta \wedge e \wedge q_i, w_T, w_F)}{WFOMC(\Delta \wedge e, w_T, w_F)}$$

Example

- Theory: one sentence
 $\forall X \in \text{People} : \text{smokes}(X) \Rightarrow \text{cancer}(X)$
- People = $\{x_1, x_2\}$
- Weight functions
 - $w_T(\text{smokes}(X)) = 3$
 - $w_F(\neg \text{smokes}(X)) = 1$
 - $w_T(\text{cancer}(X)) = 6$
 - $w_F(\neg \text{cancer}(X)) = 2$
- Model count: 9

$$WFOMC(\Delta, w_T, w_F)$$

$$= \sum_{\substack{\omega = \omega_T \cup \omega_F \\ \omega \in \Omega_\Delta}} \prod_{l \in \omega_T} w_T(\text{pred}(l)) \prod_{l \in \omega_F} w_F(\text{pred}(l))$$

$s(x_1)$	$c(x_1)$	$s(x_2)$	$c(x_2)$	Weight	
0	0	0	0	$1 \cdot 2 \cdot 1 \cdot 2$	4
0	0	0	1	$1 \cdot 2 \cdot 1 \cdot 6$	12
~~0~~	~~0~~	~~1~~	~~0~~	~~$1 \cdot 2 \cdot 3 \cdot 2$~~	12
0	0	1	1	$1 \cdot 2 \cdot 3 \cdot 6$	36
0	1	0	0	$1 \cdot 6 \cdot 1 \cdot 2$	12
0	1	0	1	$1 \cdot 6 \cdot 1 \cdot 6$	36
~~0~~	~~1~~	~~1~~	~~0~~	~~$1 \cdot 6 \cdot 3 \cdot 2$~~	36
0	1	1	1	$1 \cdot 6 \cdot 3 \cdot 6$	108
~~1~~	~~0~~	~~0~~	~~0~~	~~$3 \cdot 2 \cdot 1 \cdot 2$~~	12
~~1~~	~~0~~	~~0~~	~~1~~	~~$3 \cdot 2 \cdot 1 \cdot 6$~~	36
~~1~~	~~0~~	~~1~~	~~0~~	~~$3 \cdot 2 \cdot 3 \cdot 2$~~	36
~~1~~	~~0~~	~~1~~	~~1~~	~~$3 \cdot 2 \cdot 3 \cdot 6$~~	108
1	1	0	0	$3 \cdot 6 \cdot 1 \cdot 2$	36
1	1	0	1	$3 \cdot 6 \cdot 1 \cdot 6$	108
~~1~~	~~1~~	~~1~~	~~0~~	~~$3 \cdot 6 \cdot 3 \cdot 2$~~	108
1	1	1	1	$3 \cdot 6 \cdot 3 \cdot 6$	324
				+	676

FOKC

Example

- Theory: one sentence
 $\forall X \in \text{People} : \text{smokes}(X) \Rightarrow \text{cancer}(X)$
- People = $\{x_1, x_2\}$
- Weight functions
 - $w_T(\text{smokes}(X)) = 3$
 - $w_F(\neg \text{smokes}(X)) = 1$
 - $w_T(\text{cancer}(X)) = 6$
 - $w_F(\neg \text{cancer}(X)) = 2$
- Model count: 9

$$P(s(x_1)) = \frac{WFOMC(\Delta \wedge s(x_1), w_T, w_F)}{WFOMC(\Delta, w_T, w_F)}$$

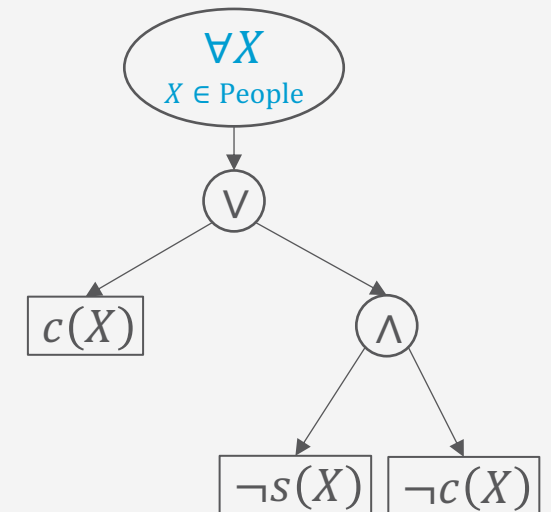
$$= \frac{36 + 108 + 324}{676} = \frac{468}{676}$$

$s(x_1)$	$c(x_1)$	$s(x_2)$	$c(x_2)$	Weight	
0	0	0	0	~~1~~ · ~~2~~ · ~~1~~ · ~~2~~	~~4~~
0	0	0	1	~~1~~ · ~~2~~ · ~~1~~ · 6	12
0	0	1	0	~~1~~ · ~~2~~ · 3 · ~~2~~	12
0	0	1	1	~~1~~ · ~~2~~ · 3 · 6	36
0	1	0	0	~~1~~ · 6 · ~~1~~ · 2	12
0	1	0	1	~~1~~ · 6 · ~~1~~ · 6	36
0	1	1	0	~~1~~ · 6 · 3 · ~~2~~	36
0	1	1	1	~~1~~ · 6 · 3 · 6	108
1	0	0	0	~~3~~ · ~~2~~ · ~~1~~ · ~~2~~	12
1	0	0	1	~~3~~ · ~~2~~ · ~~1~~ · 6	36
1	0	1	0	~~3~~ · ~~2~~ · 3 · ~~2~~	36
1	0	1	1	~~3~~ · ~~2~~ · 3 · 6	108
1	1	0	0	3 · 6 · ~~1~~ · ~~2~~	36
1	1	0	1	3 · 6 · ~~1~~ · 6	108
1	1	1	0	~~3~~ · 6 · 3 · ~~2~~	108
1	1	1	1	3 · 6 · 3 · 6	324
				+	676

FOKC

First-order (FO) Circuits

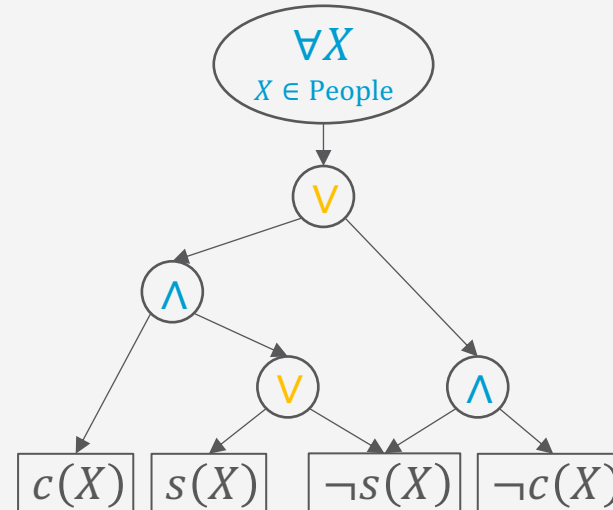
- Assume theory in Skolem normal form + CNF
 - Sequence of intensional conjunctions in CNF
 - E.g., with $s = \text{smokes}$, $c = \text{cancer}$
$$\forall X \in \text{People} : s(X) \Rightarrow c(X)$$
$$\equiv \forall X \in \text{People} : \neg s(X) \vee c(X)$$
- FO circuit (excerpt)
 - Inner nodes:
 - Extensional conjunctions/disjunctions (as before)
 - **Set conjunctions**
 - Leaf nodes
 - Positive and negative predicates, *true*, *false*
 - Full + construction: see PhD thesis by Guy Van den Broeck



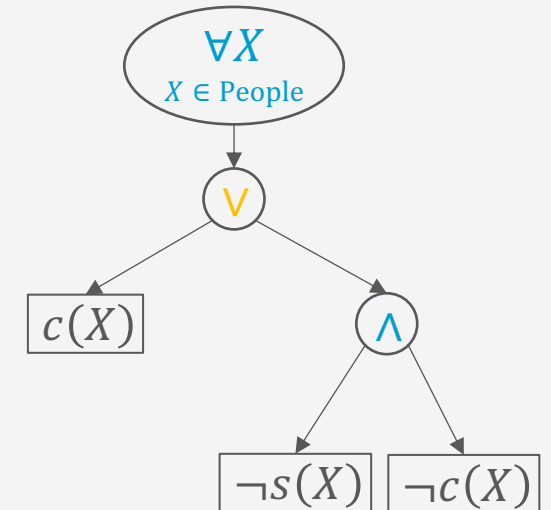
Smooth FO d-DNNF Circuits

- Properties
 - Deterministic disjunctions
 - Only one disjunct (child node) can be true at the same time
 - Decomposable conjunctions
 - Each pair of conjuncts (child nodes) must be independent
 - Smoothness
 - Each disjunct contains the same variables

smooth FO d-DNNF

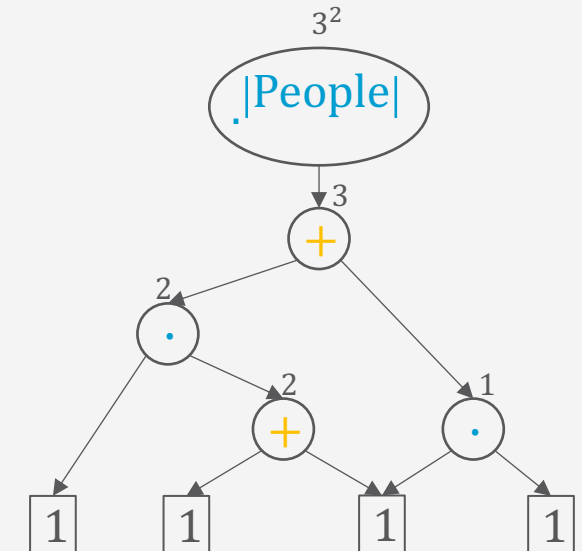
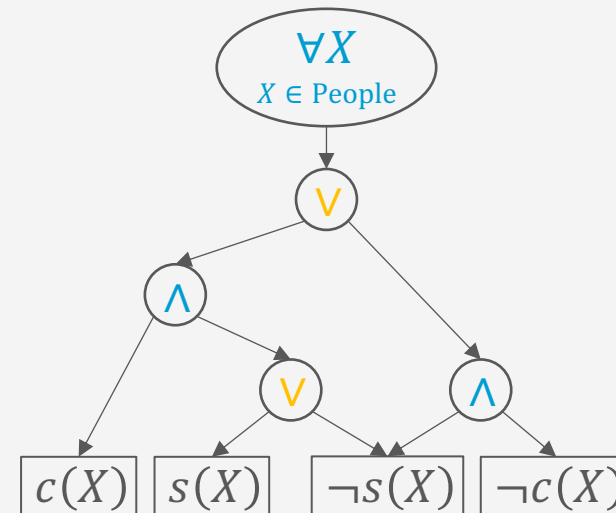


FO d-DNNF



Arithmetic FO d-DNNF Circuits

- Replace
 - Replace $\wedge$ with $\cdot$
 - Replace $\vee$ with $+$
 - Replace $\forall$ with exponentiation for $|\text{Domain}|$
 - Replace leaves with 1's
- E.g., with $|\text{People}| = |\{x_1, x_2\}| = 2$



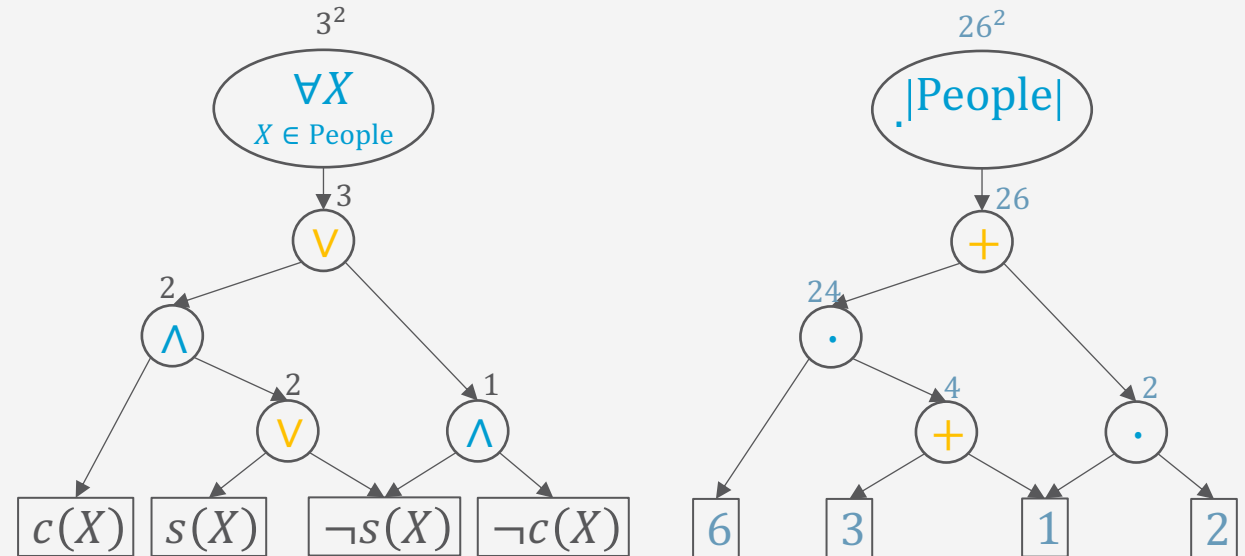
WFOMC Circuits

- Replace
 - Replace $\wedge$ with $\cdot$
 - Replace $\vee$ with $+$
 - Replace $\forall$ with exponentiation for $|\text{Domain}|$
 - Replace leaves with **weights**
 - E.g., with $|\text{People}| = |\{x_1, x_2\}| = 2$

$$WFOMC(\Delta, w_T, w_F)$$

$$= \sum_{\substack{\omega = \omega_T \cup \omega_F \\ \omega \in \Omega_\Delta}} \prod_{l \in \omega_T} w_T(pred(l)) \prod_{l \in \omega_F} w_F(pred(l))$$

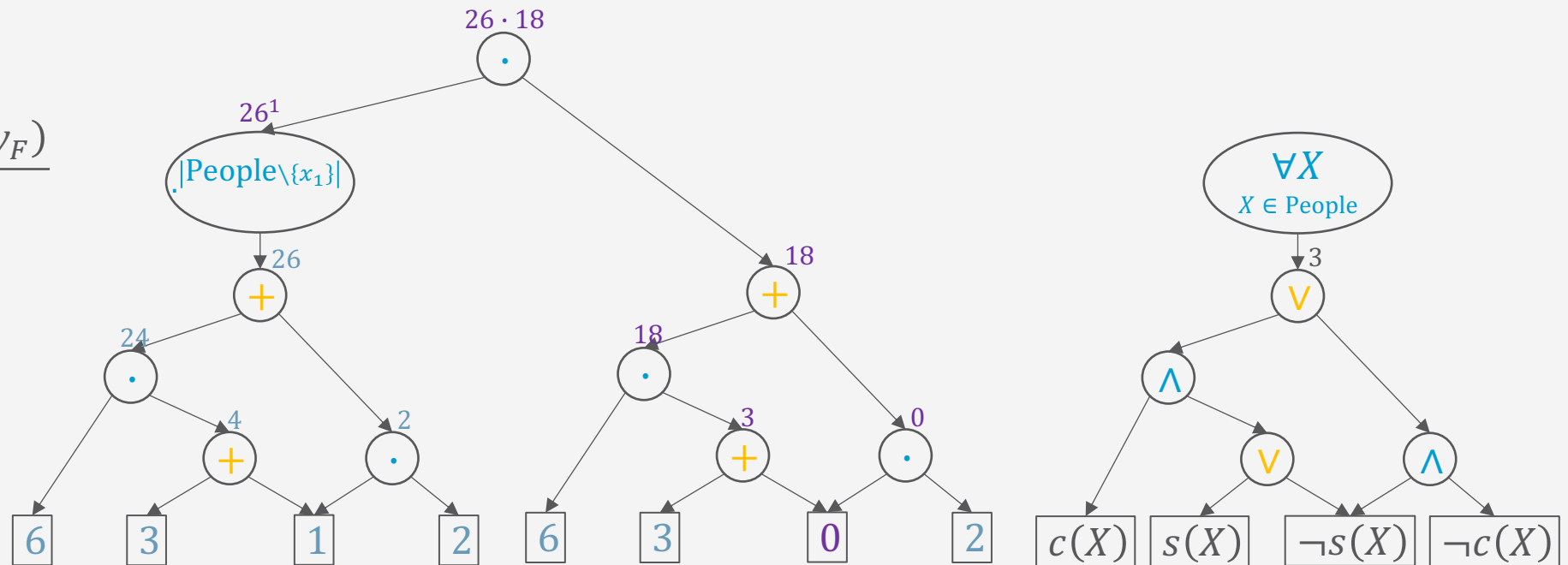
$$\begin{aligned} w_T(smokes(X)) &= 3 \\ w_F(\neg smokes(X)) &= 1 \\ w_T(cancer(X)) &= 6 \\ w_F(\neg cancer(X)) &= 2 \end{aligned}$$



WFOMC Circuits

- Given $P(q_i)$
 - Basically, compile a circuit for $\Delta \wedge q_i$ reusing components from the circuit of Δ
 - E.g., $P(s(x_1))$ with $|\text{People}| = |\{x_1, x_2\}| = 2$

$$\begin{aligned}
 &P(s(x_1)) \\
 &= \frac{WFOMC(\Delta \wedge s(x_1), w_T, w_F)}{WFOMC(\Delta, w_T, w_F)} \\
 &= \frac{468}{676} = 0.692
 \end{aligned}$$

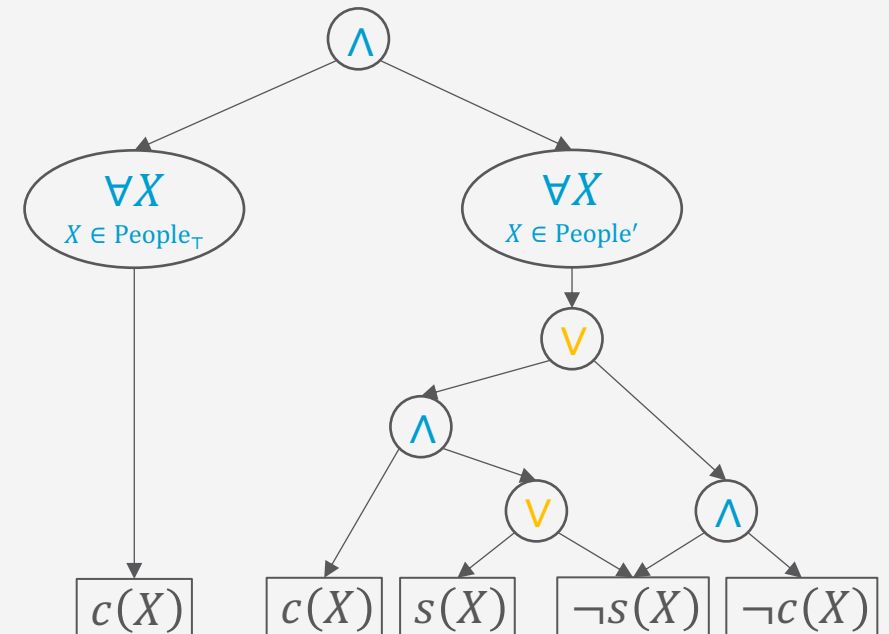


Conditioning in FO Circuits

- Evidence on propositional variables L
 - Replace leaf values with 0 where literal contradicts observation as in propositional circuits
- Evidence on unary variable $L(X)$
 - For *each* variable $L(X)$ that one wants to condition on,
 - Replace FOL-DC formula with three copies with additional domain constraints, simplify based on observation
 1. $X \in D_{\top}$ for observations $l(x)$
 2. $X \in D_{\perp}$ for observations $\neg l(x)$
 3. $X \notin D_{\top} \wedge X \notin D_{\perp}$ no observations
 - Compile a circuit for the extended theory
 - Given specific evidence, domains for $D_{\top}$, $D_{\perp}$ are determined
 - Might be empty
- Evidence on binary variable $L(X, Y)$
 - Can compile a circuit, no longer polynomial in time (reduction of #2SAT problem)

Conditioning in FO Circuits

- E.g., $\forall X \in \text{People} : s(X) \Rightarrow c(X)$ and $S(X)$
 - $\forall X \in \text{People}_\top : s(X) \Rightarrow c(X) \stackrel{s(X)}{\equiv} \forall X \in \text{People}_\top : c(X)$
 - $\forall X \in \text{People}_\perp : s(X) \Rightarrow c(X) \stackrel{\neg s(X)}{\equiv} \forall X \in \text{People}_\perp : \text{true}$
 - $\forall X \in \text{People}, X \notin \text{People}_\top, X \notin \text{People}_\perp : s(X) \Rightarrow c(X)$
- Delete Formula 2 as it is always true
- If one also wants to condition on $C(X)$, theory becomes larger again:
 - Formulas (1) and (3) contain $C(X)$ and therefore need to be replaced by three formulas, then simplify



First-order Knowledge Compilation (FOKC)

- Given
 - Theory Δ in FOL-DC in Skolem NNF
 - Weight function w_T for positive predicates, weight function w_F for negative predicates
 - Set of queries $\{P(q_i|\mathbf{e})\}_{i=1}^m$
- Build a WFOMC circuit $\mathcal{C}_\Delta$ for Δ , also preparing for evidence on $\text{rv}(\mathbf{e})$
- Condition on $\mathbf{e}$
- Calculate $WFOMC(\Delta \wedge \mathbf{e}, w_T, w_F)$ in $\mathcal{C}_\Delta$
- For each query $P(q_i|\mathbf{e})$
 - Build a WFOMC circuit $\mathcal{C}_{\Delta, q_i}$ for $\Delta \wedge q_i$ conditioned on $\mathbf{e}$
 - Compute $WFOMC(\Delta \wedge \mathbf{e} \wedge q_i, w_T, w_F)$ in $\mathcal{C}_{\Delta, q_i}$
 - Return or store $P(q_i|\mathbf{e}) = \frac{WFOMC(\Delta \wedge \mathbf{e} \wedge q_i, w_T, w_F)}{WFOMC(\Delta \wedge \mathbf{e}, w_T, w_F)}$

FOKC

MLNs for WFOMCs

- Weights in MLNs specified for formulas instead of single predicates
 - E.g., example from the beginning
 - $(\ln 7, travel(X) \wedge epid \wedge sick(X))$,
 - $(\ln 2, \neg travel(X) \vee \neg epid \vee \neg sick(X))$
- Trick:
 - Introduce a new predicate θ_i containing all free variables of ψ_i as equivalent to ψ_i
 - $\forall X \in \text{People} : \theta_1(X) \Leftrightarrow (travel(X) \wedge epid \wedge sick(X))$
 - $\forall X \in \text{People} : \theta_2(X) \Leftrightarrow (\neg travel(X) \vee \neg epid \vee \neg sick(X))$
 - Specify weight functions such that θ_i takes the weight of ψ_i
 - $w_T(\theta_1(X)) = \exp(\ln 7) = 7$
 - $w_T(\theta_2(X)) = \exp(\ln 2) = 2$
 - All other predicates and $\neg\theta_1, \neg\theta_2$ are mapped to 1 by both w_T, w_F

WFOMC Reduction

- Formally, given an MLN $\Psi = \{(w_i, \psi_i)\}_{i=1}^n$
 - Transform each weighted formula (w_i, ψ_i) into an FOL-DC formula
$$\forall \mathbf{X}_i, cs_i : \theta_i(\mathbf{X}_i) \Leftrightarrow \psi_i$$
 - where
 - $\mathbf{X}_i$ are the free variables in ψ_i
 - cs_i is the constraint set that enforces the domain constraints as given by the MLN
 - $\theta_i(\mathbf{X}_i)$ is a new predicate containing all free variables of ψ_i
 - Specify weight functions w_T, w_F such that for each
 - $w_T(\theta_i(\mathbf{X}_i)) = \exp(w_i)$
 - $w_T(p_i) = 1$ for all predicates p_i occurring in Ψ
 - $w_F(\theta_i(\mathbf{X}_i)) = 1$
- Continue with knowledge compilation

Example

- Given
 - $(\ln 7, travel(X) \wedge epid \wedge sick(X))$
 - $(\ln 2, \neg travel(X) \vee \neg epid \vee \neg sick(X))$
- Resulting theory $(t = travel, e = epid, s = sick)$
 - $\forall X \in \text{People} : \theta_1(X) \Leftrightarrow (t(X) \wedge e \wedge s(X))$
 - $\forall X \in \text{People} : \theta_2(X) \Leftrightarrow (\neg t(X) \vee \neg e \vee \neg s(X))$
 - with weight functions
 - $w_T(\theta_1(X)) = 7$
 - $w_T(\theta_2(X)) = 2$
 - Rest mapped to 1 by both w_T, w_F
- Transform formulas into CNF

Example: Normal Form

- Transform formulas into CNF: $\forall X \in \text{People} : \theta_1(X) \Leftrightarrow (t(X) \wedge e \wedge s(X))$

$$\begin{aligned}
 & \theta_1(X) \Leftrightarrow (t(X) \wedge e \wedge s(X)) && \text{(resolve } \Leftrightarrow \text{)} \\
 \equiv & \left(\theta_1(X) \Rightarrow (t(X) \wedge e \wedge s(X)) \right) \wedge \left(\theta_1(X) \Leftarrow (t(X) \wedge e \wedge s(X)) \right) && \text{(De Morgan on } \Rightarrow \text{)} \\
 \equiv & \left(\neg \theta_1(X) \vee (t(X) \wedge e \wedge s(X)) \right) \wedge \left(\theta_1(X) \vee \neg(t(X) \wedge e \wedge s(X)) \right) && \text{(move } \neg \text{ inward)} \\
 \equiv & \left(\neg \theta_1(X) \vee (t(X) \wedge e \wedge s(X)) \right) \wedge \left(\theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X) \right) && \text{(distribute } \vee \text{)} \\
 \equiv & \left(\neg \theta_1(X) \vee t(X) \right) \wedge \left(\neg \theta_1(X) \vee e \right) \wedge \left(\neg \theta_1(X) \vee s(X) \right) \wedge \left(\theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X) \right) && \text{(CNF)}
 \end{aligned}$$

- Result (each conjunct as own formula):
 - $\forall X \in \text{People} : \neg \theta_1(X) \vee t(X)$
 - $\forall X \in \text{People} : \neg \theta_1(X) \vee e$
 - $\forall X \in \text{People} : \neg \theta_1(X) \vee s(X)$
 - $\forall X \in \text{People} : \theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$

Example: Normal Form

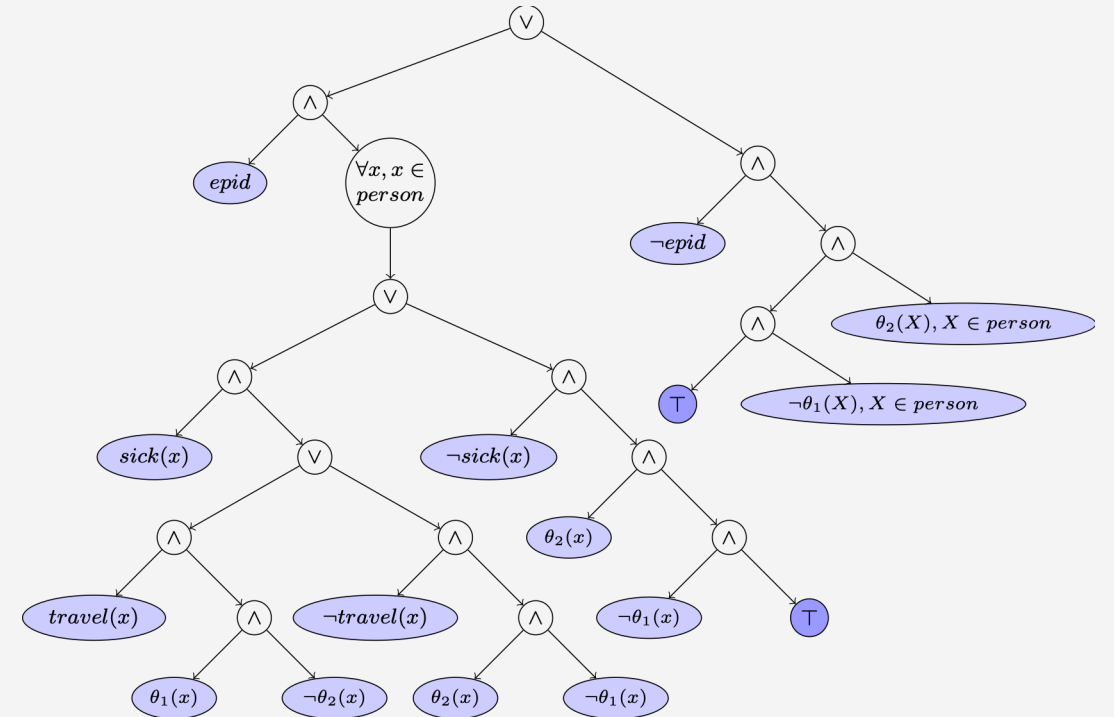
- Transform formulas into CNF: $\forall X \in \text{People} : \theta_2(X) \Leftrightarrow (\neg t(X) \vee \neg e \vee \neg s(X))$

$$\begin{aligned}
 & \theta_2(X) \Leftrightarrow (\neg t(X) \vee \neg e \vee \neg s(X)) \\
 & \equiv \left(\theta_2(X) \Rightarrow (\neg t(X) \vee \neg e \vee \neg s(X)) \right) \wedge \left(\theta_2(X) \Leftarrow (\neg t(X) \vee \neg e \vee \neg s(X)) \right) \\
 & \equiv (\neg \theta_2(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)) \wedge \left(\theta_2(X) \vee \neg(\neg t(X) \vee \neg e \vee \neg s(X)) \right) \\
 & \equiv (\neg \theta_2(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)) \wedge \left(\theta_2(X) \vee (t(X) \wedge e \wedge s(X)) \right) \\
 & \equiv (\neg \theta_2(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)) \wedge (\theta_2(X) \vee t(X)) \wedge (\theta_2(X) \vee e) \wedge (\theta_2(X) \vee s(X))
 \end{aligned}$$

- Result (each conjunct as own formula):
 - $\forall X \in \text{People} : \neg \theta_2(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee t(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee e$
 - $\forall X \in \text{People} : \theta_2(X) \vee s(X)$

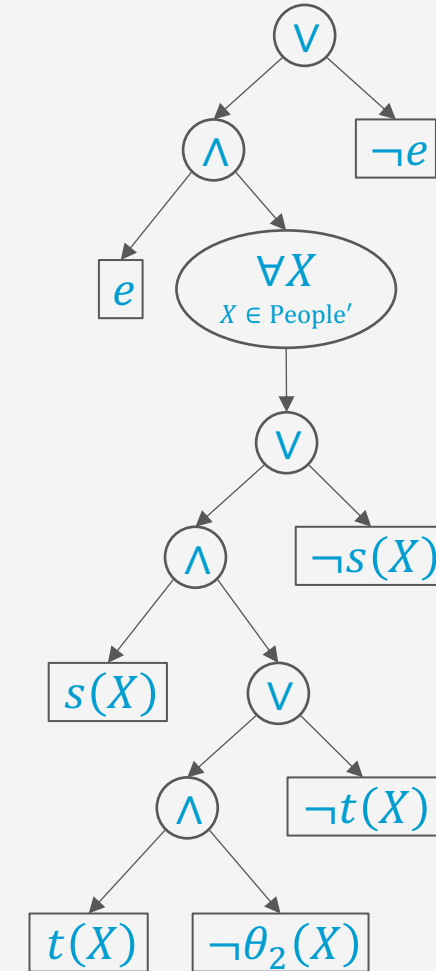
Example: FO d-DNNF Circuit

- Given theory in CNF
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee t(X)$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee e$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee s(X)$
 - $\forall X \in \text{People} : \theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 - $\forall X \in \text{People} : \neg\theta_2(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee t(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee e$
 - $\forall X \in \text{People} : \theta_2(X) \vee s(X)$
- Resulting FO d-DNNF circuit generated by the FOKC implementation
 - Some leaves repeated for readability



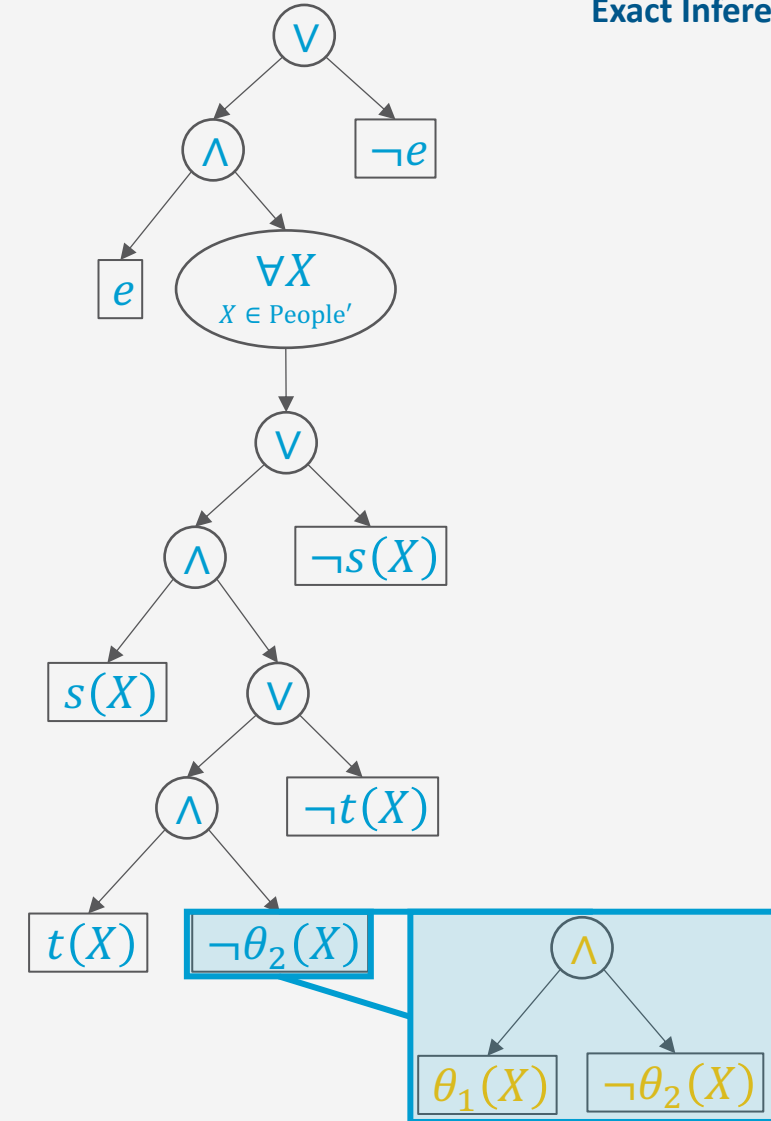
Example: FO d-DNNF Circuit

- Given theory in CNF
 - $\forall X \in \text{People} :$
 $\neg\theta_2(X) \vee \neg t(X) \vee \neg s(X) \vee \neg e$
 - $\forall X \in \text{People} :$
 $\theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee t(X)$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee e$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee s(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee t(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee e$
 - $\forall X \in \text{People} : \theta_2(X) \vee s(X)$



Example: FO d-DNNF Circuit

- Given theory in CNF
 - $\forall X \in \text{People} : \neg\theta_2(X) \vee \neg t(X) \vee \neg s(X) \vee \neg e$
 - $\forall X \in \text{People} : \theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee t(X)$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee e$
 - $\forall X \in \text{People} : \neg\theta_1(X) \vee s(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee t(X)$
 - $\forall X \in \text{People} : \theta_2(X) \vee e$
 - $\forall X \in \text{People} : \theta_2(X) \vee s(X)$



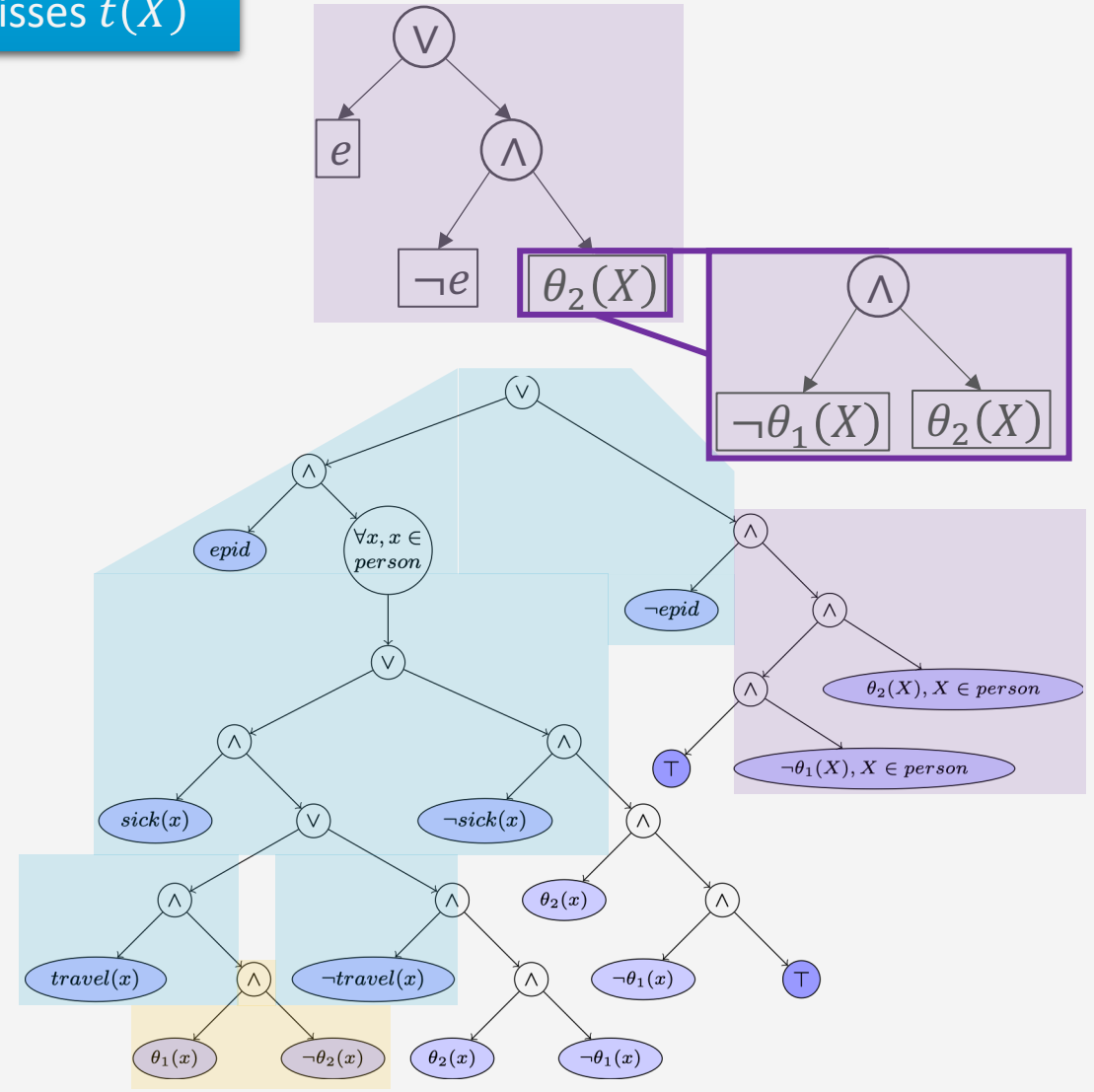
Not smooth since

- Right branch of root $\vee$ misses $s(X), t(X)$
- Right branch of $\vee$ after $\forall X$ misses $t(X)$

Exact Inference: FOKC

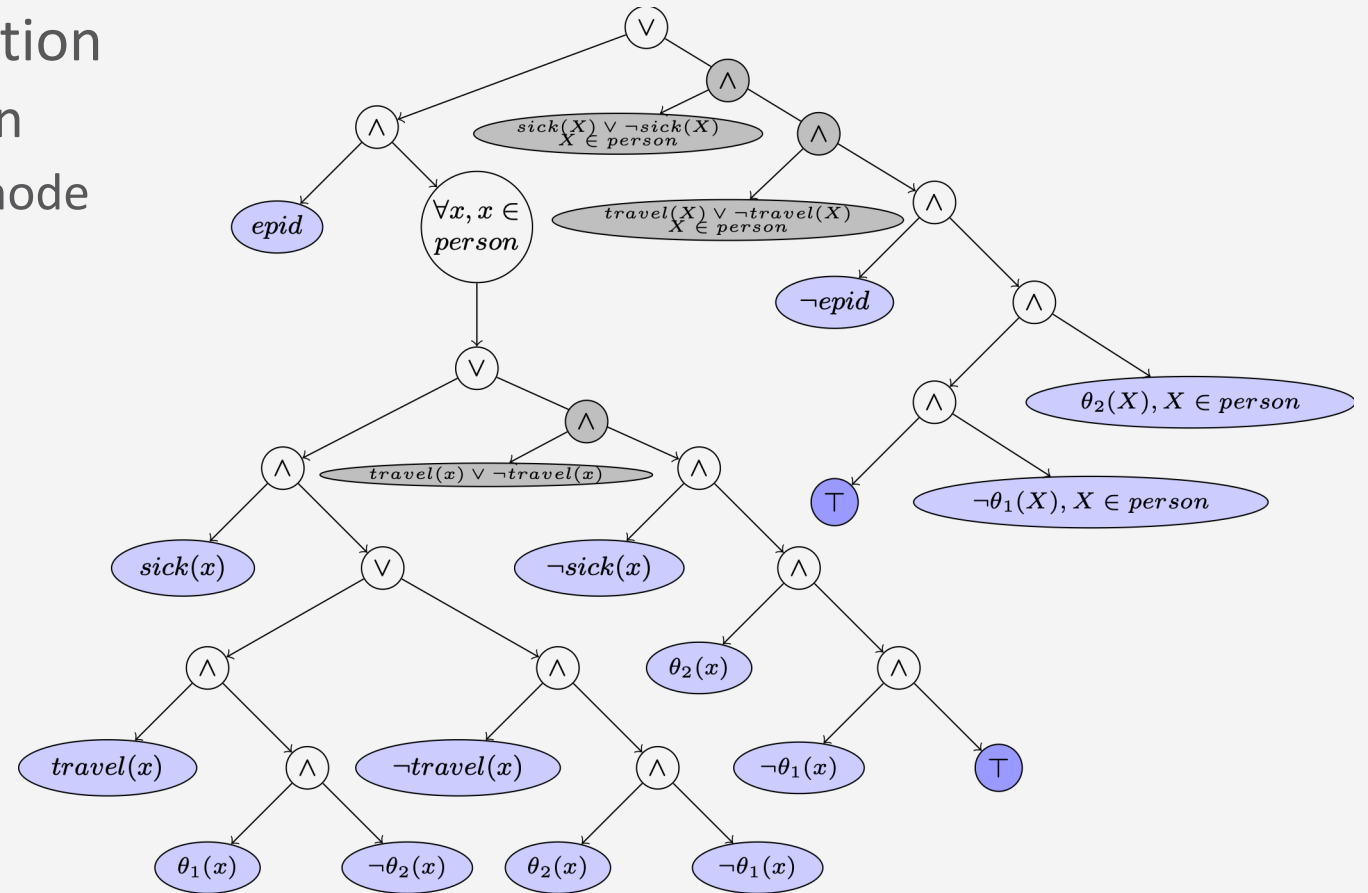
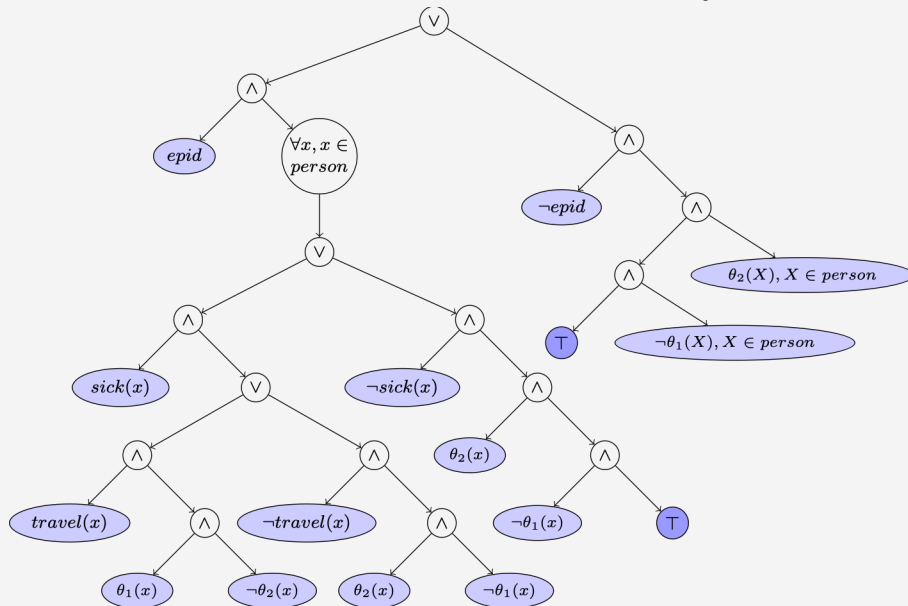
Example: FO d-DNNF Circuit

- Given theory in CNF
 1. $\forall X \in \text{People} : \neg\theta_2(X) \vee \neg t(X) \vee \neg s(X) \vee \neg e$
 2. $\forall X \in \text{People} : \theta_1(X) \vee \neg t(X) \vee \neg e \vee \neg s(X)$
 3. $\forall X \in \text{People} : \neg\theta_1(X) \vee t(X)$
 4. $\forall X \in \text{People} : \neg\theta_1(X) \vee e$
 5. $\forall X \in \text{People} : \neg\theta_1(X) \vee s(X)$
 6. $\forall X \in \text{People} : \theta_2(X) \vee t(X)$
 7. $\forall X \in \text{People} : \theta_2(X) \vee e$
 8. $\forall X \in \text{People} : \theta_2(X) \vee s(X)$



Example: Smoothed FO d-DNNF Circuit

- As generated by the FOKC implementation
- Grey parts new to not-smoothed version
 - Abbreviated depiction of $p \vee \neg p$ in one node
 - Not-smoothed version for comparison $\Downarrow$



Theoretical Results

- Compilation independent of domain sizes
 - Just like construction of FO jtree is also independent of domain sizes
- Inference
 - Polynomial in domain sizes
 - Based on the computations that are computed at different node types
- Completeness as before
 - $\mathcal{M}^{2lv}$
 - Two-logvar theories with max. two logical variables per formula
 - $\mathcal{M}^{1prv}$
 - One logical variable per predicate

Implementation

- Available at
 - <https://github.com/UCLA-StarAI/Forclift>
 - May no longer work according to Guy so you may have to try
 - <https://github.com/tanyabraun/wfomc>
 - Officially three input formats
 - Based on the normal form required (.wmc)
 - Early version of parfactor graphs (.fg)
 - MLN version (.mln)
 - MLN file format only one I got the compiled version to parse

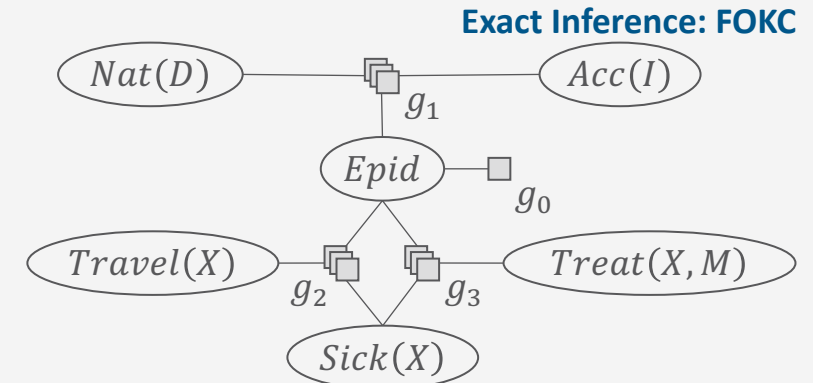
Runtimes: Increasing Domain Sizes

- Example model with all domain sizes $\in \{2, 4, \dots, 20, 30, \dots, 100, 200, \dots, 1000\}$
- No evidence
- Queries: $P(\text{Travel}(x_1))$, $P(\text{Sick}(x_1))$, $P(\text{Treat}(x_1, m_1))$, $P(\text{Nat}(d_1))$, $P(\text{Man}(w_1))$, $P(\text{Epid})$
- Compare query answering times of different inference algorithms
 - Propositional: VE, JT
 - Lifted: LVE, LJT, FOKC
 - Compare trade-off (overhead vs. fast inference) between single / multi-query algs.

• Test

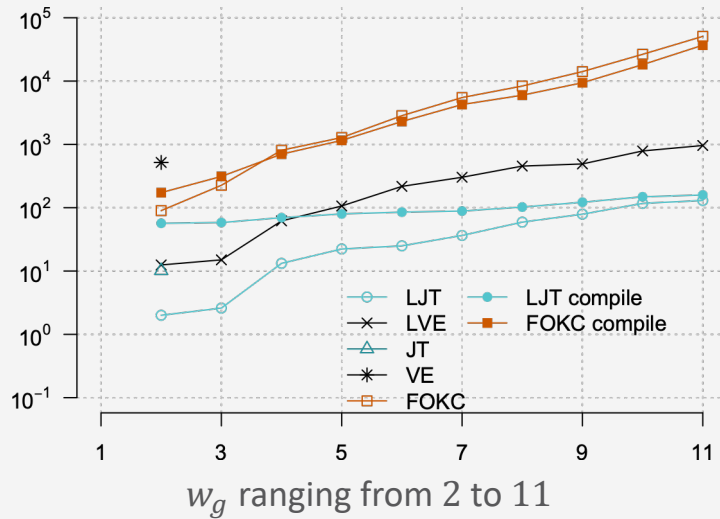
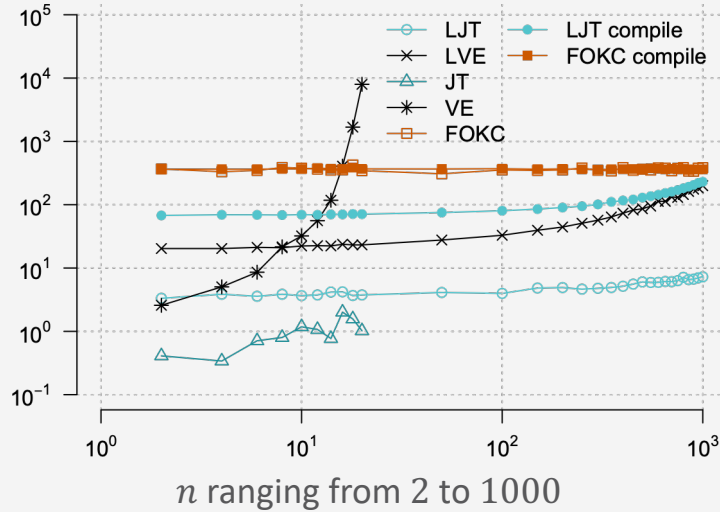
• Increasing

- Ground width w_g
 - Default: 3
- Counting width $w_\#$
 - Default: 1
- Number of nodes n_J
 - Default: 3
- Domain size n
 - Default: 1000
- Based on $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\# w_\#})$

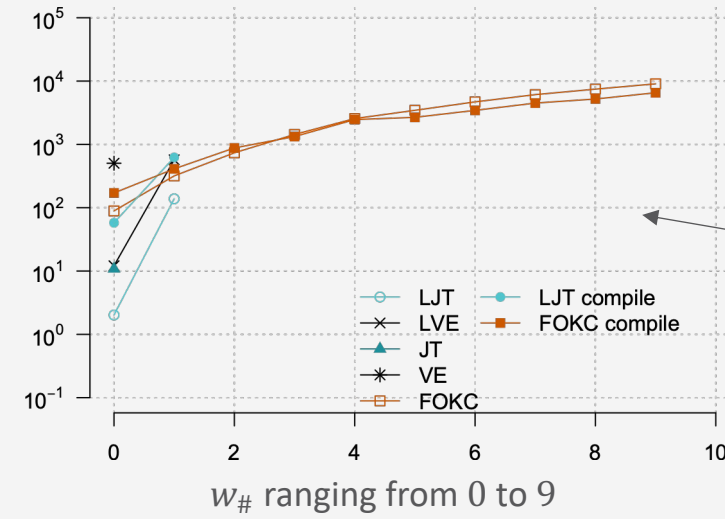
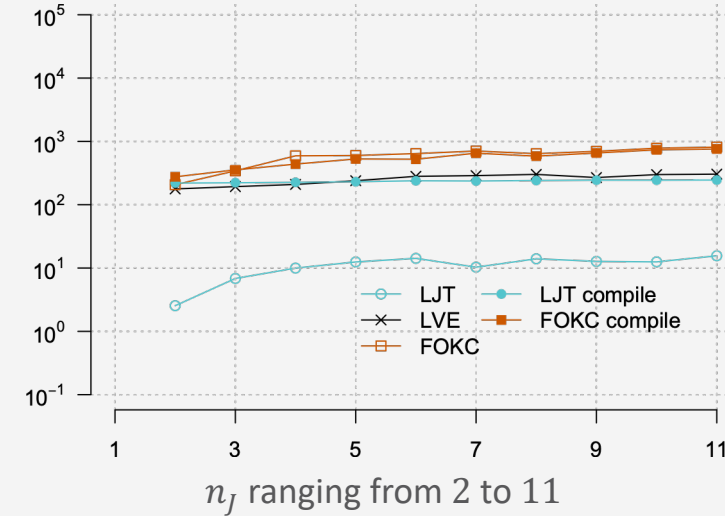


Queries Answering

FOKC almost invariant
w.r.t. domain sizes
whereas count
conversion hits LVE-
based algorithms



compile: all overhead time **Exact Inference: FOKC**



FOKC does not
build histograms,
which blow up the
representation for
LVE-based algs.

Trade-off Evaluation: Criteria

- For multi-query algorithms
 - Overhead to set off (model is *compiled* into a helper structure)
- vs.
- Shorter individual query answering time
- With
 - $t_{q,cpl}$ runtime for answering single query with an algorithm that uses compilation
 - $t_{q,uncpl}$ runtime for answering single query with an algorithm without compilation
 - $t_{c,cpl}$ runtime for compilation with an algorithm that uses compilation

- What is the ratio between individual query answering times?

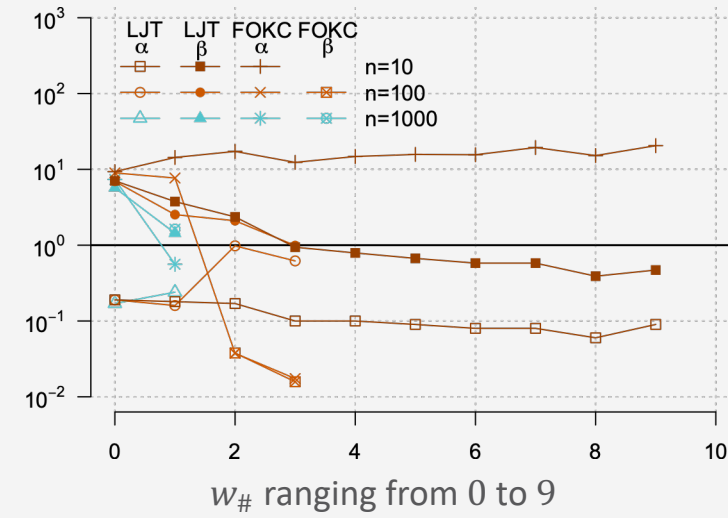
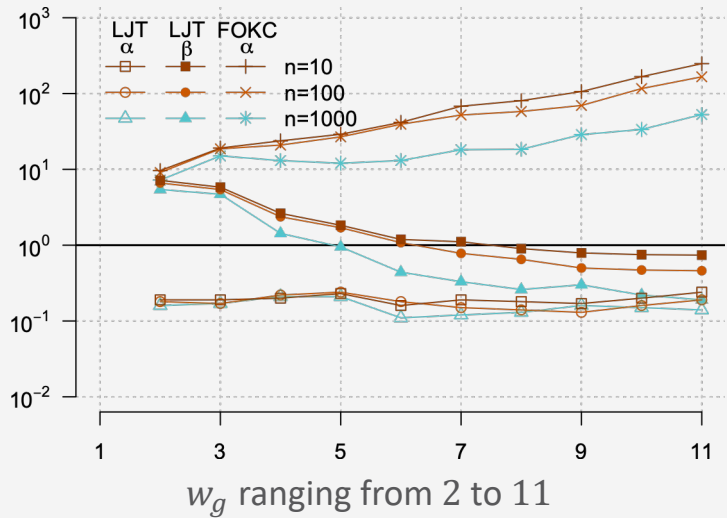
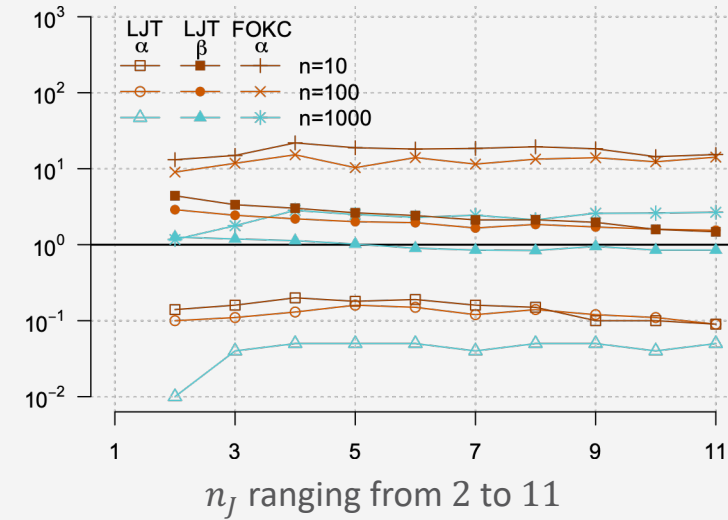
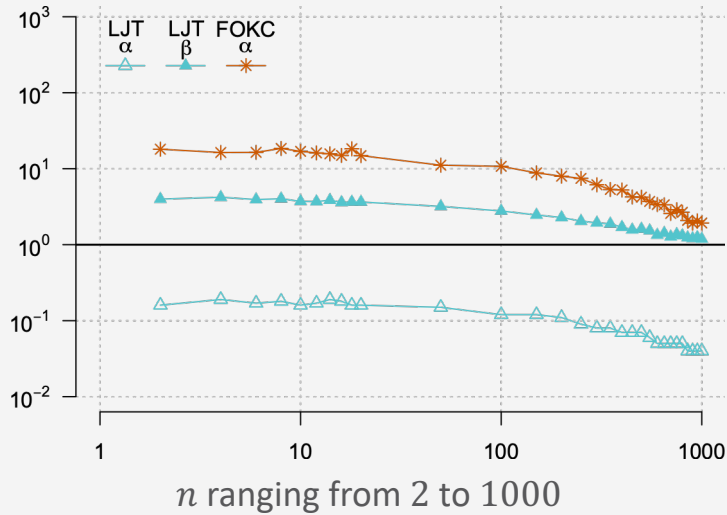
$$\alpha = \frac{t_{q,cpl}}{t_{q,uncpl}}$$

- How many queries does it take to offset the overhead?

$$\beta = \frac{t_{c,cpl}}{t_{q,uncpl} - t_{q,cpl}}$$

- Makes only sense if $\alpha > 1$

Trade-off



Probabilistic Theorem Proving (PTP)

- Based on theorem proving in logics
- Solves lifted weighted model counting problem
 - Similar to the weighted first-order model counting problem by Guy Van den Broeck
 - MLNs as input
- Implementation available: Alchemy
 - <http://alchemy.cs.washington.edu>
 - Input format: MLNs

Interim Summary

- Lifted (weighted) model counting
 - WFOMC definition
- FO circuits
 - Inner nodes can also be set conjunctions/disjunctions
 - Analogous FO dDNNF properties apply
- FOKC: Inference in FO circuits
 - Allows for tractable inference
 - Same completeness results as before
 - Runtime almost invariant to changing domain sizes in contrast to LVE-based algorithms due to count conversion
 - But: You cannot really count in MLNs → [Works by Ondřej Kuželka on counting in WFOMC](#)

Outline: 3. Inference

a. Exact Inference

i. (Weighted) Model Counting

- Model counting
- Weighted model counting, knowledge compilation (KC)

ii. Variable Elimination and the Like

- Variable elimination (VE)
- Junction tree algorithm (JT)

iii. Lifted Inference

- First-order KC, Lifted VE, lifted JT

b. Approximate Inference

- Direct sampling: Rejection sampling, (lifted) importance sampling
- (Lifted) Markov Chain Monte Carlo sampling

Appendix

LVE operator definitions

Example calculation with a greedy size-based heuristics (example model without g_0)

Formal definitions for FO dtrees

Example derivation of FO jtrees and LJT run

Count Normalisation: Formal Definition

- More general: Given a constraint $(\mathcal{X}, C_{\mathcal{X}})$, the same number of groundings of $Y \subseteq X$ exist for the different possible groundings of $Z \subseteq X \setminus Y$, with X the set of $\mathcal{X}$

- **Count function:**

Given a constraint $(\mathcal{X}, C_{\mathcal{X}})$, for any $Y \subseteq \mathcal{X}$ and $Z \subseteq \mathcal{X} \setminus Y$, the function $\text{count}_{Y|Z}(t) : C_{\mathcal{X}} \rightarrow \mathbb{N}$ is defined by

$$\text{count}_{Y|Z}(t) = |\pi_Y(C_{\mathcal{X}} \bowtie \pi_Z(\{t\}))|$$

- **Count-normalisation:**

Y is count-normalised w.r.t. to Z iff $\exists n \in \mathbb{N}$ s.t.

$$\forall t \in C_{\mathcal{X}} : \text{count}_{Y|Z}(t) = n$$

- Conditional count of Y given Z , denoted $\text{ncount}_{Y|Z}((\mathcal{X}, C_{\mathcal{X}}))$

Example

- $\text{count}_{Y|Z}(t) = |\pi_Y(C_X \bowtie \pi_Z(\{t\}))|$
 - E.g.,
 - $\mathcal{X} = (X, M)$
 - $Y = \{M\}$
 - $Z = \mathcal{X} \setminus Y = \{X\}$
 - With $a = \text{alice}, e = \text{eve}, b = \text{bob} : C_X = \{(a, m_2), (e, m_1), (e, m_2), (b, m_1), (b, m_2)\}$

$\text{count}_{M|X}((a, m_2))$

$\pi_X(\{(a, m_2)\}) = \{(a)\}$

$C_{X,M} \bowtie \{(a)\} = \{(a, m_2)\}$

$\pi_M(\{(a, m_2)\}) = \{(m_2)\}$

$|\{(m_2)\}| = 1$

$\text{count}_{M|X}((e, m_1))$

$\pi_X(\{(e, m_1)\}) = \{(e)\}$

$C_{X,M} \bowtie \{(e)\} = \{(e, m_1), (e, m_2)\}$

$\pi_M(\{(e, m_1), (e, m_2)\}) = \{(m_1), (m_2)\}$

$|\{(m_1), (m_2)\}| = 2$
 - Not count-normalised: $1 \neq 2$

Adding (a, m_1) to C_X leads to
 $\text{count}_{M|X}((a, m_2)) = 2$
 $\rightarrow M$ is count-normalised w.r.t. X

Lifted Summing Out: Operator

- Inputs:
 - Parfactor $g = \phi(\mathcal{A})_{|C}$, $C = (\mathcal{X}, C_{\mathcal{X}})$
 - PRV A_i occurring in $\mathcal{A}$ for summing out
- Preconditions:
 - $\forall B \in \text{rv}(G \setminus \{g\}) : \text{gr}(B_{|C}) \cap \text{gr}(A_i_{|(\mathcal{X}, C_{\mathcal{X}})}) = \emptyset$
 - $\forall X \in \{X \mid |\pi_X(C_{\mathcal{X}})| > 1\} : X \in \text{lv}(A_i)$
 - $X^{\text{excl}} = \text{lv}(A_i) \setminus (\mathcal{X} \setminus \text{lv}(A_i))$ count-normalised w.r.t. $X^{\text{com}} = \text{lv}(A_i) \cap \mathcal{X}$ in C : $r = \text{ncount}_{X^{\text{excl}}|X^{\text{com}}}(C)$
- Output: $\phi'(\mathcal{A}')_{|C'}$ with $C' = (\pi_{X^{\text{com}}}(\mathcal{X}), \pi_{X^{\text{com}}}(C_{\mathcal{X}}))$
 - $\mathcal{A}' = (A_1, \dots, A_{i-1}) \circ (A_{i+1}, \dots, A_n)$ (concatenation of two sequences)
 - For each assignment $\mathbf{a}' = (\dots, a_{i-1}, a_{i+1}, \dots)$ to $\mathcal{A}'$, i.e., $\forall \mathbf{a}' \in \text{ran}(\mathcal{A}')$

$$\phi'(\dots, a_{i-1}, a_{i+1}, \dots) = \left(\sum_{a_i \in \text{ran}(A_i)} \text{Mul}(a_i) \phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots) \right)^r$$
- Postcondition: $P_{G \setminus \{g\} \cup \{\text{sum-out}(g, A_i)\}} = \sum_{\text{gr}(A_i|C)} P_G$

Multinomial coefficient to eliminate CRVs correctly

$$\text{Mul}(a_i) = \begin{cases} \frac{n!}{\prod_{i=1}^m n_i!} & a_i = h \\ 1 & \text{oth.} \end{cases}$$

Lifted Multiplication: Operator

- Inputs:
 - Parfactor $g_1 = \phi_1(\mathcal{A}_1)_{|C_1}, C_1 = (\mathcal{X}_1, C_{\mathcal{X}_1})$
 - Parfactor $g_2 = \phi_2(\mathcal{A}_2)_{|C_2}, C_2 = (\mathcal{X}_2, C_{\mathcal{X}_2})$
 - One-to-one substitution $\theta = \{\mathbf{Z}_1 \rightarrow \mathbf{Z}_2\}$ between the logical variables of the shared PRVs in g_1 and g_2
- Preconditions:
 - For $i = 1, 2$: $\mathbf{Y}_i = \mathbf{X}_i \setminus \mathbf{Z}_i$ count-normalised w.r.t. $\mathbf{Z}_i$ in C_i , with $\mathbf{X}_i$ the set of $\mathcal{X}_i$, i.e., $r_i = \text{ncount}_{\mathbf{Y}_i|\mathbf{Z}_i}(C_i)$ exists
- Output: $\phi(\mathcal{A})_{|C}$ with $C = (\mathcal{X}_1\theta \bowtie \mathcal{X}_2, C_{\mathcal{X}_1\theta} \bowtie C_{\mathcal{X}_2})$
 - $\mathcal{A} = \mathcal{A}_1\theta \bowtie \mathcal{A}_2$
 - For each assignment $\mathbf{a}$ to $\mathcal{A}$, with $\mathbf{a}_1 = \pi_{\mathcal{A}_1\theta}(\mathbf{a})$ and $\mathbf{a}_2 = \pi_{\mathcal{A}_2\theta}(\mathbf{a})$

$$\phi(\mathbf{a}) = (\phi_1(\mathbf{a}_1))^{\frac{1}{r_2}} \cdot (\phi_2(\mathbf{a}_2))^{\frac{1}{r_1}}$$
- Postcondition: $G \sim G \setminus \{g_1, g_2\} \cup \{\text{multiply}(g_1, g_2, \theta)\}$

Operator does not assume that logical variables with the same applicable constants share the same name

Lifted Multiplication: Example

$$g_1 \cdot g_2 = \phi_1(\text{Sick}(X)) \cdot \phi_2(\text{Sick}(X), \text{Treat}(X, M)) = \phi(\text{Sick}(X), \text{Treat}(X, M))$$

- T constraints with $|\mathcal{D}(M)| = 2$
- 1-to- m
 - $X_1 = \text{lv}(g_1) = \{X\}$
 - $X_2 = \text{lv}(g_2) = \{X, M\}$
 - $Z_1 = \text{lv}(\text{Sick}(X)) = \{X\} = Z_2$
 - No alignment necessary
 - $Y_1 = X_1 \setminus Z_1 = \emptyset$
count-normalised w.r.t. $Z_1 = \{X\}$
 - $Y_2 = X_2 \setminus Z_2 = \{M\}$
count-normalised w.r.t. $Z_2 = \{X\}$
 - Scaling necessary: $r_2 = \text{ncount}_{Y_2|Z_2}(C_2) = 2$

$\pi_{\text{Sick}(X)}$			
<i>Sick(X)</i>	<i>Treat(X, M)</i>	ϕ	
false	false	1 · 5	·
false	true	1 · 6	
true	false	2 · 7	
true	true	2 · 8	

ϕ_1		
<i>Sick(X)</i>		
false	1	$1^{\frac{1}{2}} = \sqrt[2]{1} = 1$ $4^{\frac{1}{2}} = \sqrt[2]{4} = 2$
true	4	

$\pi_{\text{Sick}(X), \text{Treat}(X, M)}$			
<i>Sick(X)</i>	<i>Treat(X, M)</i>	ϕ_2	
false	false	5	·
false	true	6	
true	false	7	
true	true	8	

Count Conversion: Operator

- Inputs:
 - Parfactor $g = \phi(\mathcal{A})|_{\mathcal{C}}, \mathcal{C} = (\mathcal{X}, C_{\mathcal{X}})$
 - Logical variable X occurring in $\mathcal{X}$ for counting
- Preconditions:
 - There is exactly one PRV $A_i \in \text{rv}(g)$ s.t. $X \in \text{lv}(A_i)$
 - X is count-normalised w.r.t. $\mathcal{X} \setminus \{X\}$ in $\mathcal{C}$
 - For all counted logical variables $X^{\#}$ in g : $\pi_{X, X^{\#}}(C_{\mathcal{X}}) = \pi_X(C_{\mathcal{X}}) \times \pi_{X^{\#}}(C_{\mathcal{X}})$
- Output: $\phi'(\mathcal{A}')|_{\mathcal{C}}$
 - $\mathcal{A}' = (A_1, \dots, A_{i-1}) \circ (A'_i) \circ (A_{i+1}, \dots, A_n), A'_i = \#_X[A_i]$
 - For each assignment $\mathbf{a}' = (\dots, a_{i-1}, h, a_{i+1}, \dots)$ to $\mathcal{A}'$,

$$\phi'(\dots, a_{i-1}, h, a_{i+1}, \dots) = \prod_{a_i \in \text{ran}(A_i)} \phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots)^{h(a_i)}$$
 - With $h(a_i)$ denoting the count of a_i in histogram h
- Postcondition: $G \sim G \setminus \{g\} \cup \{\text{count-convert}(g, X)\}$

No inequality constraint between X and any other counted logical variable $X^{\#}$

Count Conversion: Example

- From $\phi_1(Epid, Nat(D), Acc(I))$
- To $\phi'_1(Epid, Nat(D), \#_I[Acc(I)])$
- Preconditions fulfilled
 - I occurs only in $Acc(I)$
 - I is count-normalised w.r.t. D in $((D, I), \text{dom}(D) \times \text{dom}(I))$
 - No other counted logical variable
- Converting $Acc(I)$ into $\#_I[Acc(I)]$
 $\phi'(\dots, a_{i-1}, \textcolor{brown}{h}, a_{i+1}, \dots)$
 $= \prod_{a_i \in \text{ran}(A_i)} \phi(\dots, a_{i-1}, a_i, a_{i+1}, \dots) \textcolor{brown}{h}(a_i)$

<i>Epid</i>	<i>Nat(D)</i>	$\#_I[Acc(I)]$	ϕ'_1
false	false	[0,2]	$2^0 \cdot 1^2$
false	false	[1,1]	$2^1 \cdot 1^1$
false	false	[2,0]	$2^2 \cdot 1^0$
false	true	[0,2]	$4^0 \cdot 3^2$
false	true	[1,1]	$4^1 \cdot 3^1$
false	true	[2,0]	$4^2 \cdot 3^0$
true	false	[0,2]	$6^0 \cdot 5^2$
true	false	[1,1]	$6^1 \cdot 5^1$
true	false	[2,0]	$6^2 \cdot 5^0$
true	true	[0,2]	$8^0 \cdot 7^2$
true	true	[1,1]	$8^1 \cdot 7^1$
true	true	[2,0]	$8^2 \cdot 7^0$

<i>Epid</i>	<i>Nat(D)</i>	<i>Acc(I)</i>	ϕ_1
false	false	false	1
false	false	true	2
false	true	false	3
false	true	true	4
true	false	false	5
true	false	true	6
true	true	false	7
true	true	true	8

Splitting

- Need splitting for
 - Shattering of query terms and evidence
 - Precondition 1 of sum—out operator: PRV A under $(\mathcal{X}, C_{\mathcal{X}})$ only occurs in g
 - Formalism is very flexible in terms of constraints
 - E.g., $\phi_1(R(X))_{(X, \{x_1, x_2, x_3\})}$ vs. $\phi_2(R(X))_{(X, \{x_1, x_2, x_3, x_4 x_5\})}$
- Split parfactor s.t. the set of constants occurring in constraints for a logical variable are either *identical* or *disjoint*
 - I.e., no overlaps between sets of constants per logical variable
 - E.g., split $\phi_2(R(X))_{(X, \{x_1, x_2, x_3, x_4 x_5\})}$ into
 - $\phi_2(R(X))_{(X, \{x_1, x_2, x_3\})}$
 - $\phi_2(R(X))_{(X, \{x_4 x_5\})}$

So, we need a formal *split* operation to split of (set of) constants

Splitting on Overlap

- Splitting a constraint $C_1 = (\mathcal{X}_1, C_{\mathcal{X}_1})$ on its Y -overlap with a constraint $C_2 = (\mathcal{X}_2, C_{\mathcal{X}_2})$, denoted $C_1/_Y C_2$, *partitions* $C_{\mathcal{X}_1}$ into two subsets containing all tuples for which the Y part occurs or does not occur, respectively

$$C_1/_Y C_2 = \left\{ \begin{array}{l} \left((\mathcal{X}_1), \{t \in C_{\mathcal{X}_1} \mid \pi_Y(\{t\}) \in \pi_Y(C_{\mathcal{X}_2})\} \right) \\ \left((\mathcal{X}_1), \{t \in C_{\mathcal{X}_1} \mid \pi_Y(\{t\}) \notin \pi_Y(C_{\mathcal{X}_2})\} \right) \end{array} \right\} \begin{array}{l} \text{Part shared with } C_2 \\ \text{Remaining part} \end{array}$$

- Parfactor partitioning

Given a parfactor $g = \phi(\mathcal{A})|_C$ and a partition $\mathbb{C} = \{C_i\}_{i=1}^n$ of C ,

$$\text{partition}(g, \mathbb{C}) = \{\phi(\mathcal{A})|_{C_i}\}_{i=1}^n$$

Splitting on Overlap: Example

- Consider $\phi(R(X), T(X, Y))|_{C_1}$
 - $C_1 = ((X, Y), \{x_1, x_2, x_3, x_4, x_5\} \times \{y_1, y_2\})$
 - $C_2 = ((X), \{x_1, x_2, x_3\})$
- Splitting C_1 on its $Y = \{X\}$ -overlap with C_2

$$C_{1/Y}C_2 = \left\{ \begin{array}{l} ((X, Y), \{t \in C_{(X,Y)} \mid \pi_X(\{t\}) \in \{x_1, x_2, x_3\}\}) \\ ((X, Y), \{t \in C_{(X,Y)} \mid \pi_X(\{t\}) \notin \{x_1, x_2, x_3\}\}) \end{array} \right\} = \left\{ \begin{array}{l} ((X, Y), \{x_1, x_2, x_3\} \times \{y_1, y_2\}) \\ ((X, Y), \{x_4, x_5\} \times \{y_1, y_2\}) \end{array} \right\}$$

- Partitioning

$$\text{partition}(g, C_{1/Y}C_2) = \left\{ \begin{array}{l} \phi(R(X), T(X, Y))|_{((X,Y), \{x_1, x_2, x_3\} \times \{y_1, y_2\})} \\ \phi(R(X), T(X, Y))|_{((X,Y), \{x_4, x_5\} \times \{y_1, y_2\})} \end{array} \right\}$$

Splitting: Operator

- Inputs:
 - Parfactor $g = \phi(\mathcal{A})|_C, C = (\mathcal{X}, C_{\mathcal{X}})$
 - PRV $A = R(Y)$ occurring in $\mathcal{A}$
 - PRV $A' = R(Y)|_{C'}$ or $\#_Y[R(Y)|_{C'}]$

- Precondition: *none*

- Output:

$$\text{partition}(g, \mathbb{C}), \mathbb{C} = C /_Y C' \setminus \emptyset$$

- Postcondition:

$$G \sim G \setminus \{g\} \cup \text{split}(g, A, A')$$

Splitting: Example

- Inputs:
 - Parfactor $\phi(R(X), T(X, Y))_{|C_1}$
 - $C_1 = ((X, Y), \{x_1, x_2, x_3, x_4, x_5\} \times \{y_1, y_2\})$
 - $R(X)$
 - $R(X)_{|C_2}, C_2 = ((X), \{x_1, x_2, x_3\})$
- Output:

$$\text{partition}(g, C_1 /_Y C_2) = \left\{ \begin{array}{l} \phi(R(X), T(X, Y))_{|((X, Y), \{x_1, x_2, x_3\} \times \{y_1, y_2\})} \\ \phi(R(X), T(X, Y))_{|((X, Y), \{x_4, x_5\} \times \{y_1, y_2\})} \end{array} \right\}$$

Other Operators

- Further “enablers” of lifted summing out all are *variants of splitting on overlap and partitioning*
- Splitting of CRVs: Operator called **expand**
 - More complex as histograms have to be split
 - E.g., a histogram $[1,3]$ for $\{x_1, x_2, x_3, x_4\}$ may have to be split on $\{x_1, x_2\}$
- Count-normalisation: Operator called **count—normalise**
 - Split a constraint s.t. in the set of resulting constraints, each constraint is count-normalised w.r.t. to desired $\mathbf{Y}_i | \mathbf{Z}_i$ property
 - Group sets of constants by the different counts $\text{ncount}_{\mathbf{Y}_i | \mathbf{Z}_i}(C_i)$ they yield
- Grounding – the last resort: Operator **ground** as expected
 - Splitting on individual constants
- More information:

Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel: Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. In: *Journal of Artificial Intelligence Research*, 2013. (or in Nima Taghipour’s PhD thesis)

Lifted Absorption: Operator

- Inputs:
 - Parfactor $g = \phi(\mathcal{A})|_C, C = (\mathcal{X}, C_{\mathcal{X}})$
 - PRV $A_i = R(Y)$ or (P)CRV $A_i = \#_X[R(Y)]$ occurring in $\mathcal{A}$
 - Evidence parfactor $g_e = \phi(R(Y))|_{C_e}$ with o = observed value of $R(Y)$ in g_e
- Let
 - $\mathbf{X}^{excl} = Y \setminus \text{lv}(\mathbf{A} \setminus \{A_i\})$ (exclusive to A_i), $\mathbf{X}^{nce} = \text{lv}(A_i) \setminus \text{lv}(\mathbf{A} \setminus \{A_i\})$ (not-counted exclusive to A_i)
 - $\mathbf{X}^{rem} = \text{lv}(\mathcal{X}) \setminus \mathbf{X}^{excl}$ (remaining in g), $\mathbf{X}^{ncr} = \text{lv}(\mathcal{A}) \setminus \mathbf{X}^{excl}$ (not-counted remaining in g)
- Preconditions:
 1. $\text{gr}(A_i|_C) \subseteq \text{gr}(A_i|_{C_e})$
 2. $\mathbf{X}^{nce}$ is count-normalised w.r.t. $\mathbf{X}^{ncr}$ in C , i.e., $r = \text{ncount}_{\mathbf{X}^{nce}|\mathbf{X}^{ncr}}(C)$ exists
- Output: $g' = \phi'(\mathcal{A}')|_{C'}, C' = (\pi_{\mathbf{X}^{rem}}(\mathcal{X}), \pi_{\mathbf{X}^{rem}}(C_{\mathcal{X}}))$
 - $\mathcal{A}' = (A_1, \dots, A_{i-1}) \circ (A_{i+1}, \dots, A_n)$

$$\phi'(\dots, a_{i-1}, a_{i+1}, \dots) = \phi(\dots, a_{i-1}, e, a_{i+1}, \dots)^r$$
 - with $e = o$ if $A_i = R(Y)$ and
 - otherwise e = a histogram with $e(o) = \text{ncount}_{X|\text{lv}(\mathcal{A})}(C)$ and $e(o') = 0, o' \neq o$
- Postcondition: $G \cup \{g_e\} \sim G \setminus \{g\} \cup \{g_e, \text{absorb}(g, A_i, g_e)\}$

Lifted Absorption: Evidence for CRVs

- Output: $g' = \phi'(\mathcal{A}')|_{C'}$
 $\phi'(\dots, a_{i-1}, a_{i+1}, \dots) = \phi(\dots, a_{i-1}, e, a_{i+1}, \dots)^r$
 - e = a histogram with $e(o) = \text{ncount}_{X|\text{lv}(\mathcal{A})}(C)$ and $e(o') = 0, o' \neq o$
- Evidence PRV appears as inner PRV of a (P)CRV
 - Turn observations into histogram
 - All groundings have the same observation in evidence parfactor
 - Peak-shaped histogram with $\text{ncount}_{X|\text{lv}(\mathcal{A})}(C)$ at position o and 0 otherwise
 - E.g., $\text{Nat}(D) = \text{false}$ for all $\text{gr}(\text{Nat}(D)) \rightarrow o = \text{false}$ in g_e
 - Given parfactor: $g = \phi(\text{Epid}, \#_D[\text{Nat}(D)])$
 - $\text{ncount}_{D|\emptyset}(\top) = 2$
 - Forms histogram: $[0,2]$
 - Output: $g' = \phi'(\text{Epid})$

$\text{Nat}(D)$	ϕ_e
false	1
true	0

Epid	$\#_D[\text{Nat}(D)]$	$\phi^\#$
false	$[0,2]$	1
false	$[1,1]$	2
false	$[2,0]$	3
true	$[0,2]$	4
true	$[1,1]$	5
true	$[2,0]$	6

Shattering on Evidence & Absorption

- Given a set of evidence parfactors $\{g_e\}_{e=1}^m$ and a model $G = \{g_i\}_{i=1}^n$

- For each $g_e = \phi_e(A_e)_{|C_e}$:

Shattering

- For each $g_i = \phi_i(\mathcal{A})_{|C_i}$:

- If $A_e \in rv(g_i)$:
 - Split g_i on C_e , i.e.,

$$G \leftarrow G \setminus \{g_i\} \cup \text{split}(g_i, A_e, A_e|_{C_e})$$

- For each $g_e = \phi_e(A_e)_{|C_e}$:

Absorption

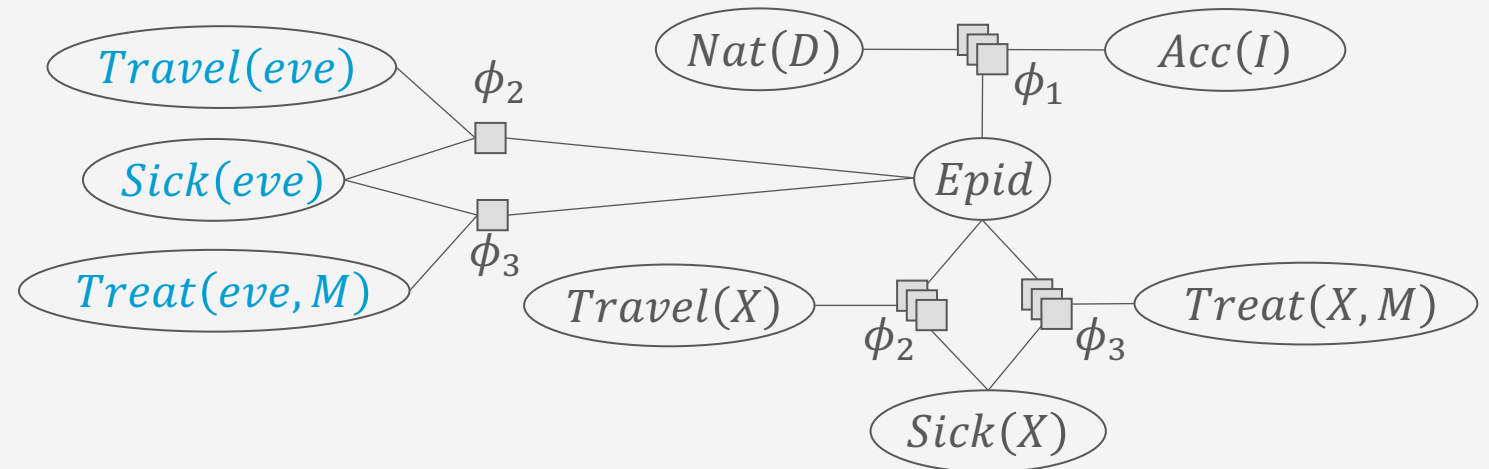
- For each $g_i = \phi_i(\mathcal{A})_{|C_i}$:

- If $A_e \in rv(g_i)$:
 - Absorb g_e in g_i

$$G \leftarrow G \setminus \{g_i\} \cup \text{absorb}(g_i, A_e, g_e)$$

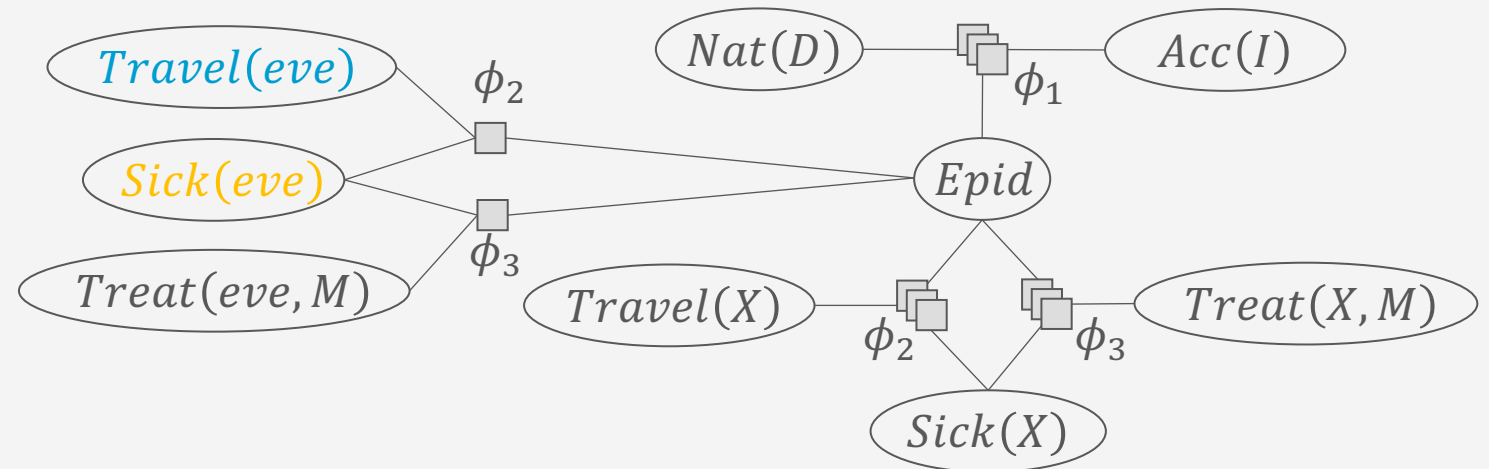
LVE: Example

- Model: $G = \{g_i\}_{i=1}^3$
 - T constraints
- Query term: $Travel(eve)$
- Evidence: $Sick(eve) = true$
- After shattering:



LVE: Example

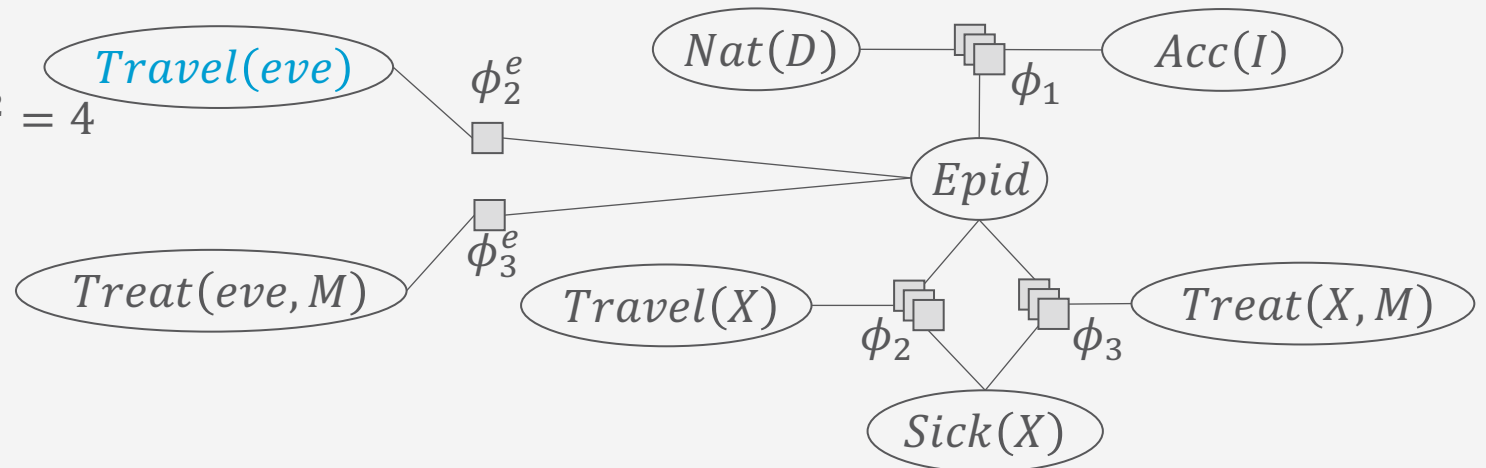
- Absorbing evidence $Sick(eve) = true$:
 - Absorb evidence in $\phi_2(Epid, Sick(eve), Travel(eve))$
 - Yields $\phi_2^e(Epid, Travel(eve))$
 - Absorb evidence in $\phi_3(Epid, Sick(eve), Treat(eve, M))$
 - Yields $\phi_3^e(Epid, Treat(eve, M))$



LVE: Example

- Eliminate all non-query terms
 - PRVs fulfilling sum–out preconditions:
 - $Treat(eve, M)$
 - Yields $\phi_3^{e'}(Epid) \rightarrow \text{size: } 2^1 = 2$
 - $Travel(X)$
 - Yields $\phi_2'(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$
 - $Treat(X, M)$
 - Yields $\phi_3'(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$

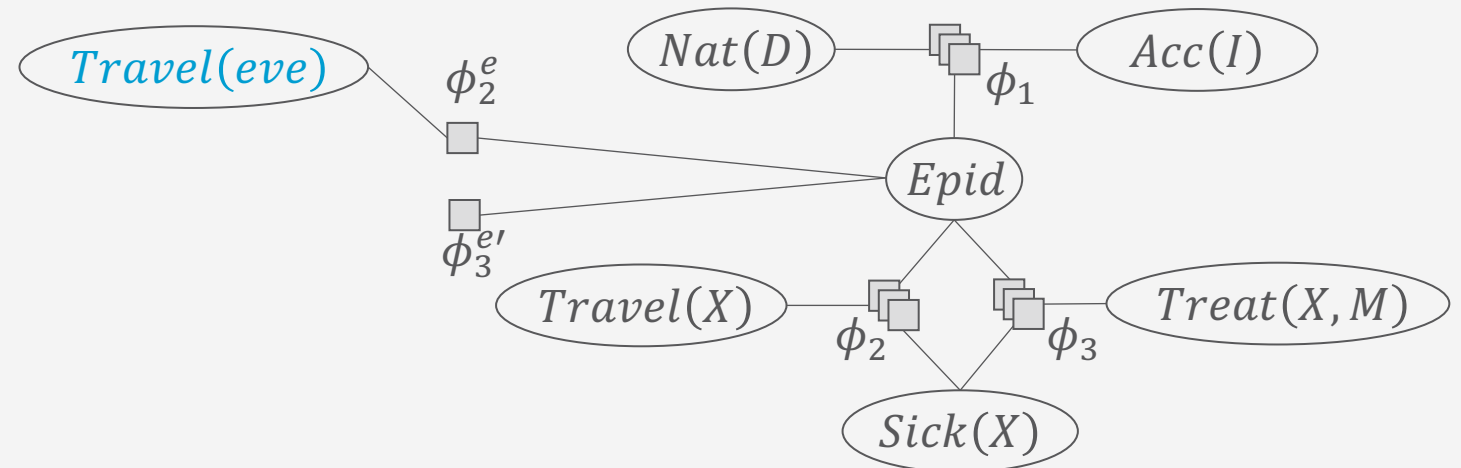
Smallest size



LVE: Example

- Eliminate all non-query terms
 - PRVs fulfilling sum-out preconditions:
 - Travel(X)*
 - Yields $\phi'_2(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$
 - Treat(X, M)*
 - Yields $\phi'_3(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$

Chosen at random

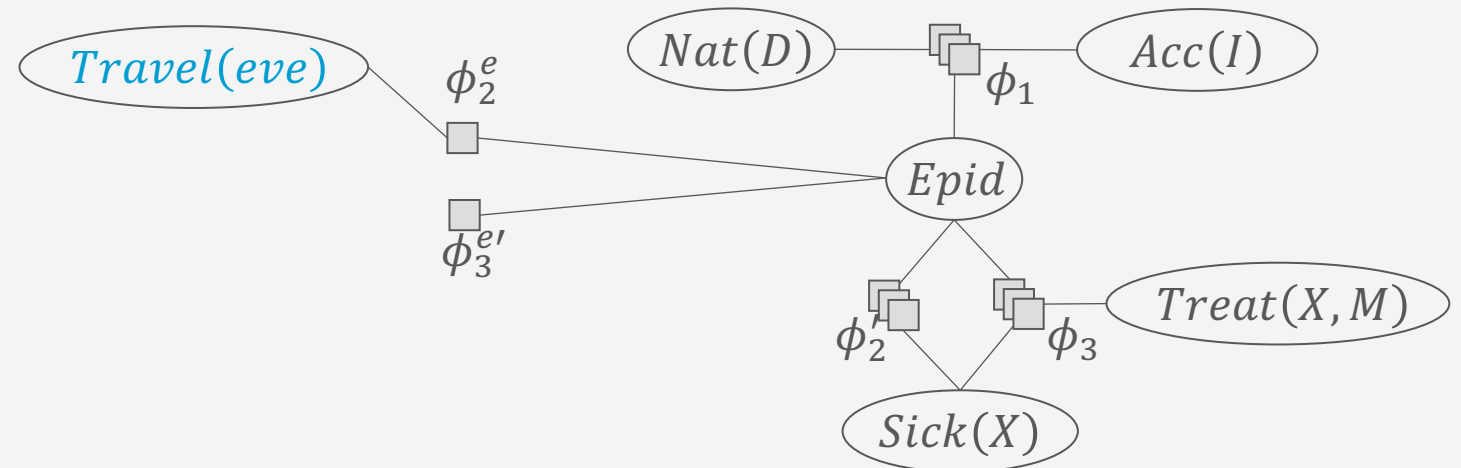


LVE: Example

- Eliminate all non-query terms
- PRVs fulfilling sum-out preconditions:

- $Treat(X, M)$
 - Yields $\phi'_3(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$

Only one



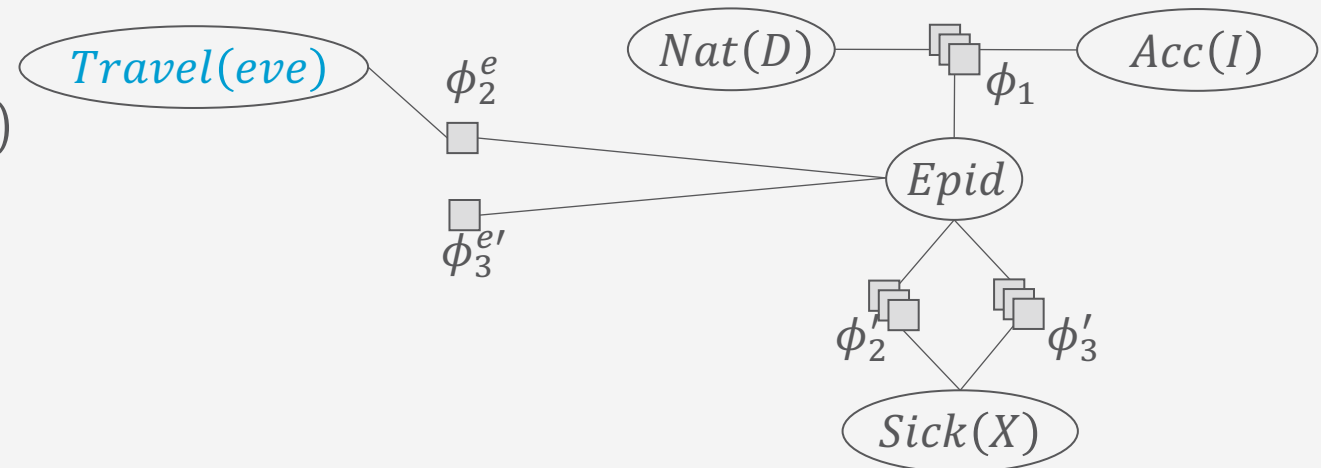
LVE: Example

- Eliminate all non-query terms
 - No PRVs fulfilling sum-out preconditions; others:

- Multiply ϕ'_2 and ϕ'_3
 - Yields $\phi'_{23}(Epid, Sick(X)) \rightarrow \text{size: } 2^2 = 4$

Chosen at random

- Multiply ϕ_2^e and $\phi_3^{e'}$
 - Yields $\phi'_{23}(Epid, Travel(eve)) \rightarrow \text{size: } 2^2 = 4$
- Count-convert $Nat(D)$
 - Yields $\phi_1^D(Epid, \#_D[Nat(D)], Acc(I)) \rightarrow \text{size: } 2 \cdot 3 \cdot 2 = 12$
- Count-convert $Man(W)$
 - Yields $\phi_1^I(Epid, Nat(D), \#_I[Acc(I)]) \rightarrow \text{size: } 2^2 \cdot 3 = 12$

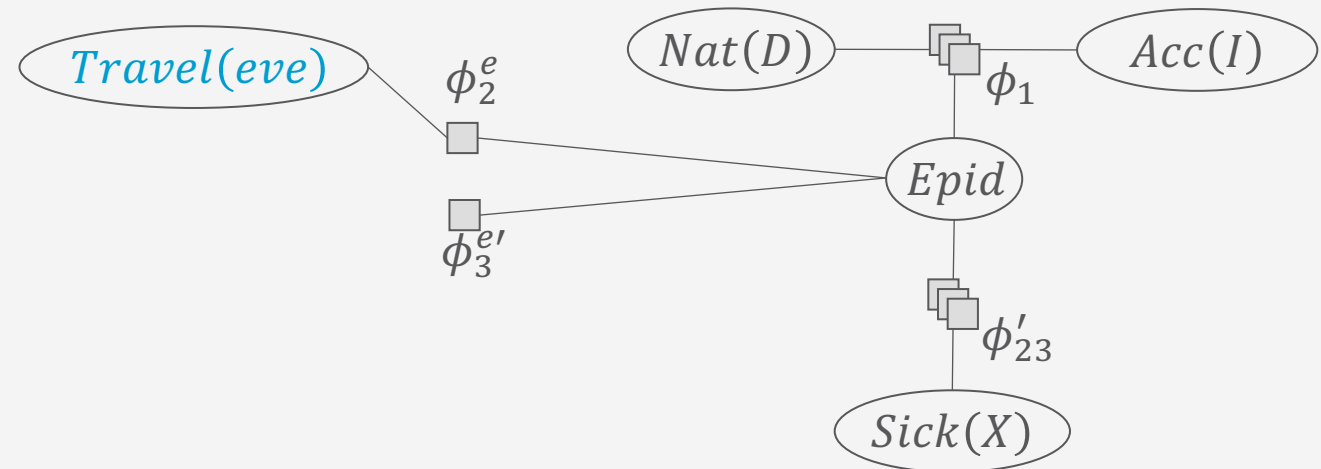


LVE: Example

- Eliminate all non-query terms
- PRVs fulfilling sum–out preconditions:

- $Sick(X)$
 - Yields $\phi''_2(Epid) \rightarrow \text{size: } 2$

Only one



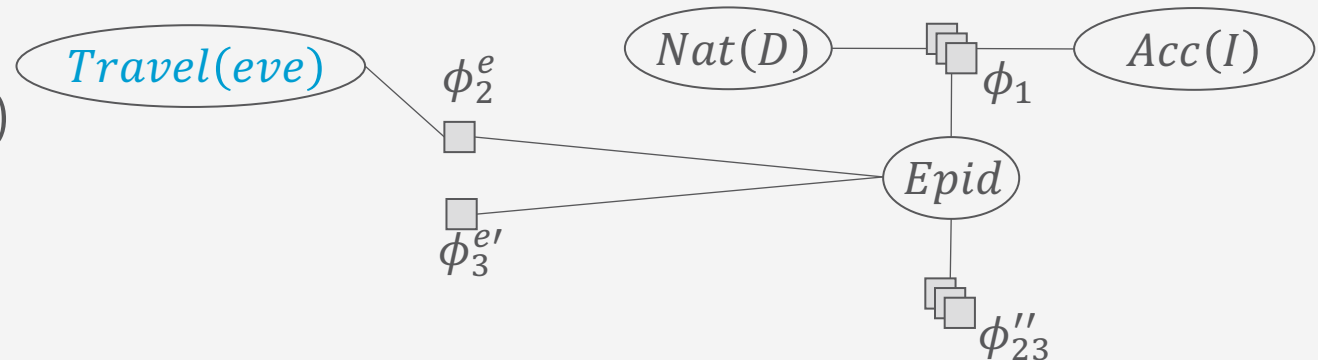
LVE: Example

- Eliminate all non-query terms
 - No PRVs fulfilling sum-out preconditions; others:

- Multiply $\phi_3^{e'}$ and ϕ_{23}''
 - Yields $\phi_{23}^{e''}(Epid) \rightarrow$ size: 2

Smallest size

- Multiply ϕ_2^e and $\phi_3^{e'}$
 - Yields $\phi_{23}'(Epid, Travel(eve)) \rightarrow$ size: $2^2 = 4$
- Count-convert $Nat(D)$
 - Yields $\phi_1^D(Epid, \#_D[Nat(D)], Acc(I)) \rightarrow$ size: $2 \cdot 3 \cdot 2 = 12$
- Count-convert $Man(W)$
 - Yields $\phi_1^I(Epid, Nat(D), \#_I[Acc(I)]) \rightarrow$ size: $2^2 \cdot 3 = 12$



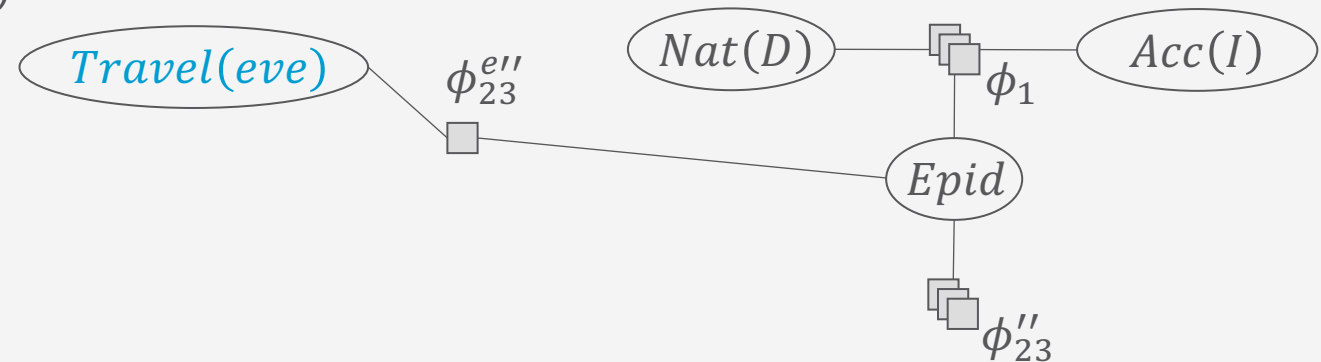
LVE: Example

- Eliminate all non-query terms
- No PRVs fulfilling sum-out preconditions; others:

- Multiply ϕ_2^e and $\phi_{23}^{e''}$
 - Yields $\phi_{23}^{e'}(Epid, Travel(eve)) \rightarrow \text{size: } 2^2 = 4$

Smallest size

- Count-convert $Nat(D)$
 - Yields $\phi_1^D(Epid, \#_D[Nat(D)], Acc(I)) \rightarrow \text{size: } 2 \cdot 3 \cdot 2 = 12$
- Count-convert $Man(W)$
 - Yields $\phi_1^I(Epid, Nat(D), \#_I[Acc(I)]) \rightarrow \text{size: } 2^2 \cdot 3 = 12$



LVE: Example

- Eliminate all non-query terms
 - No PRVs fulfilling sum–out preconditions; others:
 - Count-convert $Nat(D)$
 - Yields $\phi_1^D(Epid, \#_D[Nat(D)], Man(W)) \rightarrow \text{size: } 2 \cdot 3 \cdot 2 = 12$
 - Count-convert $Acc(I)$
 - Yields $\phi_1^I(Epid, Nat(D), \#_I[Acc(I)]) \rightarrow \text{size: } 2^2 \cdot 3 = 12$

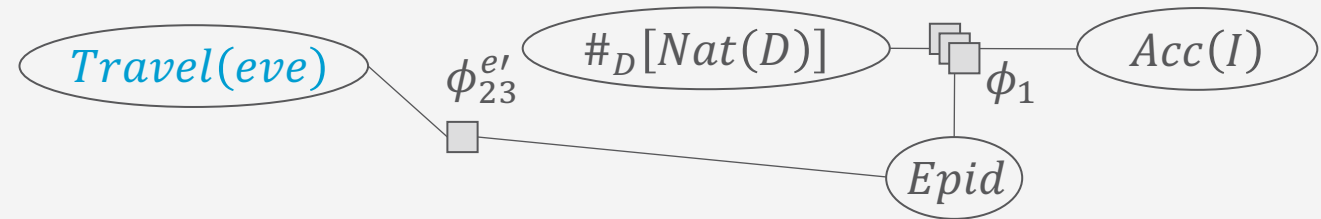


LVE: Example

- Eliminate all non-query terms
- PRVs fulfilling sum-out preconditions:

- $Acc(I)$
 - Yields $\phi'_1(Epid, \#_D[Nat(D)]) \rightarrow \text{size: } 2 \cdot 3 = 6$

Only one



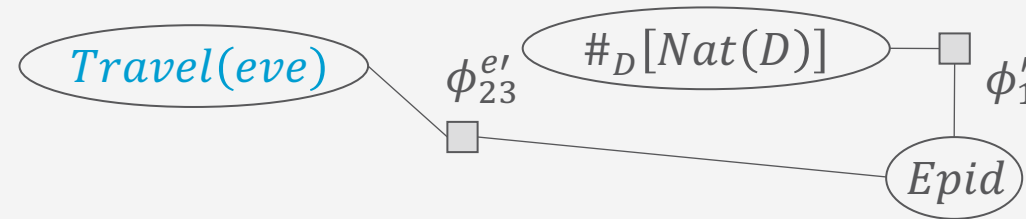
LVE: Example

- Eliminate all non-query terms
 - PRVs fulfilling sum–out preconditions:

- $\#_D[Nat(D)]$
 - Yields $\phi_1''(Epid) \rightarrow \text{size: } 2$

Only one

No uncounted logvars left
(basically standard VE + CRVs)



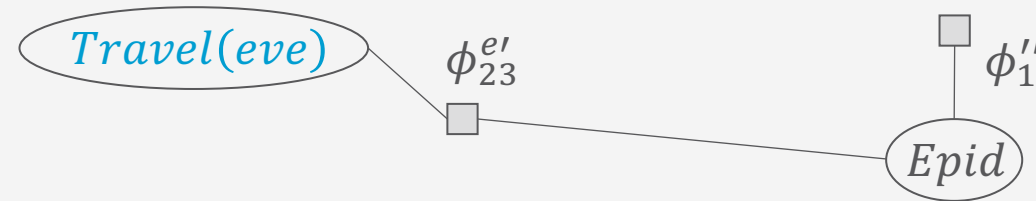
LVE: Example

- Eliminate all non-query terms
 - No PRVs fulfilling sum–out preconditions; others:

- Multiply ϕ_1'' and $\phi_{23}^{e'}$
 - Yields $\phi_{123}^{e''}(\text{Travel}(\text{eve}), \text{Epid}) \rightarrow \text{size: 4}$

Only one

Only propositional and ground
random variables left (standard VE)

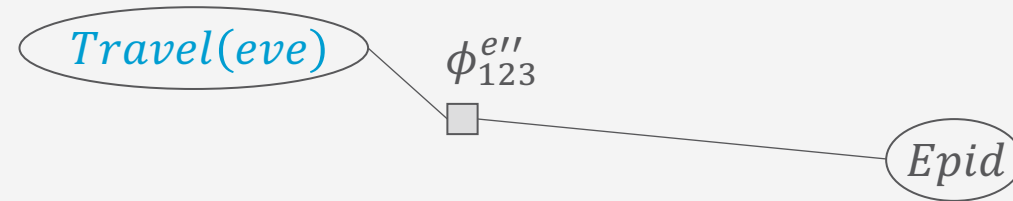


LVE: Example

- Eliminate all non-query terms
 - PRVs fulfilling sum–out preconditions:

- *Epid*
 - Yields $\phi(\text{Travel}(\text{eve})) \rightarrow \text{size: 2}$

Only one



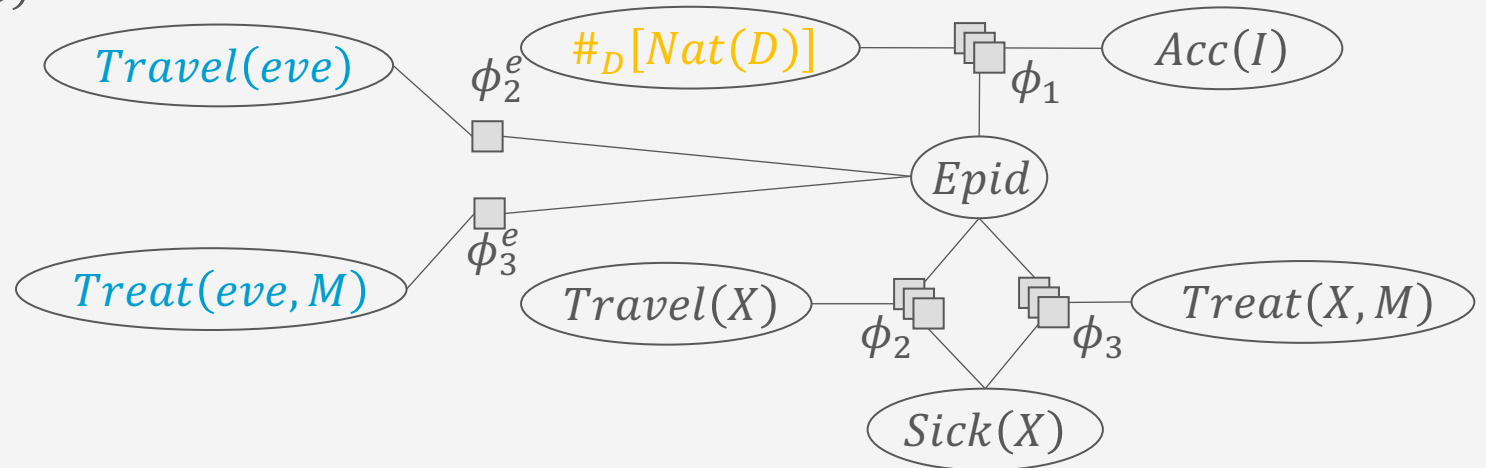
LVE: Example

- No non-query terms left
- Multiply all parfactors in G together
 - Only one parfactor $g = \phi(\textit{Travel}(\textit{eve}))$
- Normalise g
 - Yields $g' = \phi'(\textit{Travel}(\textit{eve}))$ containing the probability distribution over $\textit{Travel}(\textit{eve})$
- Return g'



$$P(\text{Travel}(\text{eve})|\text{sick}(\text{eve}))$$

$$= \frac{1}{Z} \sum_{e \in \text{ran}(\text{Epid})} \phi_2^e(\text{Travel}(\text{eve}), e) \sum_{h_n \in \text{ran}(\#_D[\text{Nat}(D)])} \text{Mul}(h_n) \left(\sum_{a \in \text{ran}(\text{Acc}(I))} \phi_1(e, h_n, a) \right)^{|\text{dom}(I)|} \\ \left(\sum_{s \in \text{ran}(\text{Sick}(X))} \left(\sum_{tt \in \text{ran}(\text{Treat}(X, M))} \phi_3(e, s, tt) \right)^{|\mathcal{D}(M)|} \sum_{t \in \text{ran}(\text{Travel}(X))} \phi_2(e, s, t) \right)^{|\text{dom}(X)|} \\ \sum_{te \in \text{ran}(\text{Treat}(\text{eve}, M))} \phi_3(e, te)$$



- After **shattering**, **absorption**, and the required **count conversion**

DPG Definition

- Assume that (sub)model G fulfils a *normal form** where
 - * possible to rewrite any model in polynomial time into normal form
 - Domains are either disjoint or identical
 - Logical variables share the same name if they refer to the same domain over different parfactors
 - Constraints are T
 - Formal definition by Taghipour includes inequality constraints
- ⇒ No further splitting operations necessary (split, expand, count-normalise)
- Decomposition into partial groundings of G by logical variable X with $\forall g \in G: X \in \text{lv}(g)$

$$DPG(G, X) = \bigcup_{g \in G} g\theta, \theta = \{X \rightarrow x\}$$

LVE for Parameterised Queries

At this point, G contains only $Q|_C$ terms but the logical variables in $Q|_C$ may still be uncounted; the next loop counts them if possible

Normalisation changes to account for histograms encoding multiple assignments

```

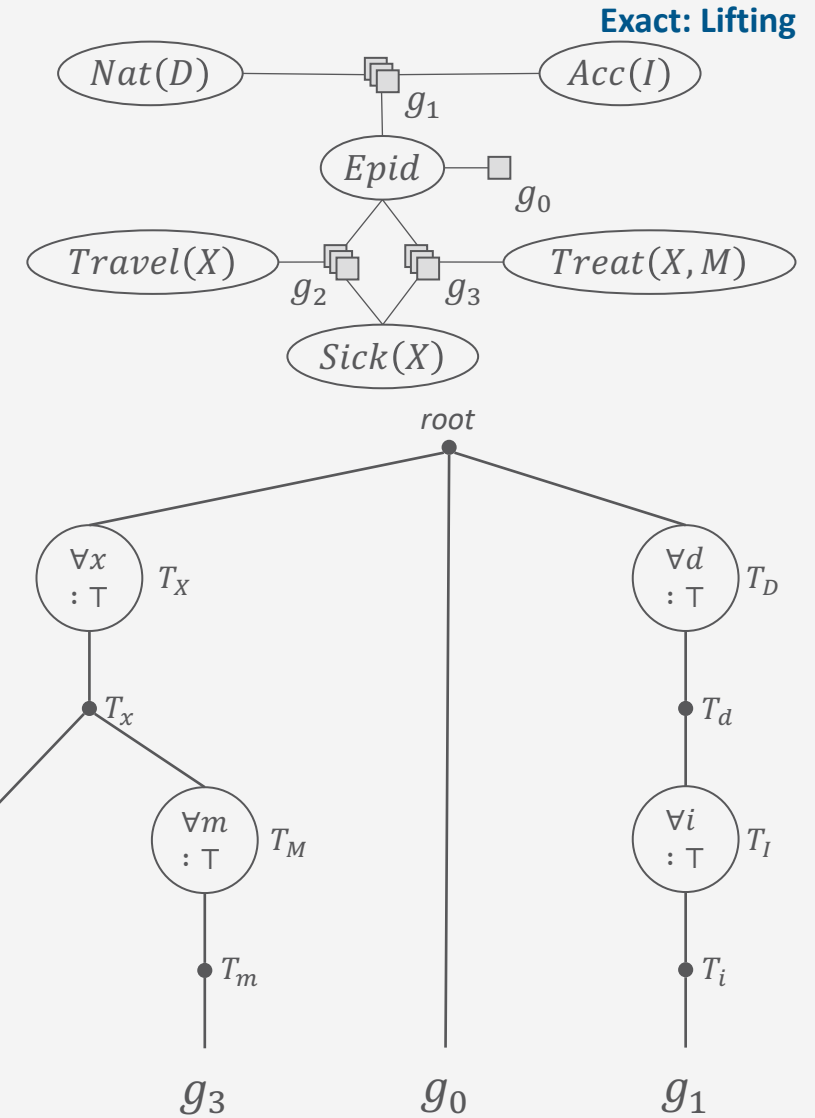
LVE( $G, Q|_C, \{g_e\}_{e=1}^m$ )
   $G \leftarrow$  Shatter  $G$  on  $Q|_C, \{g_e\}_{e=1}^m$ , and on itself
   $G \leftarrow$  Absorb  $\{g_e\}_{e=1}^m$  in  $G$ 
  while  $G$  contains non-query terms do
    if a PRV  $A$  fulfils the preconditions of sum-out then
       $G \leftarrow$  Apply sum-out to  $A$  in  $G$ 
    else
       $G \leftarrow$  Apply an enabling operator on some parfactors in  $G$ 
    while  $lv(G) \neq \emptyset$  do
      if  $\exists X \in lv(G)$  s.t.  $X$  is countable in  $g \in G$  then
         $G \leftarrow$  Apply count-convert to  $X$  in  $g$ 
      else
         $G \leftarrow$  Apply an enabling operator on some parfactors in  $G$ 
       $g \leftarrow$  Multiply all parfactors in  $G$  into one parfactor
       $g \leftarrow$  Normalise the potentials in  $g$ 
  return  $g$ 
  
```

Complexity

- Given liftable query over query terms Q
 - Class of query terms Q containing at most one constant for each logical variable in $lv(G)$ if ground or one set of constants if parameterised
- Assumption is that $q = |Q|$ is reasonably small
 - Especially if comparing r^q to $r^{w_g} \cdot n_{\#}^{r_{\#}w_{\#}}$
s.t. we can consider it outweighed by $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
- Liftable parameterised queries require only at most q additional count conversions, which are bounded by $O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$, and hopefully, $q \ll n_T$
- I.e., LVE complexity given a liftable model, a liftable query, and liftable evidence remains at
$$O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$$

FO Dtree Definition

- Let $DPG, VE, Leaf$ be the sets of all DPG, VE, leaf nodes each
- Then, an FO dtree T for a model G is given by $T = (V, E)$ where
 - $V = DPG \cup VE \cup Leaf$
 - $E = (DPG \times VE) \cup (VE \times DPG) \cup (VE \times VE) \cup (VE \times Leaf)$
 - VE can follow DPG / VE nodes, DPG / leaf can follow VE nodes
 - Each DPG node T_X has a child VE node T_x whose model G_x is a representative model of G_X with $G_x = G_X \theta, \theta = \{X \rightarrow x\}$
 - Each leaf with representative constant x in its parfactor descends from exactly one DPG node $T_X = (X, x, C)$
 - Each leaf descending from DPG node $T_X = (X, x, C)$ has representative constant x in its parfactor
 - Effect: At beginning of construction, one has to partition initial model G into one partition of parfactors containing only random variables and one partition of parfactors containing logical variables

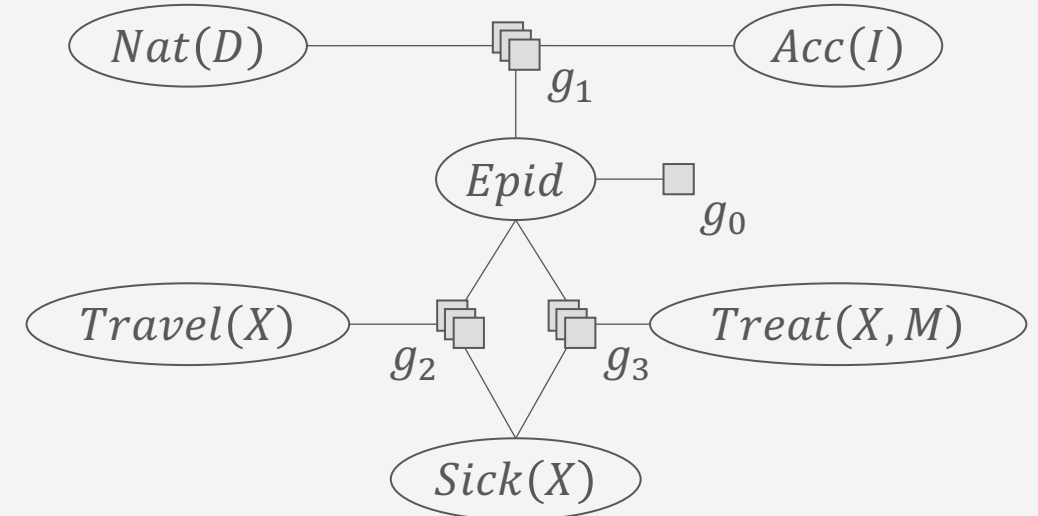


Problem: Many Queries

- Set of queries
 - $P(\text{Travel}(\text{eve}))$
 - $P(\text{Sick}(\text{bob}))$
 - $P(\text{Treat}(\text{eve}, m_1))$
 - $P(\text{Epid})$
 - $P(\text{Nat}(\text{flood}))$
 - $P(\text{Acc}(\text{chem}))$
 - Combinations of variables
- Under evidence
 - $\text{Sick}(X') = \text{true}$
 - $X' \in \{\text{alice}, \text{eve}\}$

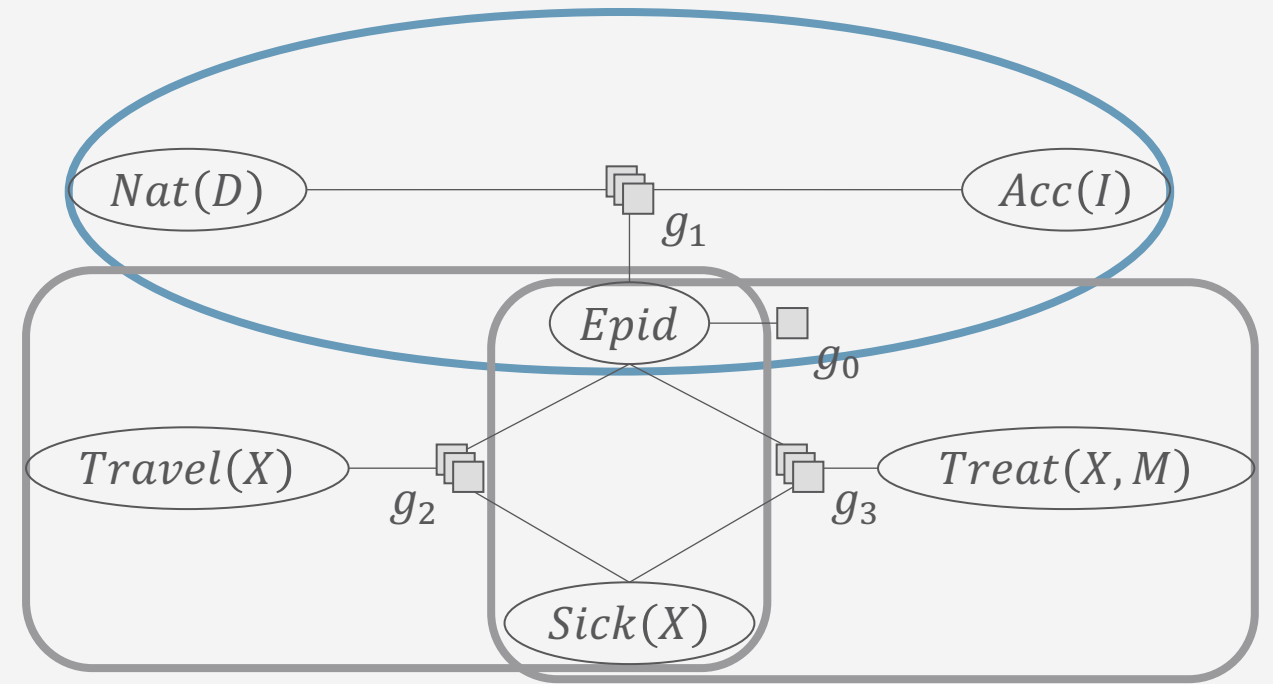
- LVE restarts with initial model for each query

Build a helper structure to precompute parts



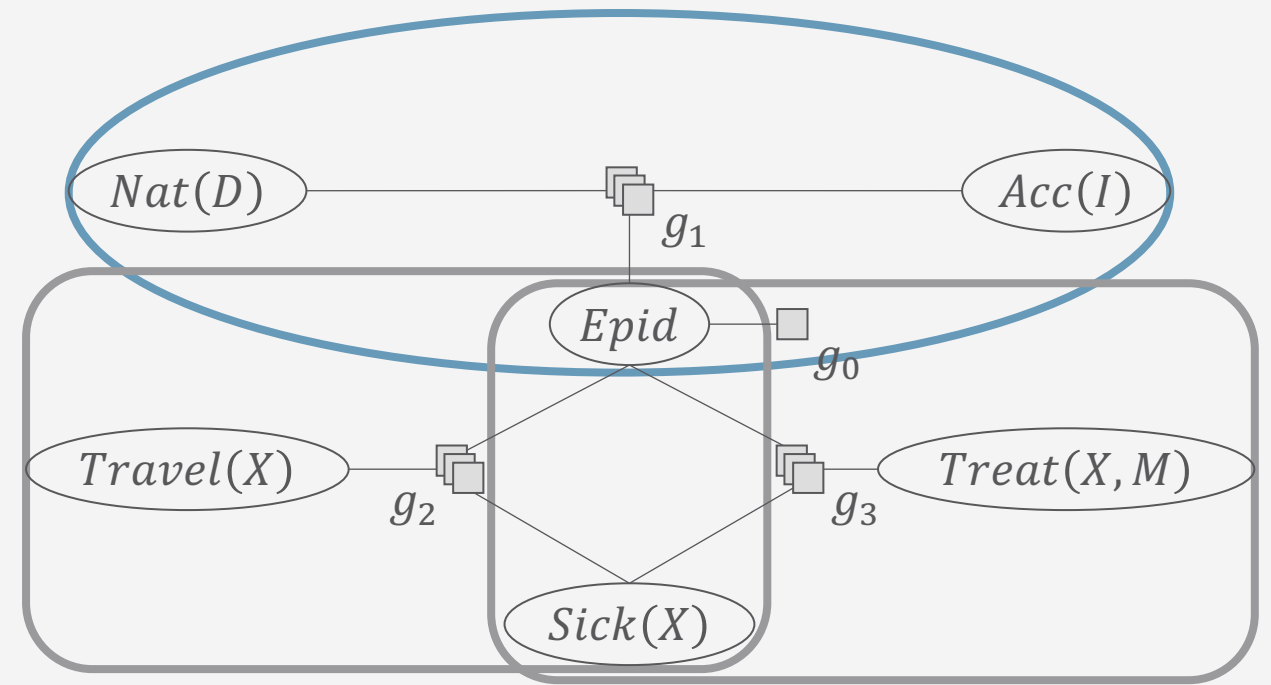
Clustering of Models

- Idea: Find subsets (clusters) of PRVs that are “enough” for certain queries
 - E.g.,
 - For queries about instances of *Nat(D), Acc(I), Epid*
 - *Nat(D), Acc(I), Epid* enough
 - For queries about instances of *Travel(X), Sick(X), Epid*
 - *Travel(X), Sick(X), Epid* enough
 - For queries about instances of *Treat(X, M), Sick(X), Epid*
 - *Treat(X, M), Sick(X), Epid* enough



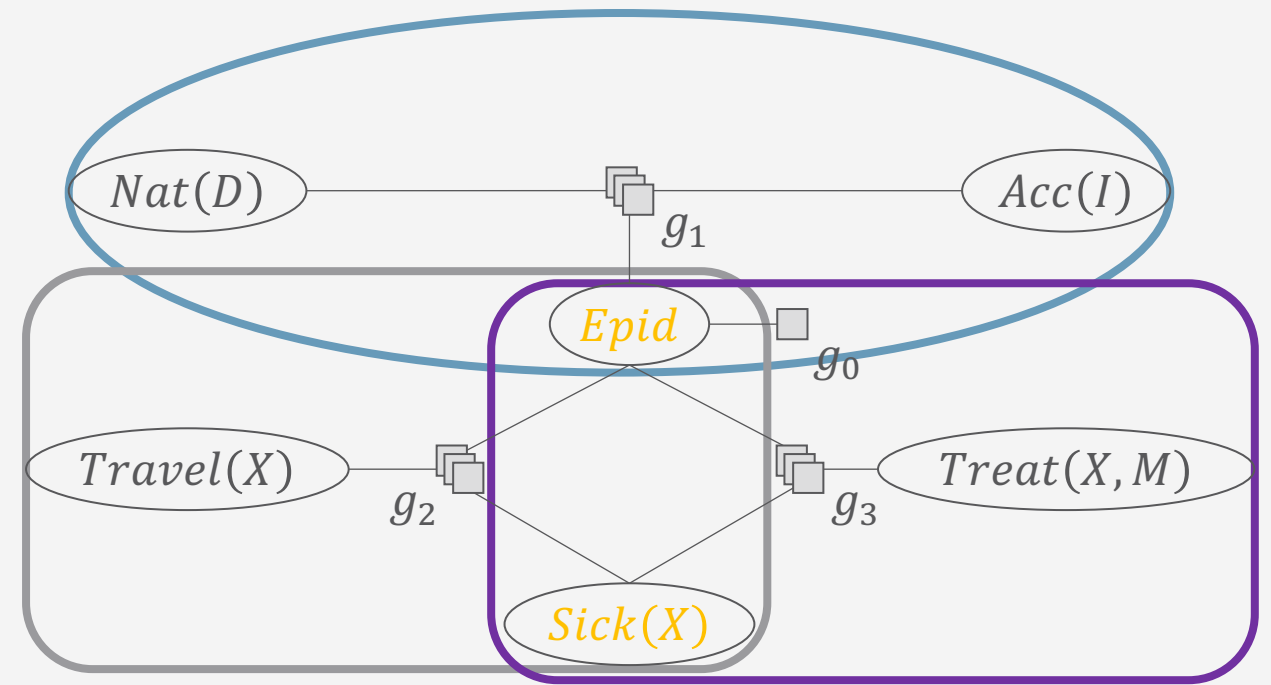
Clustering of Models

- But: If only parfactors used that contain the PRVs of a cluster, information stored in all other parfactors ignored
- E.g.,
 - $Nat(D), Acc(I), Epid: g_1$
→ misses g_2, g_3
 - $Travel(X), Sick(X), Epid: g_2$
→ misses g_1, g_3
 - $Treat(X, M), Sick(X), Epid: g_3$
→ misses g_1, g_2
 - Whatever we do with $g_0...$
- Only correct if clusters are *independent* from each other
 - How can we achieve independence?



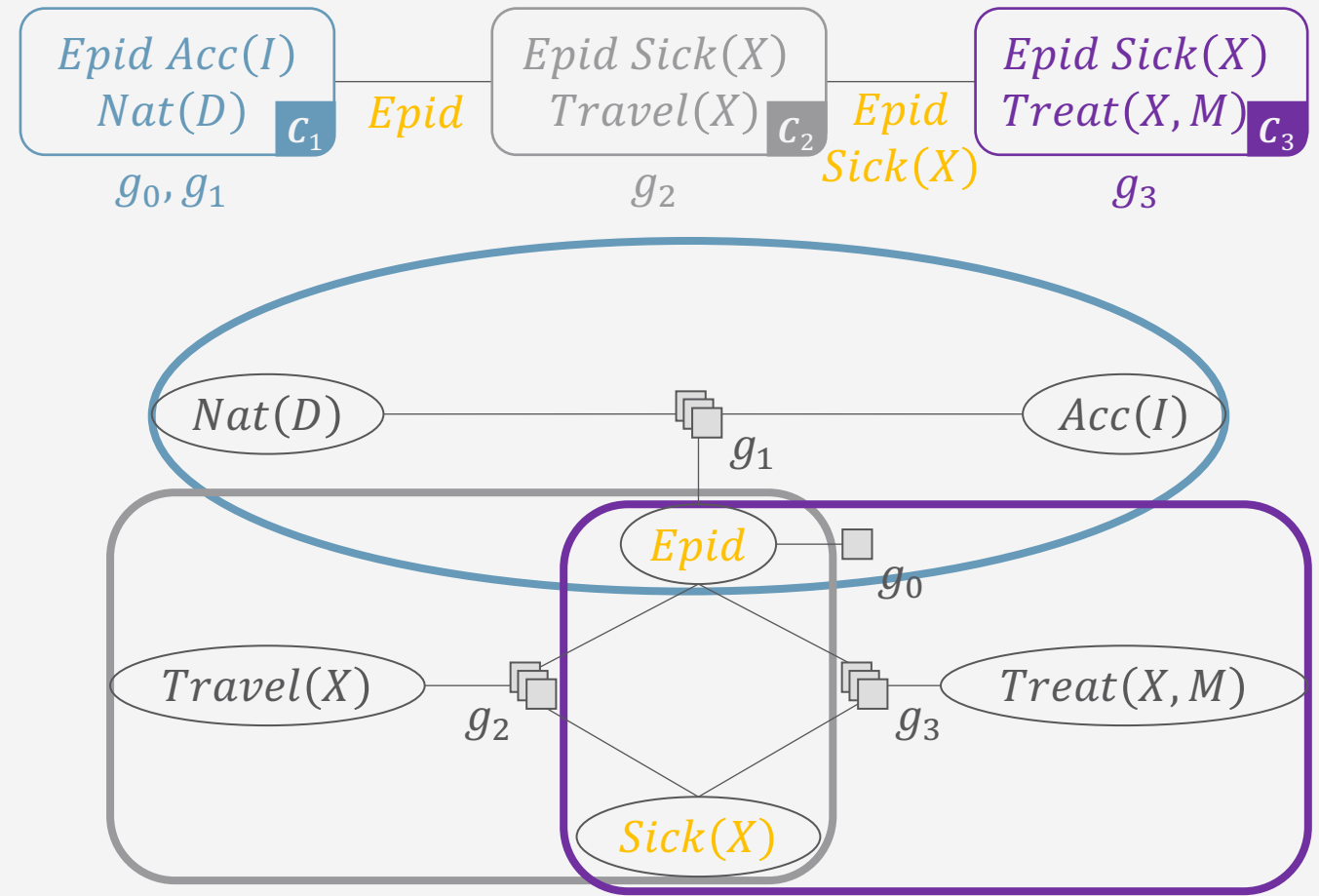
Clustering of Models

- Factorised models encode independences:
 - Any two subsets of variables are conditionally independent given a *separating* subset S
 - Separating* subset S : All paths from one subset to the other run through S
 - Also known as *global Markov property*
 - E.g.,
 - $Nat(D), Acc(I), Epid: g_1$
→ independent of the rest given $Epid$
 - $Travel(X), Sick(X), Epid: g_2$
→ independent of the rest given $Epid, Sick(X)$
 - $Treat(X, M), Sick(X), Epid: g_3$
→ independent of the rest given $Epid, Sick(X)$



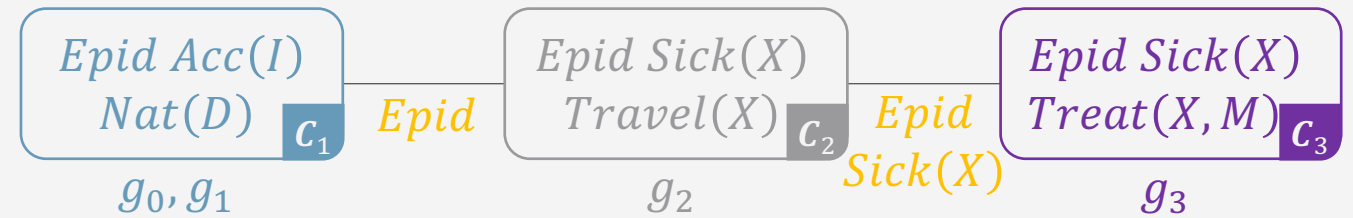
Clustering of Models

- Put clusters and their separators into a graph structure where
 - Nodes are clusters with parfactors assigned containing the cluster PRVs (*local model*)
 - Edges are labelled with the *separator* (separating subset) between neighbouring nodes
 - If two nodes contain the same PRV, every node on the path between the two nodes contain the PRV (*running intersection property*)



Clustering of Models

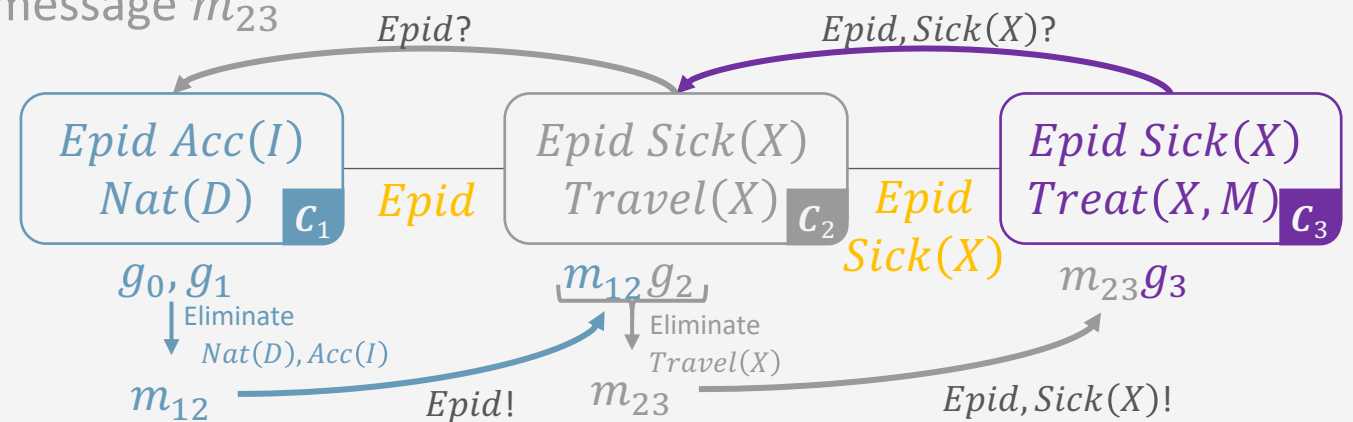
- Next: Make clusters actually independent of each other
- Each cluster i asks its neighbours $j \in nbs(i)$ for information about the separator $\mathcal{S}_{ij}$ between them
 - Other clusters have to collect all the information from the model that lies behind the separator on its part, eliminate the non-separator PRVs from that information using LVE, and send the result in a message m_{ji} , i.e., a set of parfactors, back
- Having the information on the separators to all neighbours makes a cluster independent from its neighbours and therefore all other parts of the model
 - Ensures that each cluster of PRVs has all model information needed available for query answering on instances of its cluster PRVs



Clustering of Models

- Next: Make clusters actually independent of each other
- E.g., $\mathcal{C}_3: g_3 \rightarrow$ independent of the rest given $Epid, Sick(X)$
 - Asks neighbour $\mathcal{C}_2$ for information on $Epid, Sick(X)$
 - $\mathcal{C}_2$ asks neighbour $\mathcal{C}_1$ for information on $Epid$
 - $\mathcal{C}_1$ sends information on $Epid$ in message m_{12}
 - Eliminates $Nat(D), Acc(I)$ from g_0, g_1 for m_{12}
 - $\mathcal{C}_2$ sends info. on $Epid, Sick(X)$ to $\mathcal{C}_3$ in message m_{23}
 - Eliminates $Travel(X)$ from g_2, m_{12} for m_{23}
 - With m_{23} , $\mathcal{C}_3$ is independent of its neighbour $\mathcal{C}_2$ and therefore also of $\mathcal{C}_1$
 - As $\mathcal{C}_2$ is independent given m_{12} of $\mathcal{C}_1$

The same has to be done for $\mathcal{C}_2$ and $\mathcal{C}_1$



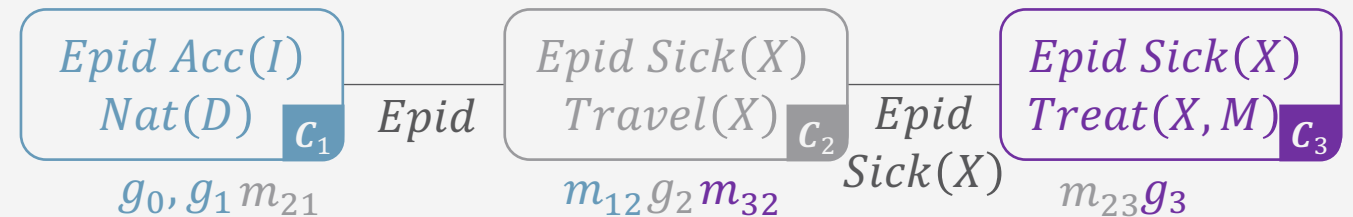
Clustering of Models

- With each cluster i independent of the rest, each i can answer queries about instances of its PRVs based on its local model and the messages received
 - Query terms: grounded instances or parameterised versions of its PRVs
 - Conjunctive queries if terms only concern the cluster PRVs
 - E.g., $\mathcal{C}_3: g_3 \rightarrow$ independent of the rest given $Epid, Sick(X)$
 - Based on g_3 and m_{23} , $\mathcal{C}_3$ can answer queries about $Epid, Sick(X), Treat(X, M)$ such as

$$P(Sick(X)),$$

$$P(Treat(eve, m_2)),$$

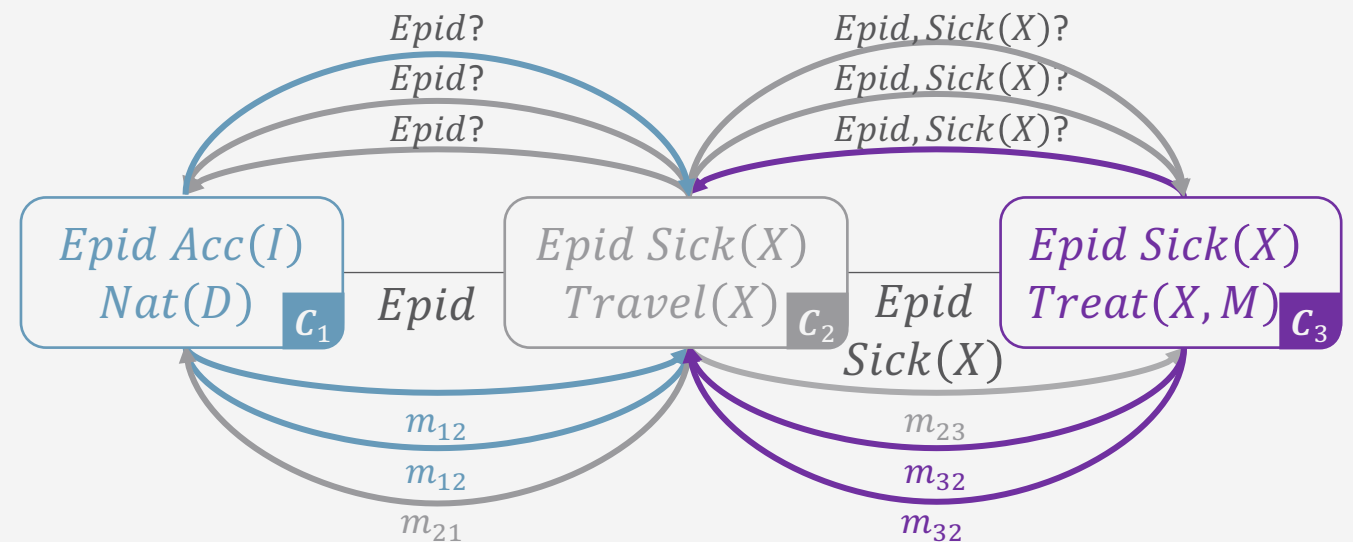
$$P(Epid, Sick(alice))$$
 - Cannot answer any queries about $Nat(D), Acc(I), Travel(X)$ but $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, can



Clustering of Models

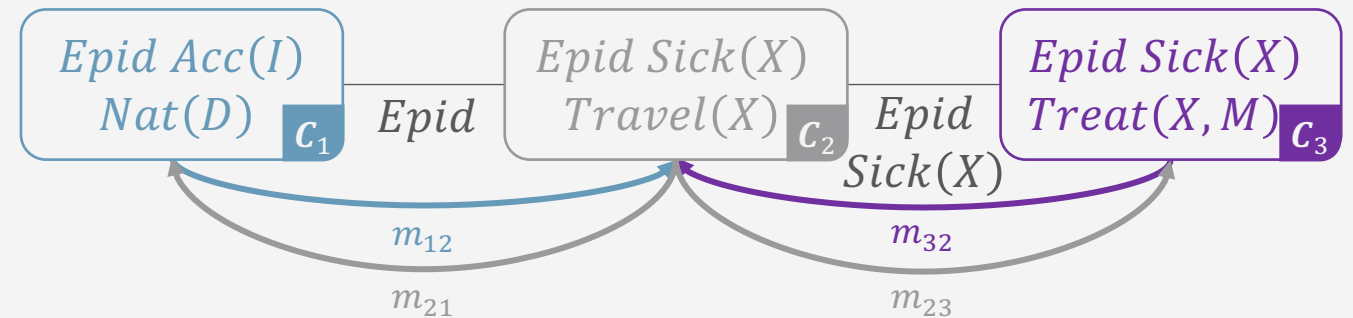
- Problem left: If each cluster asks for information on separators, some messages are sent multiple times
 - E.g.,
 - C_3 asks C_2 , which asks C_1
 - Messages calculated and sent: m_{12}, m_{23}
 - C_2 asks C_1 and C_3
 - Messages calculated and sent: m_{12}, m_{32}
 - C_1 asks C_2 , which asks C_3
 - Messages calculated and sent: m_{32}, m_{21}

Organise in way that messages are calculated only once



Clustering of Models

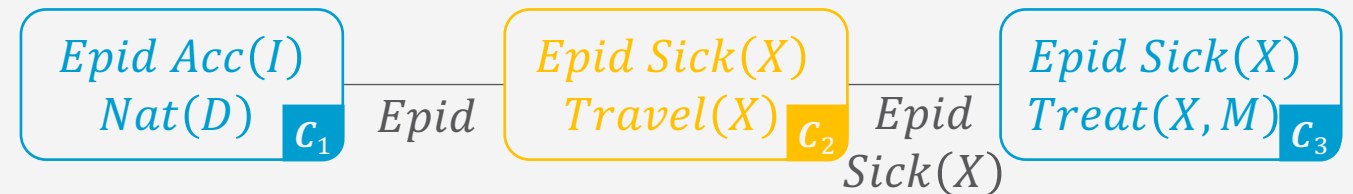
- Use dynamic programming to organise the order of asking or rather sending information in messages:
 - If a node i has received all information from neighbours but one, j , node i sends a message with its information on the separator S_{ij} to j
 - If a node i has received all messages, then it sends messages to all neighbours j that have not received a message yet
- When computing the message, i takes into consideration
 - its local model G_i as well as
 - the messages m_{ki} received from all other neighbours k but the receiving neighbour j



Clustering of Models

Graph structured called (first-order) junction tree
and algorithm called (lifted) junction tree algorithm

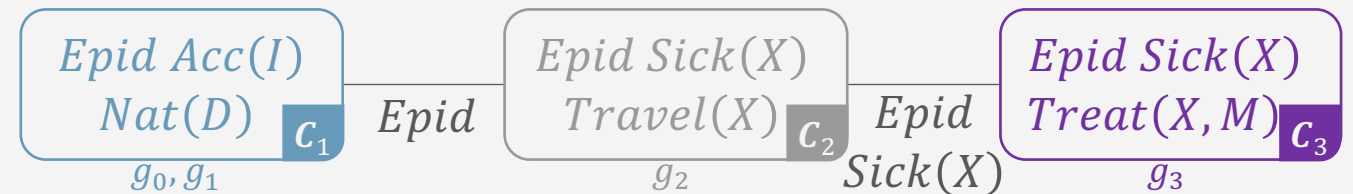
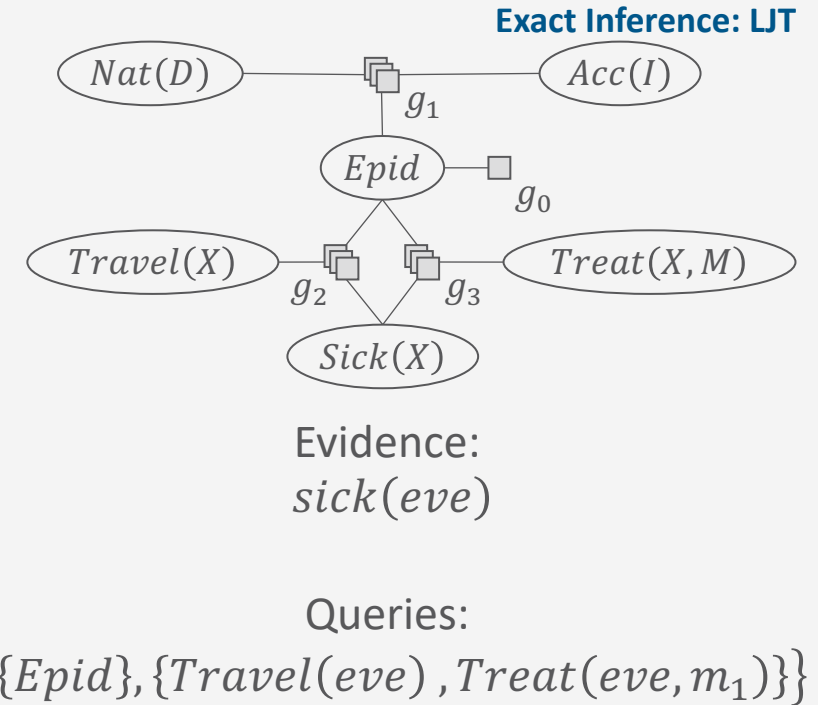
- Observations:
 - If a node i has received all information from neighbours but one, j , node i sends a message with its information on the separator S_{ij} to j
 - Trivially true at leaf nodes (*periphery*), can start sending immediately to its only neighbour (in *parallel*!)
 - From periphery inbound, new nodes trigger this first condition
 - If a node i has received all messages, then it sends messages to all neighbours j that have not yet received a message
 - As messages are sent further inwards, a first node at the centre will have received all messages and will start sending messages outbound, leading to new nodes triggering this second condition



Two passes from periphery to centre and back suffice to distribute all information and make the clusters independent from each other*

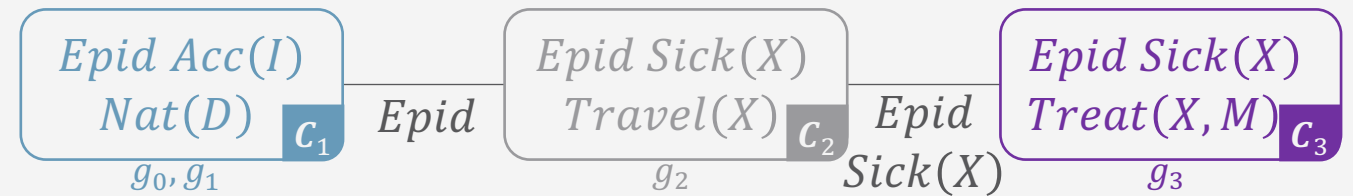
Lifted Junction Tree Algorithm (LJT)

- Inputs
 - Model G
 - Evidence e as evidence parfactors
 - Set of query terms $\{Q_i\}_{i=1}^m$
 - Queries on G : $P(Q_i \mid e)$, $i \in \{1, \dots, m\}$
- LJT consists of four steps
 - Build FO jtree J for model G
 - Enter evidence e in J
 - Pass messages in J
 - Answer queries $\{Q_i\}_{i=1}^m$



First-order Jtree (FO Jtree)

- As seen on the earlier slides
 - Acyclic graph
 - Nodes contain PRVs, which form clusters
 - Edges are based on the separators between the clusters
 - Nodes have parfactors assigned
- Next slides:
 - Formal definition
 - Construction
 - Get an *acyclic* structure with valid *separators* and each parfactor of a model assigned to a *local model*



Parameterised Clusters

- Node of an FO jtree: Set of PRVs called parameterised cluster (**parcluster**)
- Let $\mathbf{X}$ be a set of logical variables, $\mathbf{A}$ a set of PRVs with $lv(\mathbf{A}) \subseteq \mathbf{X}$, and $(\mathcal{X}, C_{\mathcal{X}})$ a constraint on $\mathbf{X}$
- Then, a parcluster $\mathbf{C}$ is given by

$$\forall \mathbf{x} \in C_{\mathcal{X}} : \mathbf{A}_{|(\mathbf{x}, C_{\mathcal{X}})}$$

- $\mathbf{A}_{|(\mathbf{x}, C_{\mathcal{X}})}$ for short
 - Again, $(\mathcal{X}, C_{\mathcal{X}})$ can be omitted if T constraint encoded
- Depicted as a round shape containing $\mathbf{A}$ or just $\mathbf{A}$
 - Again, constraint usually not depicted
- E.g., parcluster $\mathbf{C}_2$

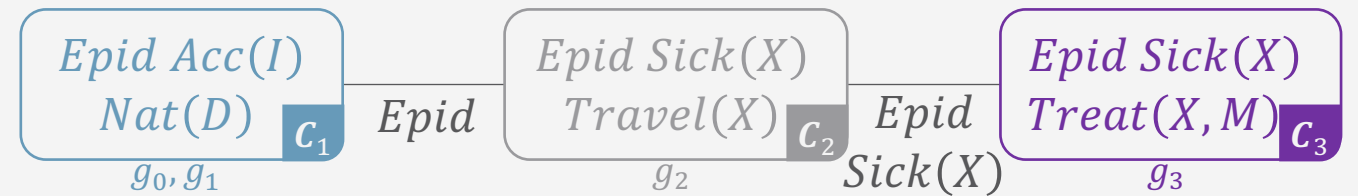
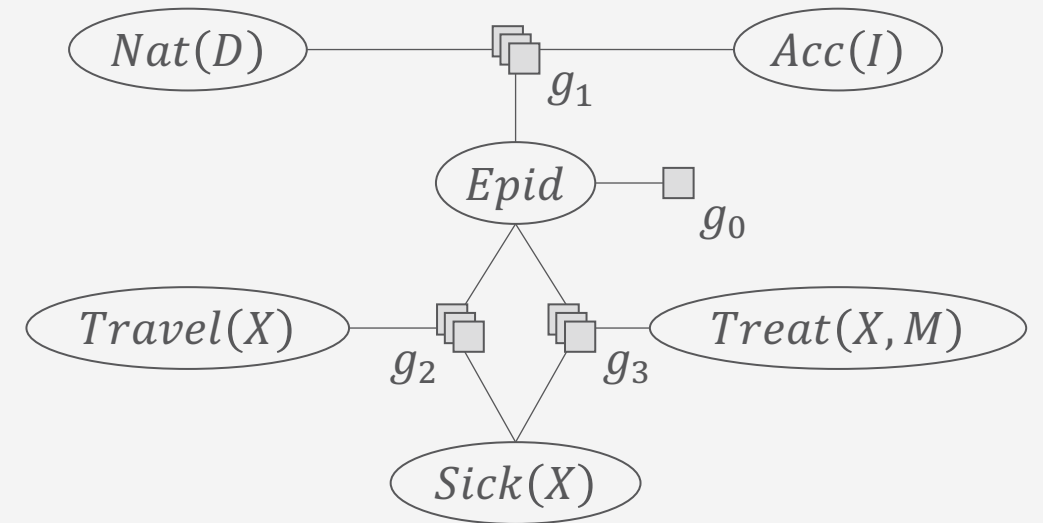
$$\begin{aligned} \forall x \in dom(X) : \{Epid, Sick(x), Travel(x)\}_{|(x, dom(X))} \\ = \{Epid, Sick(X), Travel(X)\}_{|(X, dom(X))} \\ = \{Epid, Sick(X), Travel(X)\} \end{aligned}$$

Epid Sick(X)
Travel(X)

Epid Sick(X)
Travel(X) $\mathbf{C}_2$

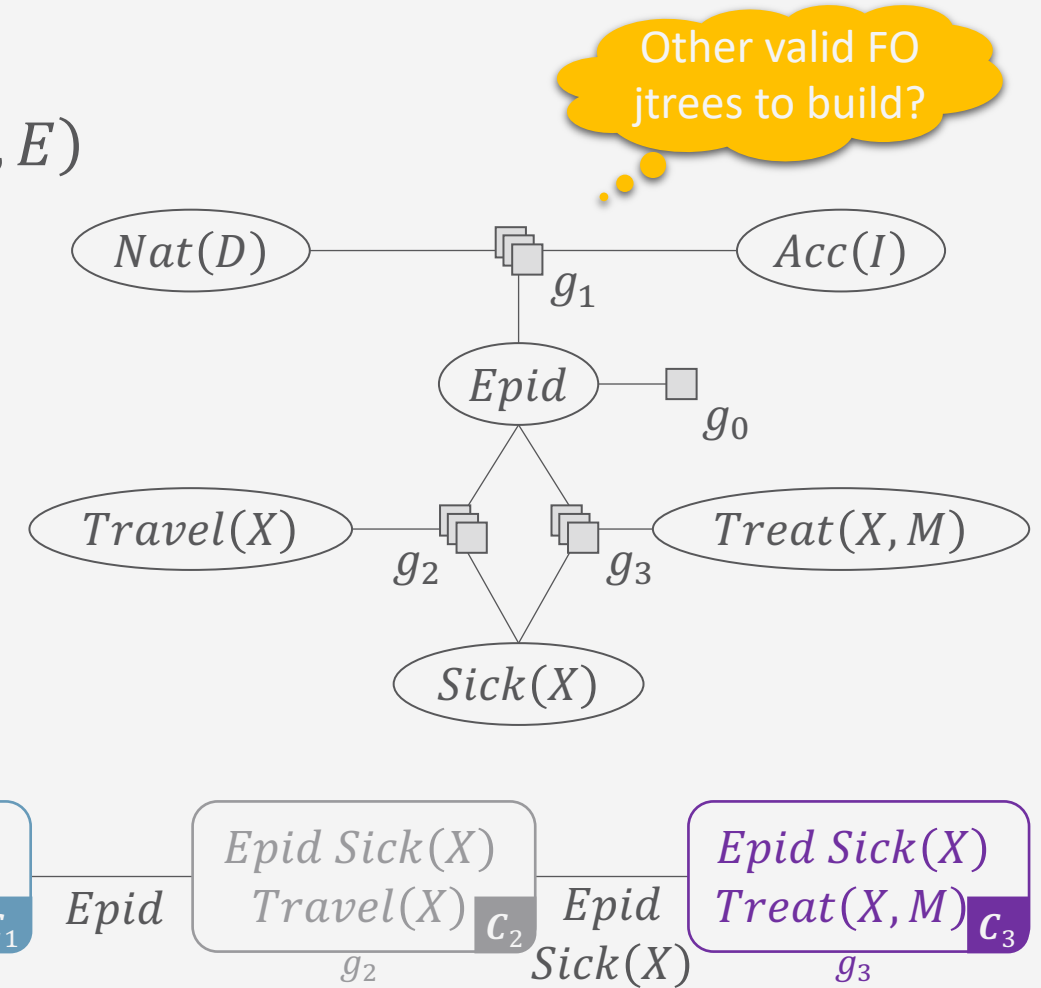
FO Jtree

- An FO jtree J for a model G is a cycle-free graph (V, E)
 - Set of nodes $V \subseteq 2^{rv(G)}$
 - I.e., nodes are sets of PRVs (parclusters)
 - $2^{rv(G)}$ denotes the power set of $rv(G)$
 - Set of edges $E \subseteq \{\{i, j\} \mid i, j \in V, i \neq j\}$
 - Has to be cycle free, which includes no self-loops
 - E.g., as depicted on the left
 - But at this point in the definition, could be any subsets of PRVs



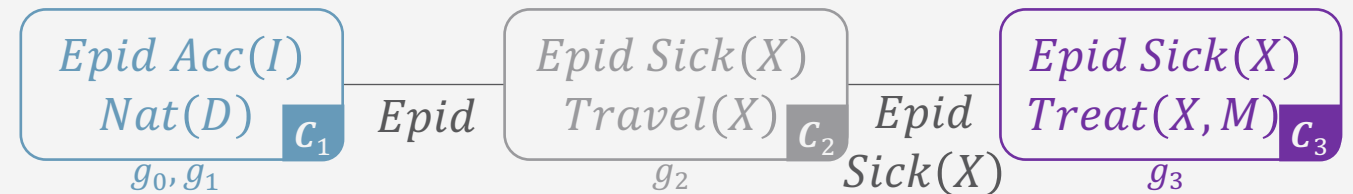
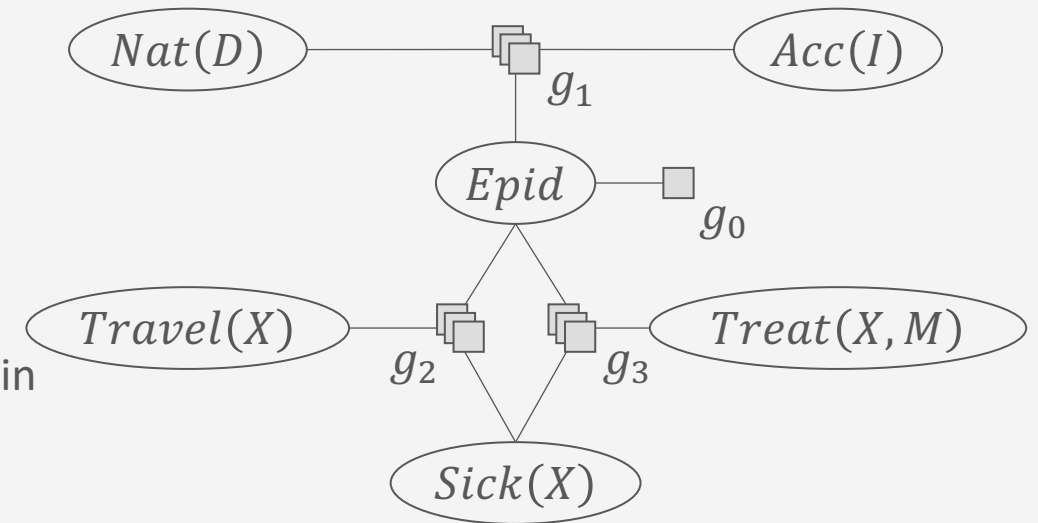
FO Jtree

- An FO jtree J for a model G is a cycle-free graph (V, E)
 - Has to satisfy three properties:
 - $\forall \mathcal{C} \in V : \mathcal{C} \subseteq rv(G)$
 - Every parcluster consists of PRVs from G
 - $\forall g \in G : \exists \mathcal{C} \in V : rv(g) \subseteq \mathcal{C}$
 - Arguments of every parfactor in G occur in a parcluster
 - If $\exists A \in rv(G) : A \in \mathcal{C}_i \wedge A \in \mathcal{C}_j$ with $\mathcal{C}_i, \mathcal{C}_j \in V$, then $\forall \mathcal{C}_k \in V$ on the path between $\mathcal{C}_i, \mathcal{C}_j : A \in \mathcal{C}_k$ (running intersection property)
 - If a PRV occurs in two parclusters, it also occurs in every parcluster on the path between them
 - E.g., as depicted on the left



FO Jtree

- An FO jtree J for a model G is a cycle-free graph (V, E)
 - Is **minimal** if by removing a PRV from a parcluster, the FO jtree ceases to be an FO jtree
 - I.e., no longer fulfils at least one property
 - E.g., depicted on the left
 - Cannot remove any PRV from any parcluster
 - Otherwise, a parfactor would no longer have its arguments in one parcluster



FO Jtree

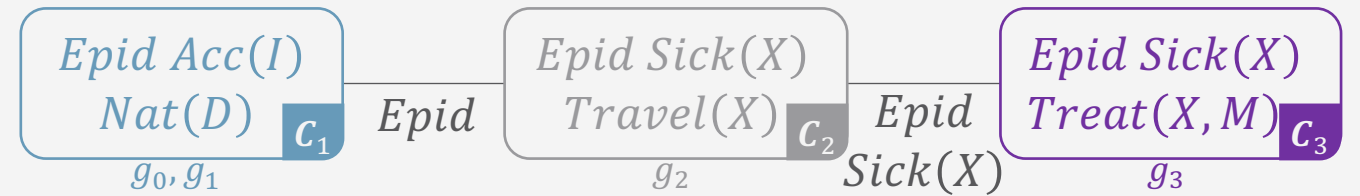
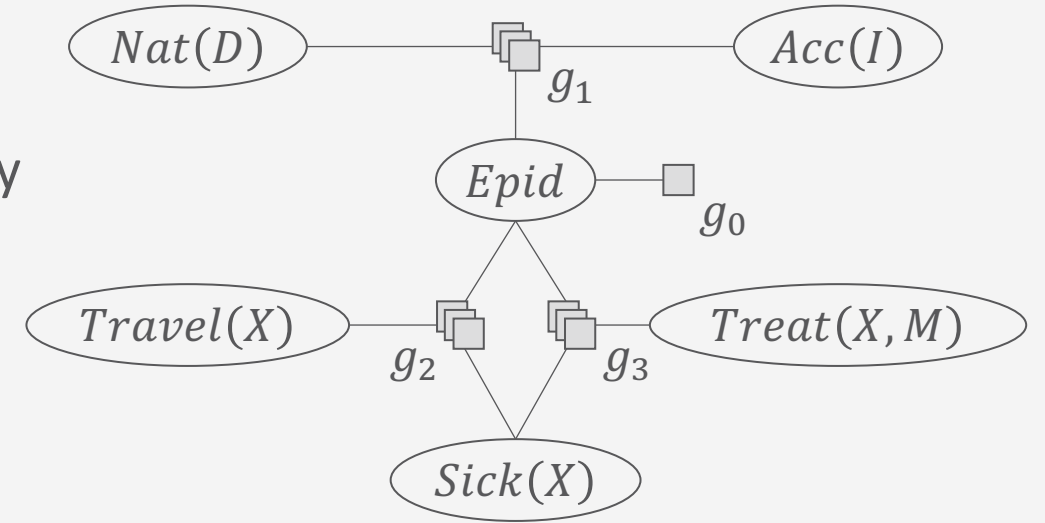
- An FO jtree J for a model G is a cycle-free graph (V, E)
 - Set S_{ij} called **separator** of edge $\{i, j\} \in E$, defined by

$$S_{ij} = C_i \cap C_j$$
 - Term $nbs(i)$ refers to the neighbours of C_i , defined by

$$nbs(i) = \{j \mid \{i, j\} \in E\}$$
 - Each C_i has a **local model** G_i and

$$\forall g \in G_i : rv(g) \subseteq C_i$$
 - Local models G_i partition G , i.e.,

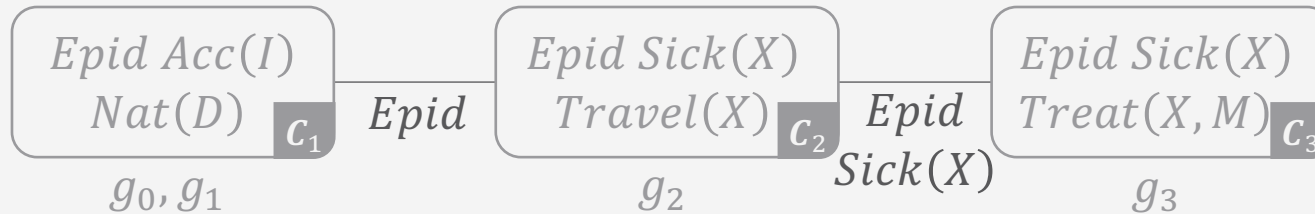
$$G = \bigcup_{i \in V} G_i, \forall i, j \in V, i \neq j : G_i \cap G_j = \emptyset, G_i \neq \emptyset$$



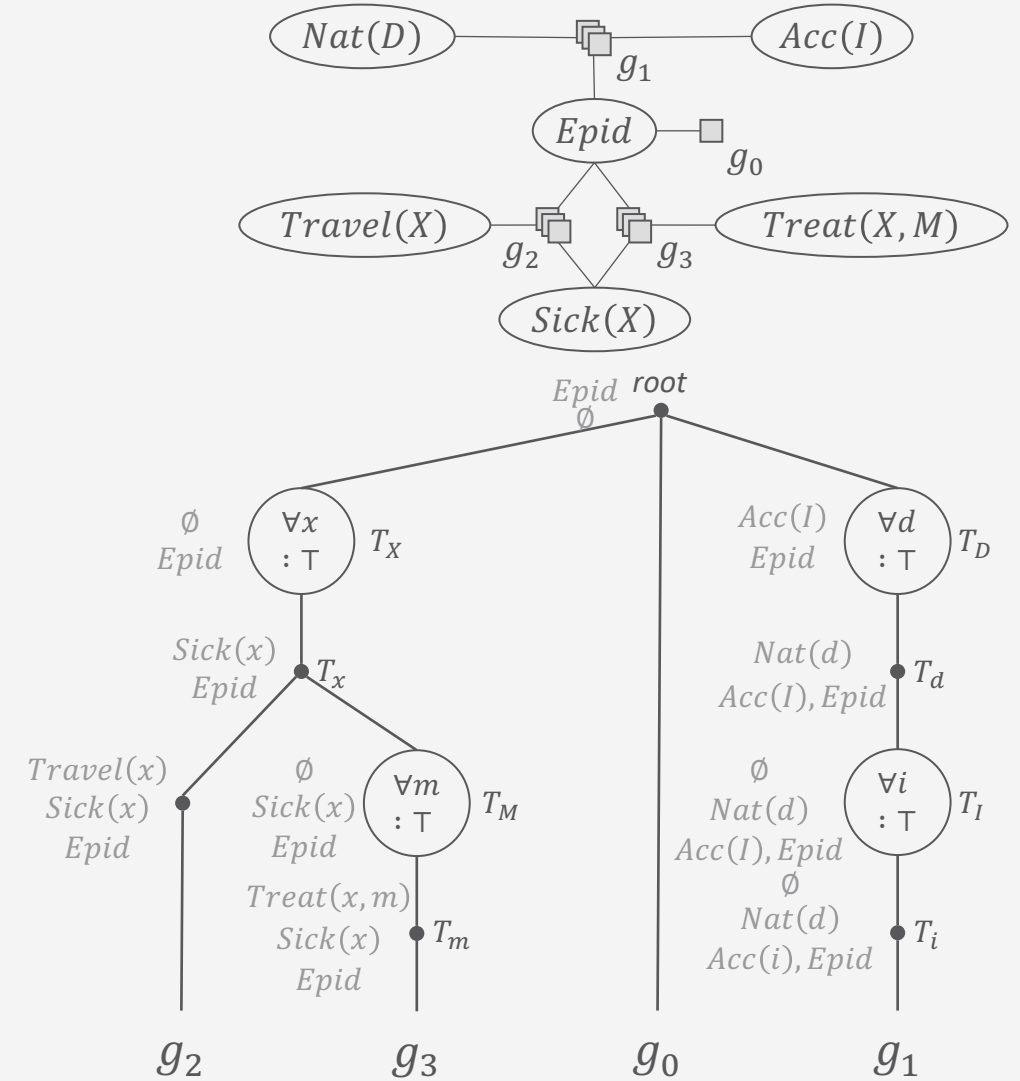
Construction

- Where do we get the FO jtree from s.t. the jtree
 - is acyclic
 - fulfils the three FO jtree properties
 - has the model parfactors automatically assigned to fitting parclusters?

→ Clusters of an FO dtree
 + undirected dtree edges
 + minimisation
 = FO jtree

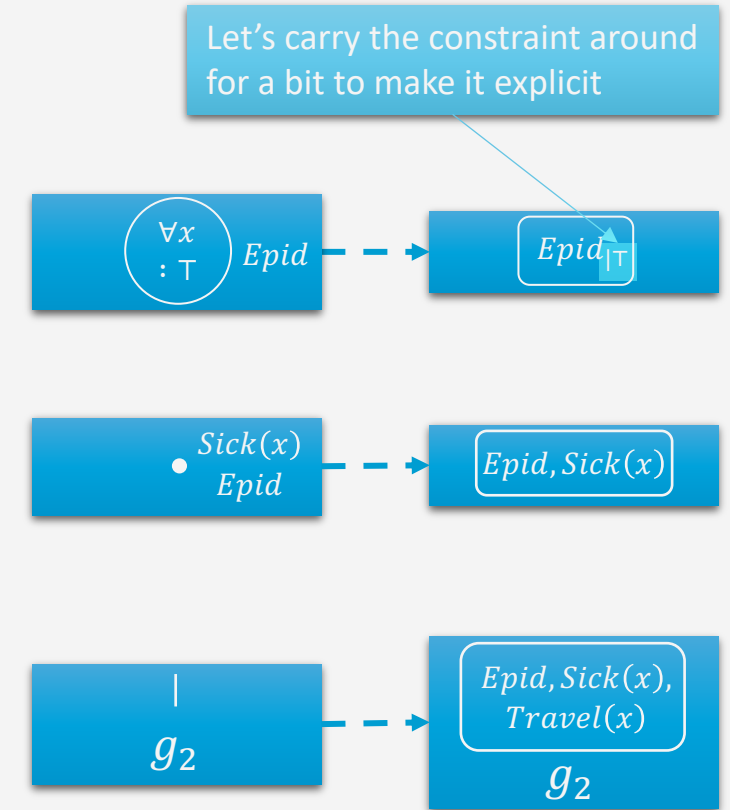


Exact Inference: LJT



Clusters → Parclusters

- Given an FO dtree T for a model G with clusters for each node
- Given a cluster $\{A_1, \dots, A_n\}$ of a DPG node (X, x, C)
 - Resulting parcluster $\mathcal{C}_j = \{A_1, \dots, A_n\}_{|C}$
 - Local model $G_j = \emptyset$
- Given a cluster $\{A_1, \dots, A_n\}$ of a VE node
 - Resulting parcluster $\mathcal{C}_j = \{A_1, \dots, A_n\}_{|T}$
 - Local model $G_j = \emptyset$
- Given a cluster $\{A_1, \dots, A_n\}$ from a leaf node with parfactor g_i
 - Resulting parcluster $\mathcal{C}_j = \{A_1, \dots, A_n\}_{|T}$
 - Local model $G_j = \{g_i\}$

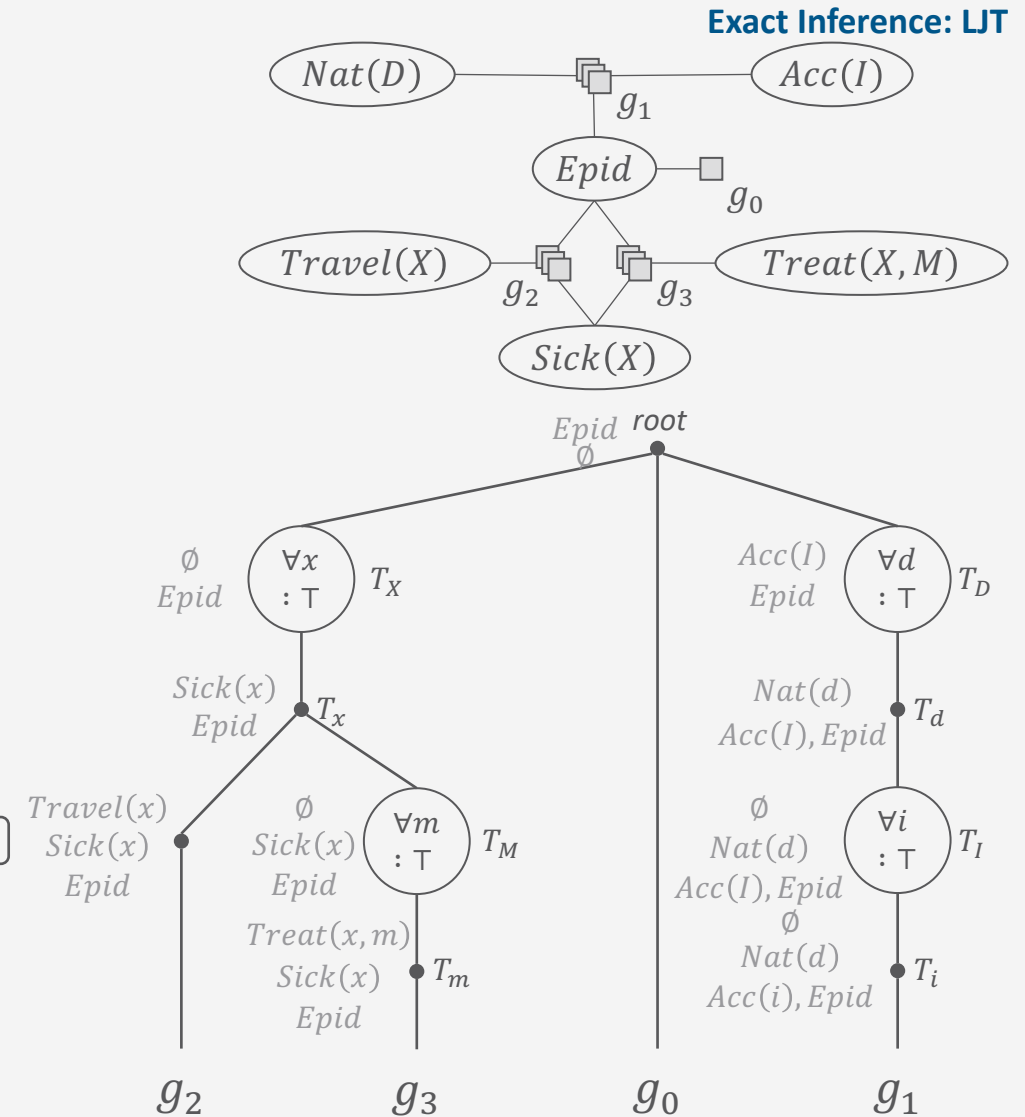
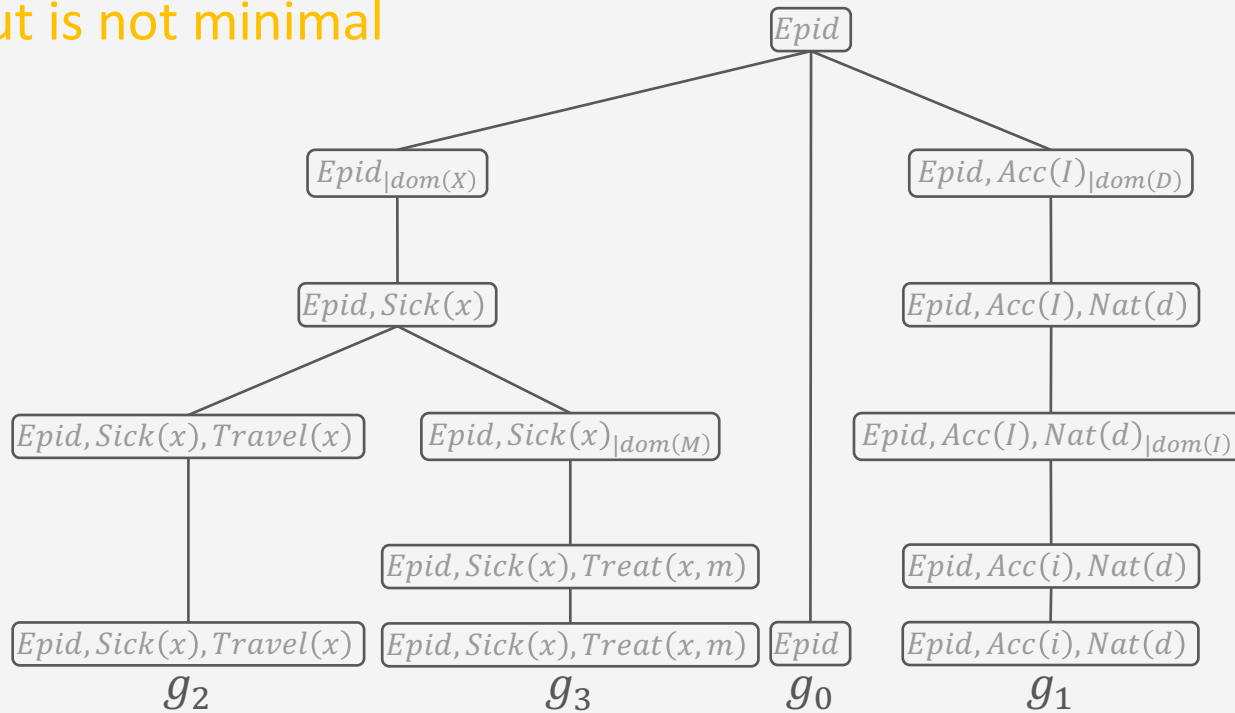


FO Dtree $\rightarrow$ FO Jtree

- Forming an FO jtree J from an FO dtree T of a model G
- Nodes of J
 - Parclusters resulting from clusters of T as shown on previous slide
 - Each parcluster has a source node in T
- Edges of J
 - Add an edge between two parclusters whenever there is an edge between the source nodes of the two parclusters in T

FO Dtree $\rightarrow$ FO Jtree

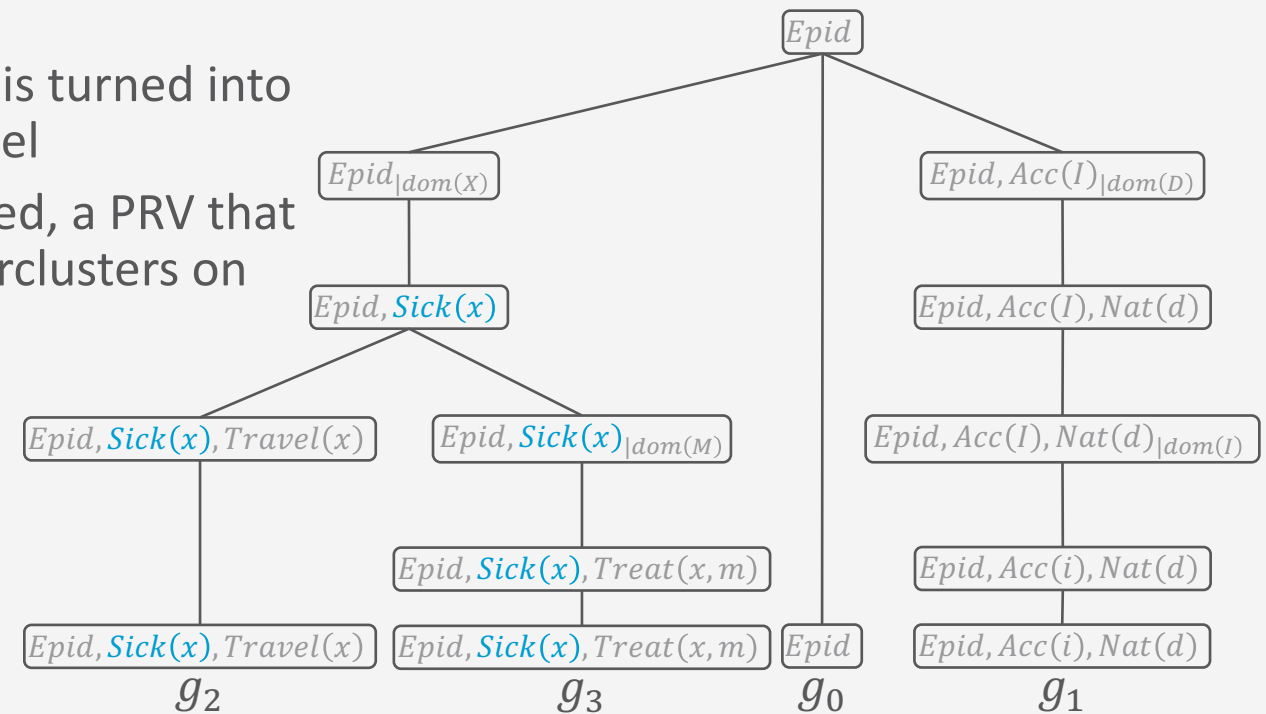
- Result after transformation
 - Fulfils the three jtree properties
 - But is not minimal



FO Dtree $\rightarrow$ FO Jtree

- Transformation result fulfils the three jtree properties
 - Properties hold by construction of the FO dtree
 - Parclusters can only contain model PRVs
 - Each parfactor occurs at a dtree leaf, which is turned into a parcluster with the parfactor as local model
 - Based on how cutset & context are calculated, a PRV that occurs in two parclusters will occur in all parclusters on the path between them*
 - E.g., *Sick(X)*

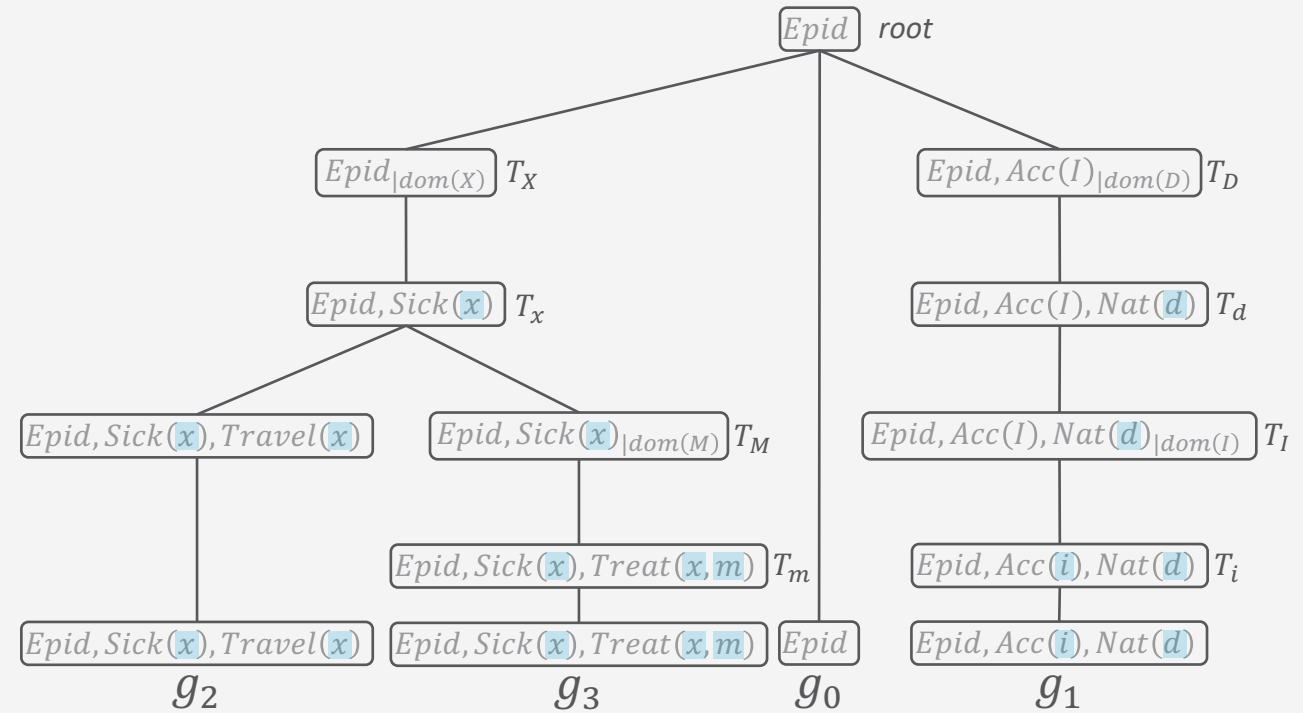
- $\forall C \in V : C \subseteq rv(G)$
- $\forall g \in G : \exists C \in V : rv(g) \subseteq C$
- If $\exists A \in rv(G) : A \in C_i \wedge A \in C_j$ with $C_i, C_j \in V$, then $\forall C_k \in V$ on the path between $C_i, C_j : A \in C_k$



FO Dtree $\rightarrow$ FO Jtree

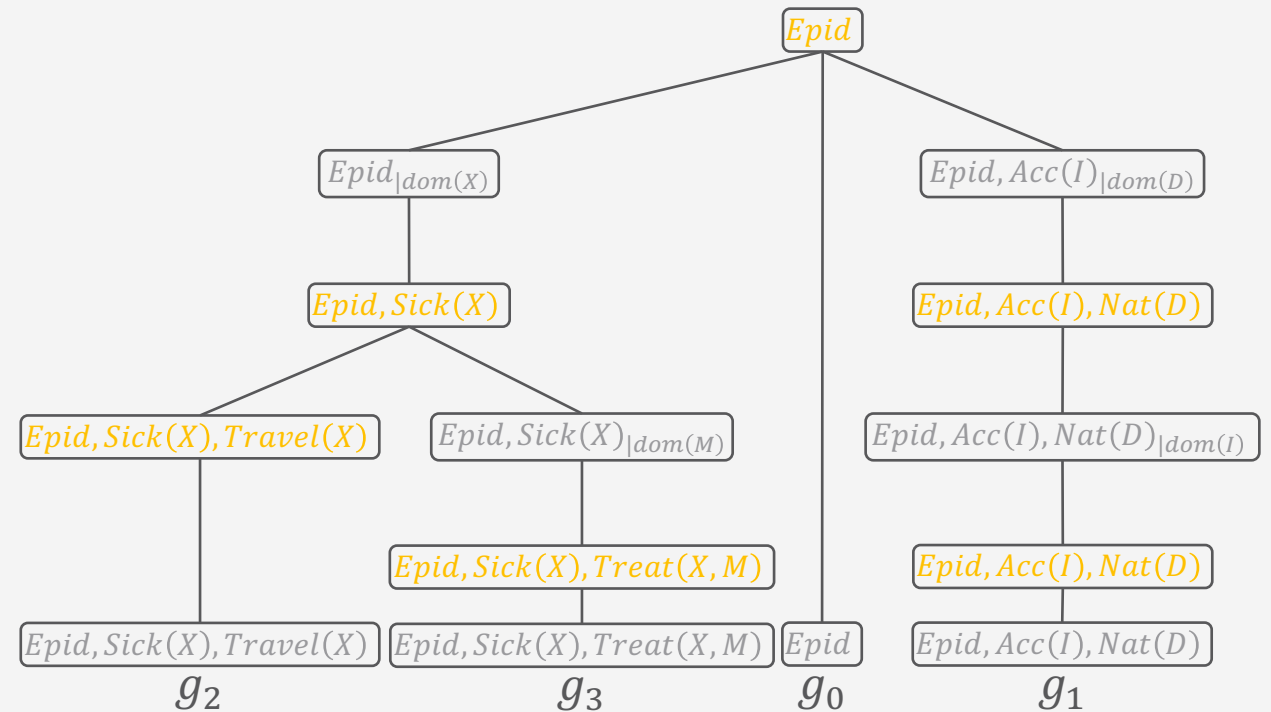
- But: Parclusters may contain a logical variable X or its representative x
- For each source DPG node T_X
 - Apply the inverse substitution θ^{-1} to the one applied during FO dtree construction to all parclusters that come from descendants of T_X :

$$\begin{aligned} &\theta^{-1} \\ &= \{X \rightarrow x\}^{-1} \\ &= \{x \rightarrow X\} \end{aligned}$$



FO Dtree $\rightarrow$ FO Jtree

- Result after transformation **not minimal**
 - Can remove complete parclusters and still have an FO jtree
 - Even if we keep parclusters that carry constraint information that we would otherwise lose
 - E.g.,
 - Parclusters **marked**
- Observation
 - Parclusters are subsets of other parclusters
 - Use for minimisation



Minimisation

- Merge parclusters $\mathcal{C}_i, \mathcal{C}_j$ with local models G_i, G_j iff $gr(\mathcal{C}_i) \subseteq gr(\mathcal{C}_j) \vee gr(\mathcal{C}_j) \subseteq gr(\mathcal{C}_i)$
 - Assuming T constraints and same names for logical variables that reference the same domain (from normal form of FO dtree), then the following suffices:

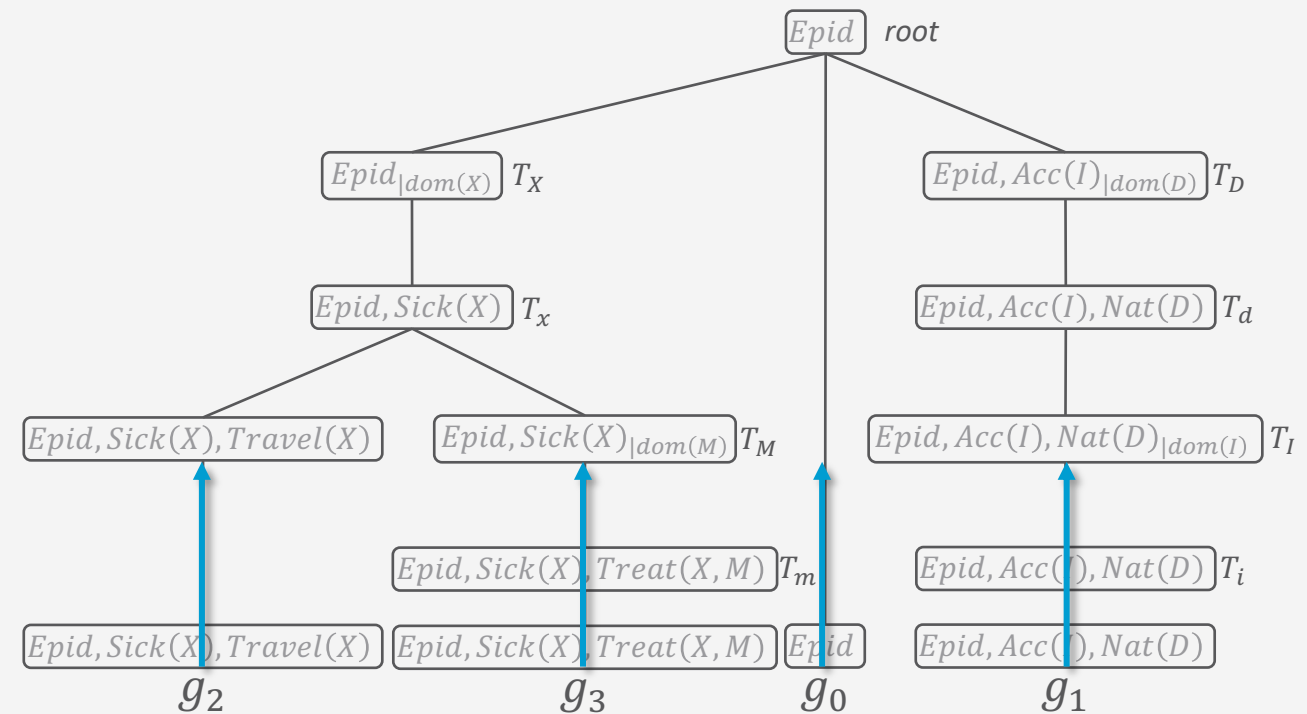
$$\mathcal{C}_i \subseteq \mathcal{C}_j \vee \mathcal{C}_j \subseteq \mathcal{C}_i$$

- Checking on a PRV and logical variable level instead of a grounded level
- Merging parclusters $\mathcal{C}_i, \mathcal{C}_j$ into parcluster $\mathcal{C}_k$
 - $\mathcal{C}_k = \mathcal{C}_i \cup \mathcal{C}_j$
 - $G_k = G_i \cup G_j$
 - Changes in FO jtree (V, E)
 - $V = V \setminus \{\mathcal{C}_i, \mathcal{C}_j\} \cup \mathcal{C}_k$
 - $E = E \setminus \{\{i, l\} \mid l \in nbs(i)\} \setminus \{\{j, l\} \mid l \in nbs(j)\} \cup \{\{k, l\} \mid l \in nbs(i) \vee l \in nbs(j), l \neq i, l \neq j\}$

Reassigns all neighbours
of $\mathcal{C}_i, \mathcal{C}_j$ to $\mathcal{C}_k$

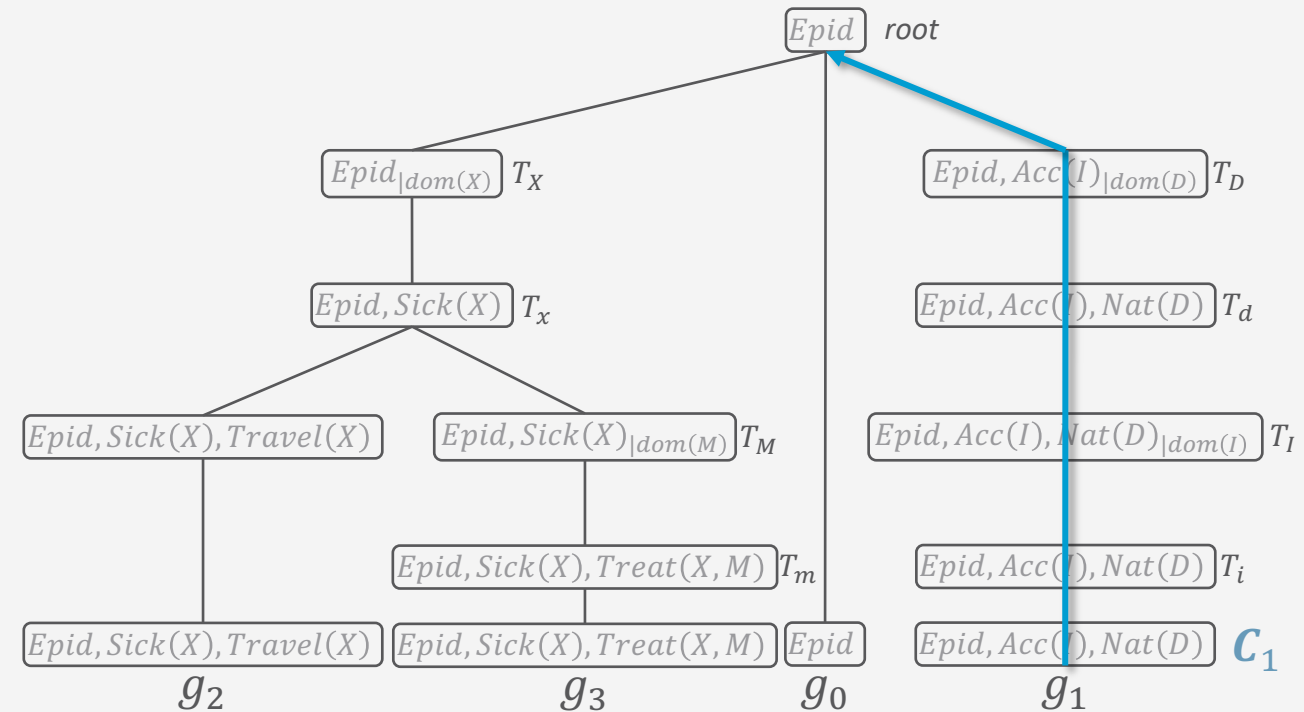
Minimisation

- Possible merging strategy
 - Start at leaves and merge **inbound**
 - Until no further merging possible
 - I.e., no parcluster a subset of another
- After merging, the resulting FO jtree is minimal
- E.g.,
 - Start at leaves with
 - local model $\{g_0\}$
 - local model $\{g_1\}$
 - local model $\{g_2\}$
 - local model $\{g_3\}$



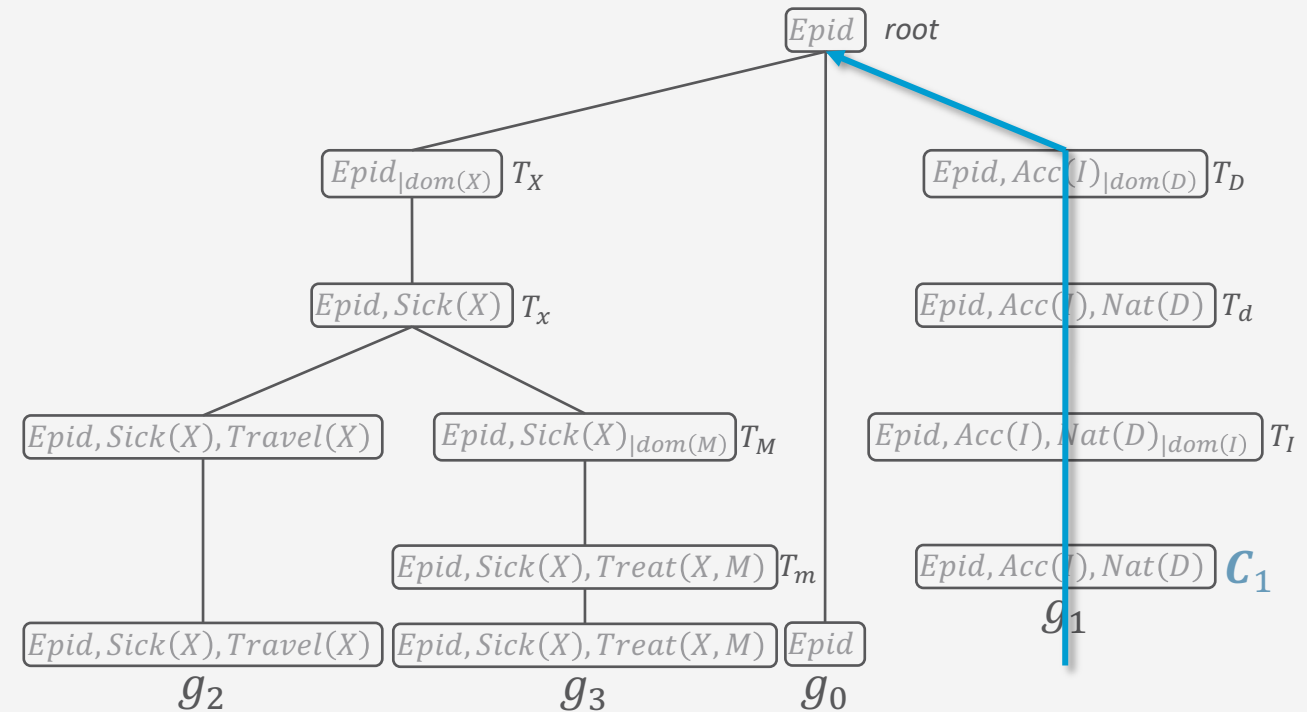
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it C_1
 - Merge **inbound**
 - C_1 and T_i parcluster identical
→ merge (call result C_1 again)



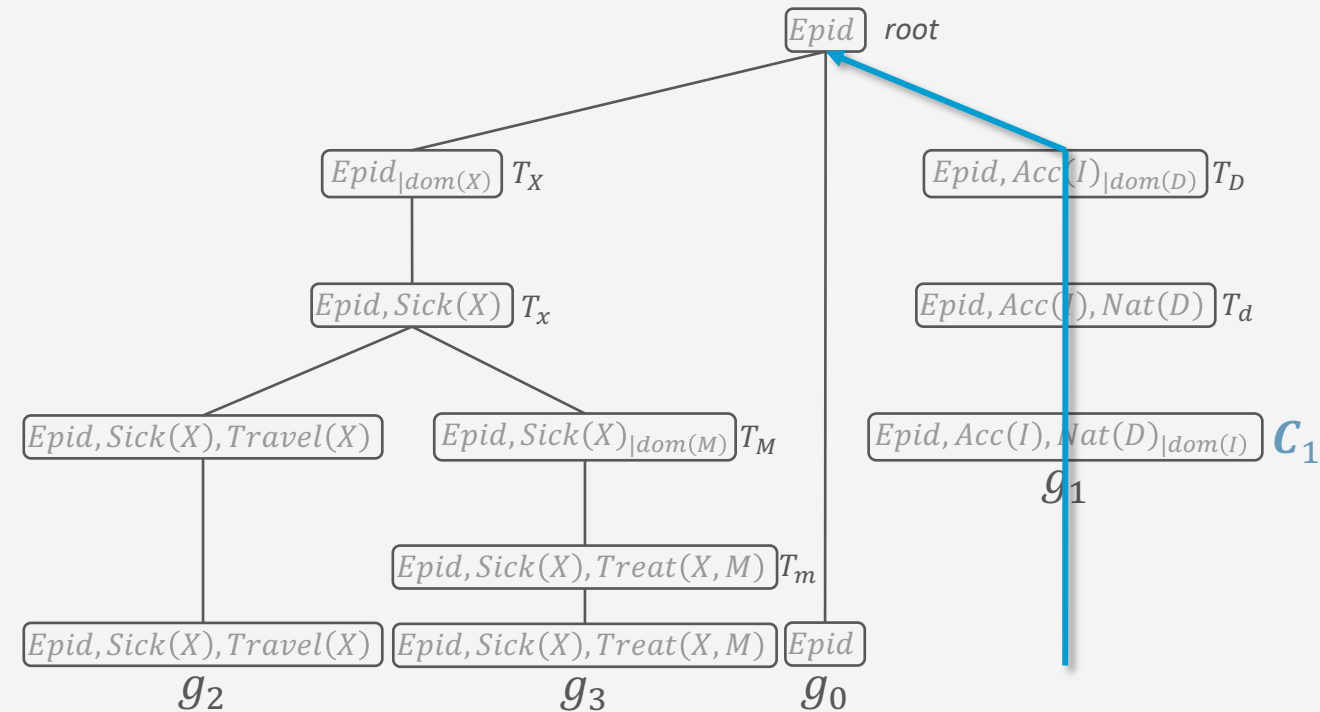
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it $\mathcal{C}_1$
 - Merge **inbound**
 - $\mathcal{C}_1$ and T_i parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_I parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)



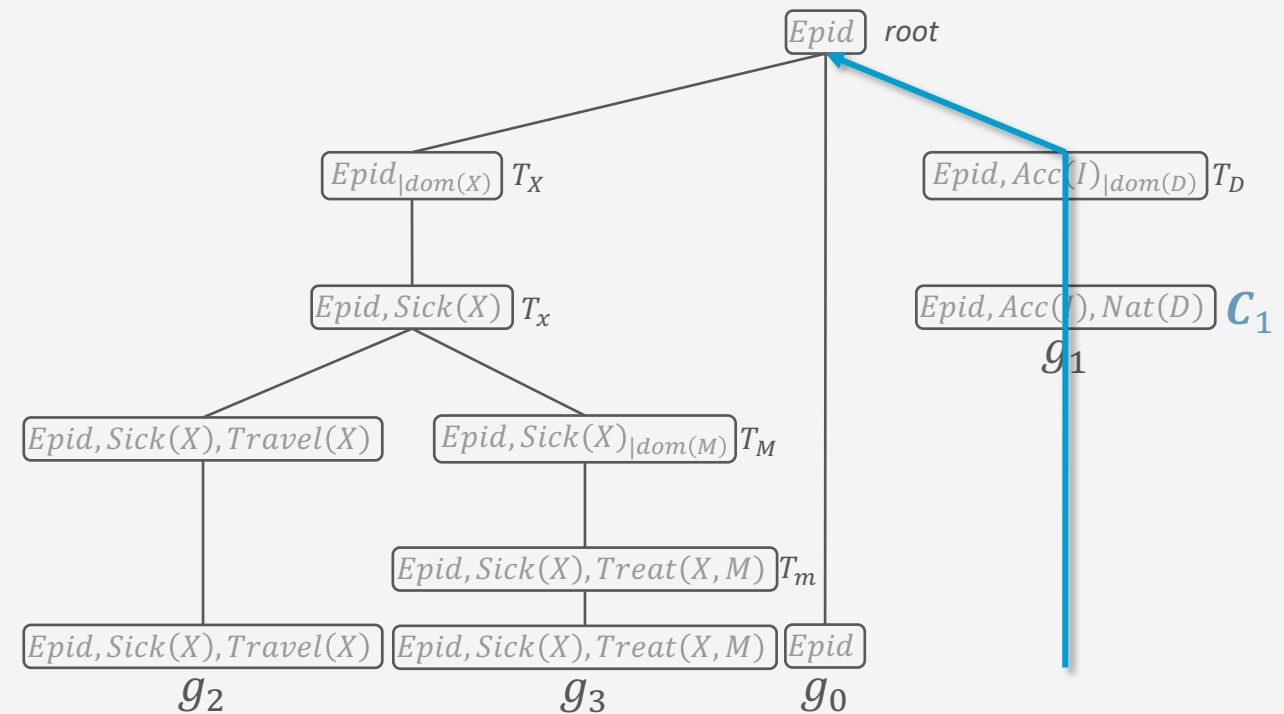
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it $\mathcal{C}_1$
 - Merge **inbound**
 - $\mathcal{C}_1$ and T_i parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_I parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_d parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)



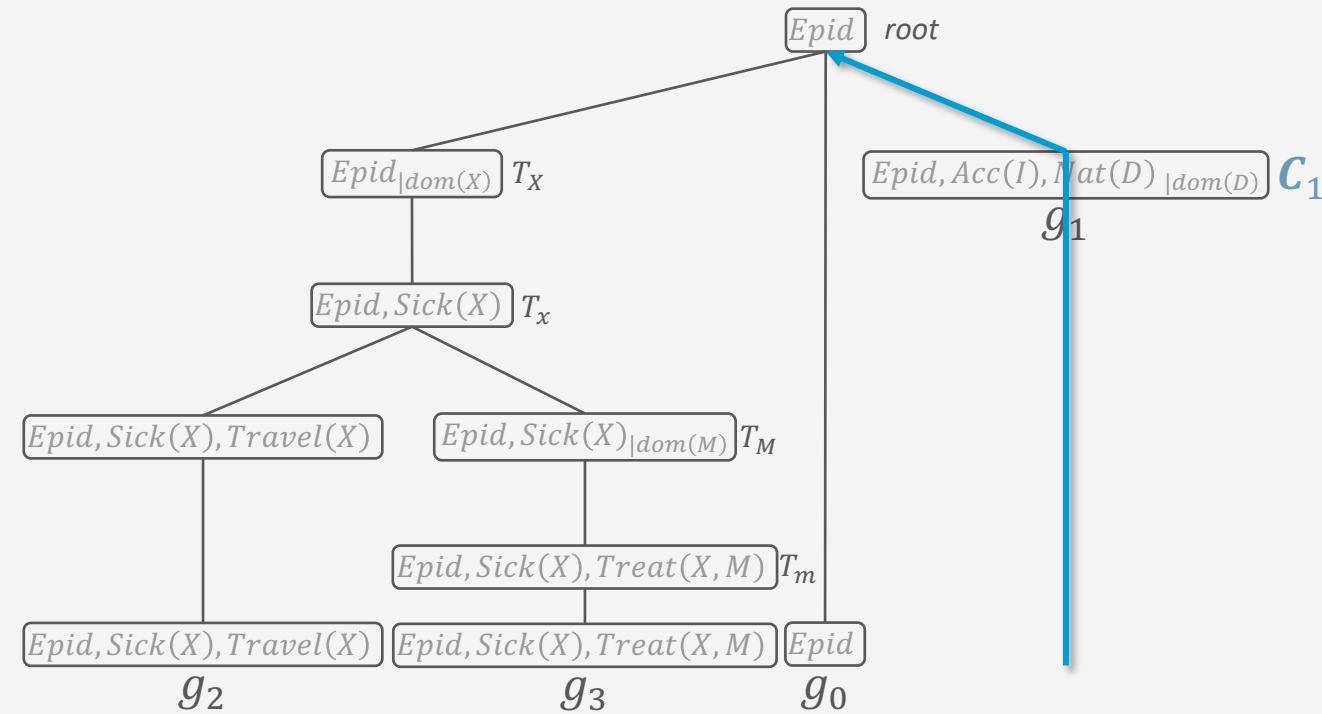
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it $\mathcal{C}_1$
 - Merge **inbound**
 - $\mathcal{C}_1$ and T_i parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_I parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_d parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - T_D parcluster subset of $\mathcal{C}_1$ $\rightarrow$ merge (call result $\mathcal{C}_1$ again)



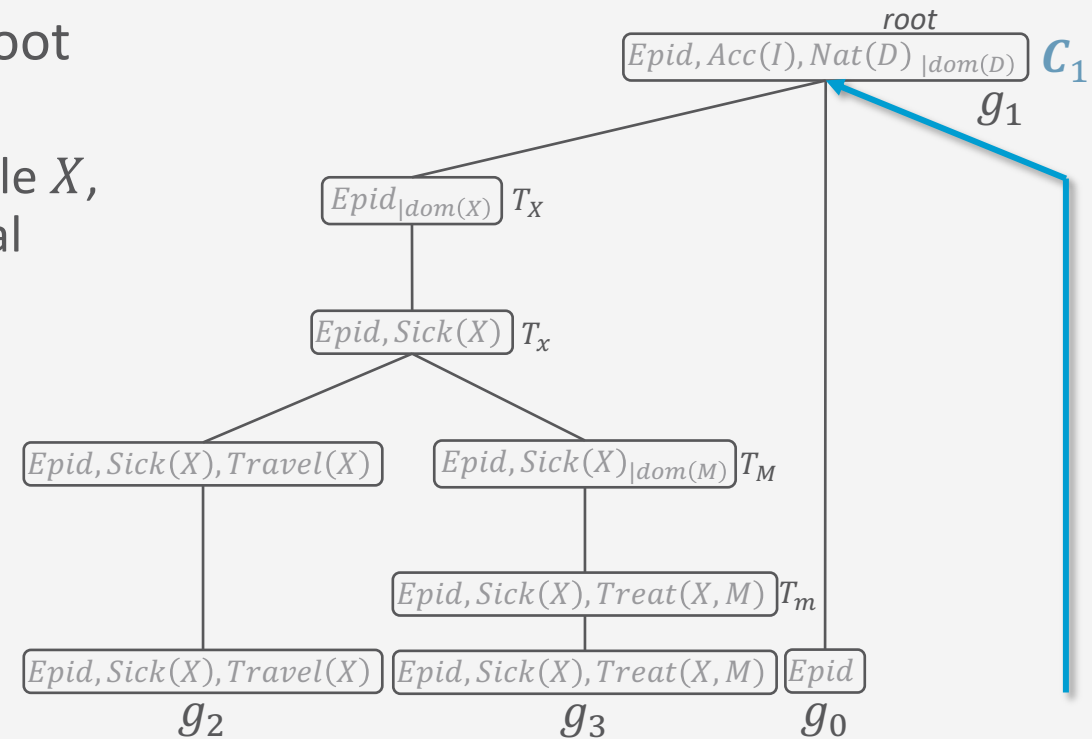
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it $\mathcal{C}_1$
 - Merge **inbound**
 - $\mathcal{C}_1$ and T_i parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_I parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - $\mathcal{C}_1$ and T_d parcluster identical $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - T_D parcluster subset of $\mathcal{C}_1$ $\rightarrow$ merge (call result $\mathcal{C}_1$ again)
 - Root parcluster subset of $\mathcal{C}_1$ $\rightarrow$ merge (call result $\mathcal{C}_1$ again)



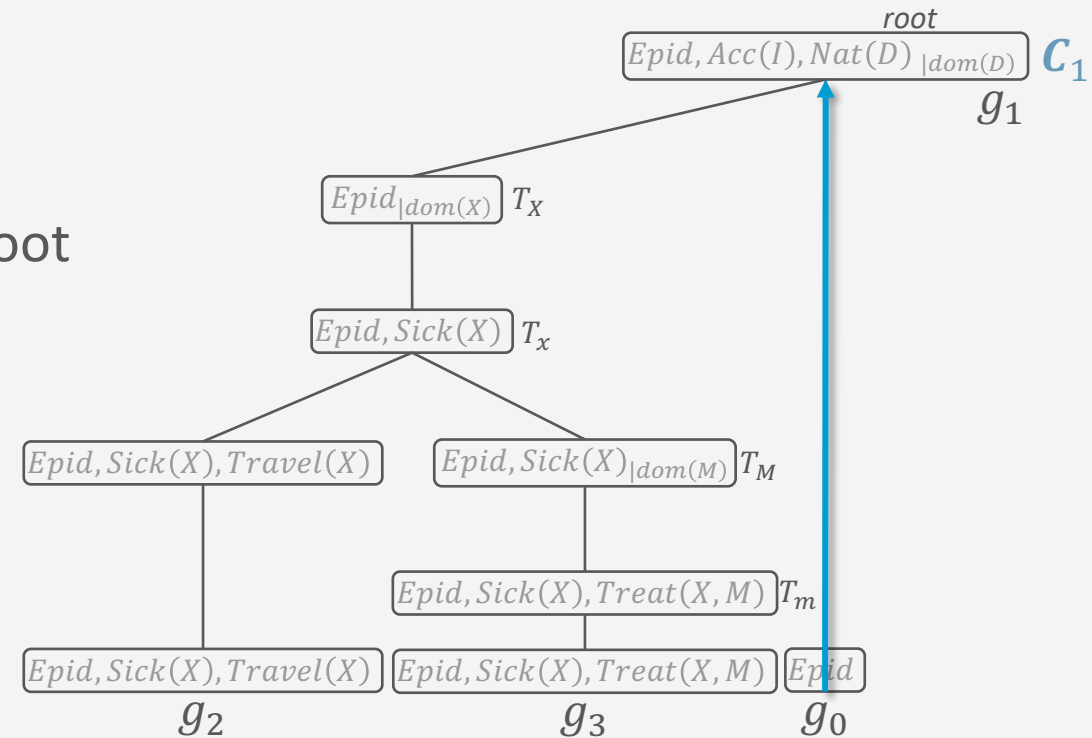
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
 - Let us call it $\mathcal{C}_1$
 - At this point, we have reached the former root and cannot merge further inbound
 - Also: the T_X parcluster contains logical variable X , which is not a subset or superset of the logical variables of $\mathcal{C}_1$ (D, I)
 - Merging stops



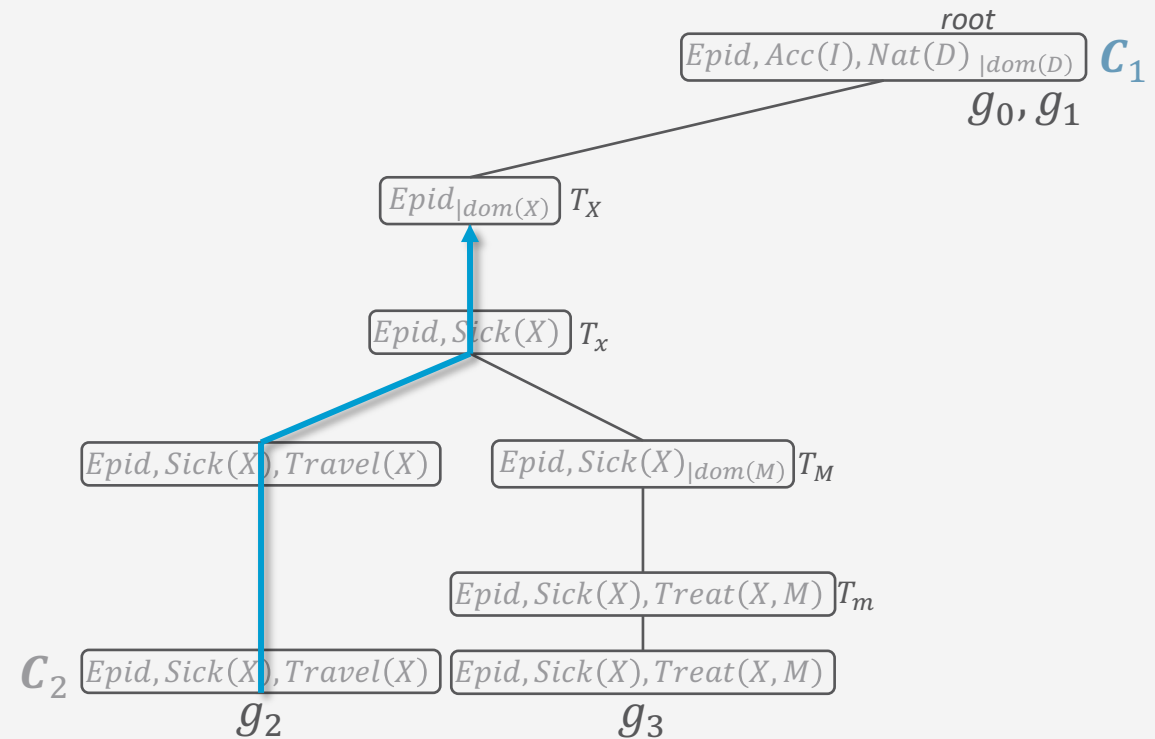
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_0\}$
 - Let us call it $\mathcal{C}_0$
 - Merge **inbound**
 - $\mathcal{C}_0$ subset of $\mathcal{C}_1$
→ merge (call result $\mathcal{C}_1$ again)
- At this point, we have reached the former root again and cannot merge further inbound
 - Also again: the T_X parcluster contains logical variable X , which is not a subset or superset of the logical variables of $\mathcal{C}_1$ (D, I)
 - Merging *stops*



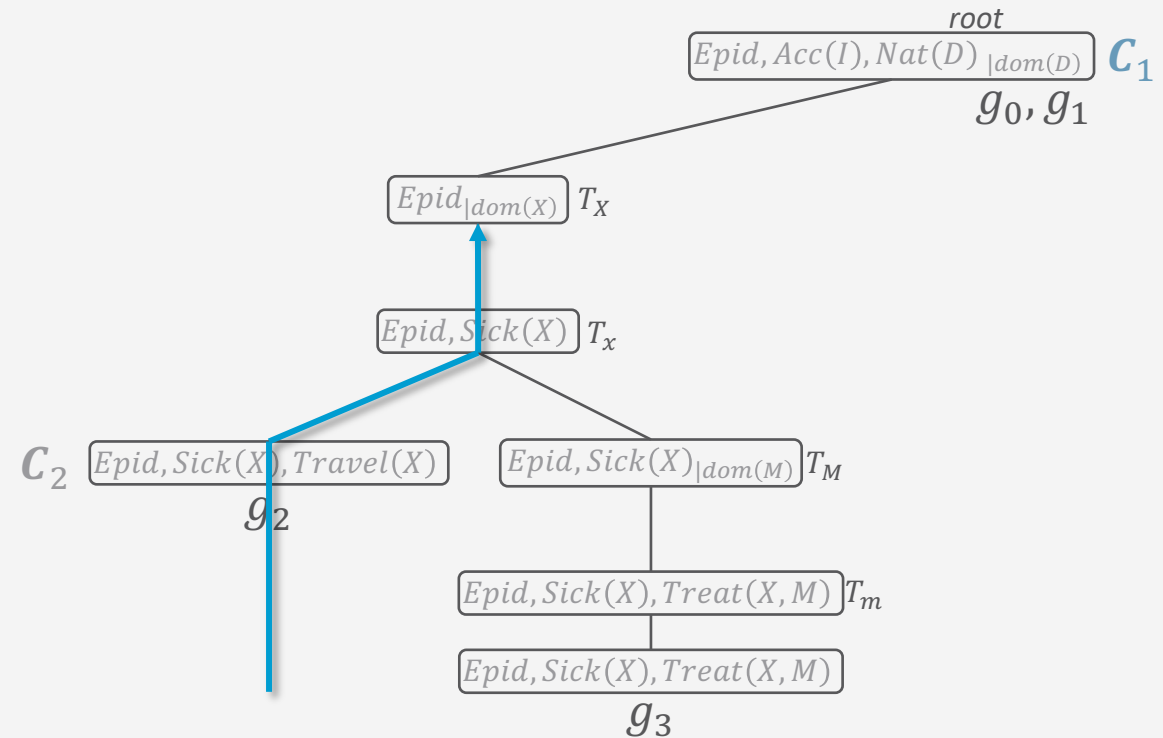
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_2\}$
 - Let us call it $\mathcal{C}_2$
 - Merge **inbound**
 - $\mathcal{C}_2$ and neighbouring parcluster identical
→ merge (call result $\mathcal{C}_2$ again)



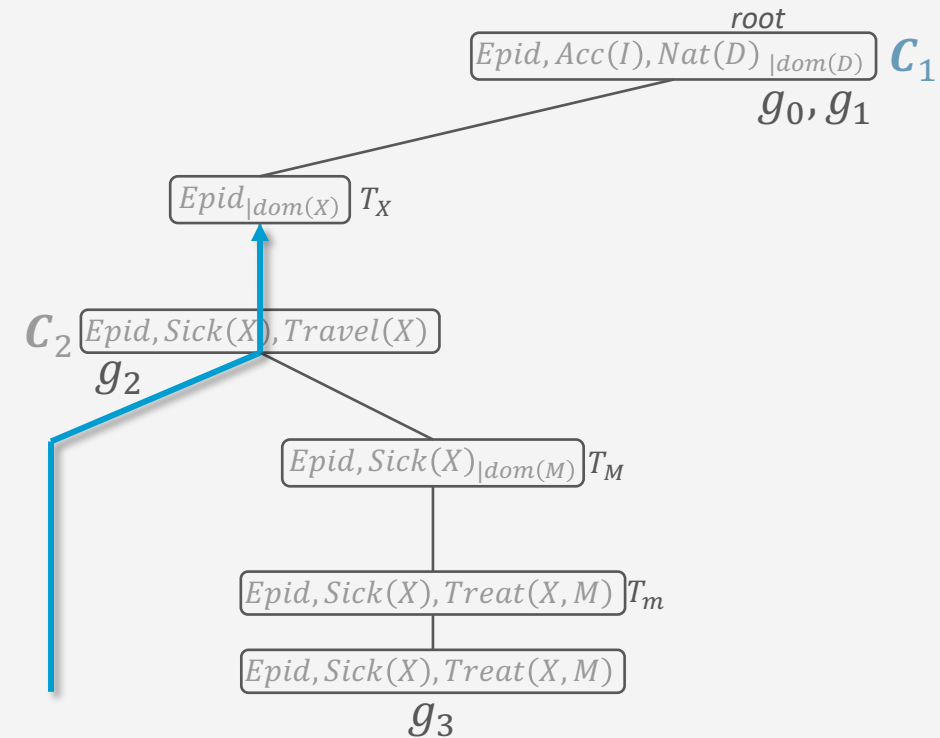
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_2\}$
 - Let us call it $\mathcal{C}_2$
 - Merge **inbound**
 - $\mathcal{C}_2$ and neighbouring parcluster identical
→ merge (call result $\mathcal{C}_2$ again)
 - T_x parcluster is a subset of $\mathcal{C}_2$
→ merge (call result $\mathcal{C}_2$ again)



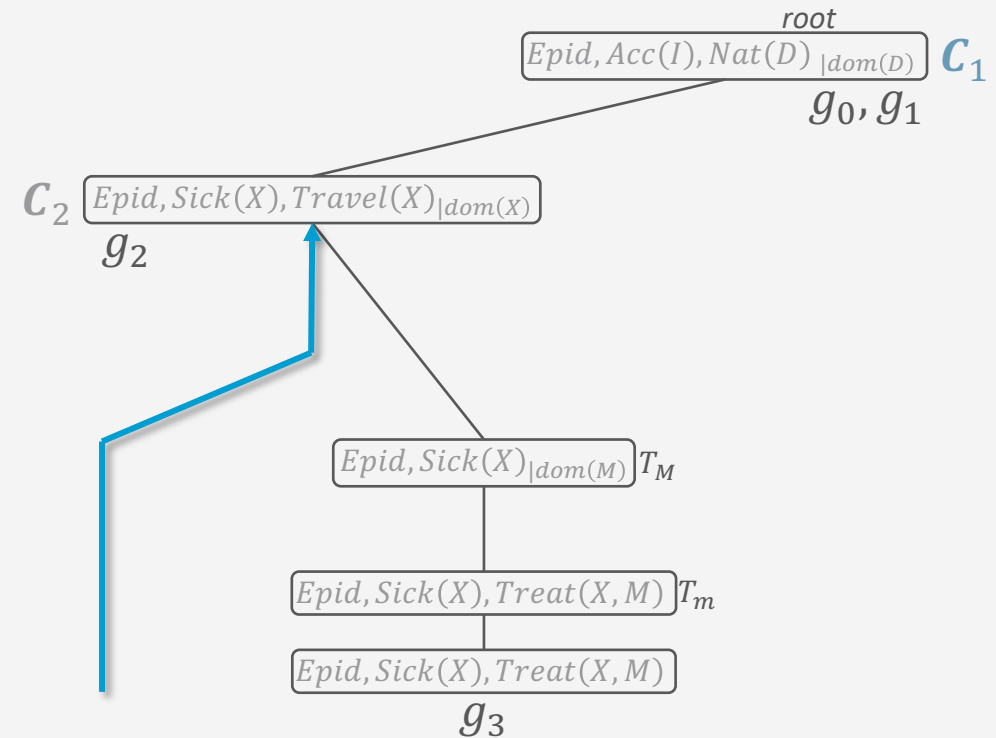
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_2\}$
 - Let us call it $\mathcal{C}_2$
 - Merge **inbound**
 - $\mathcal{C}_2$ and neighbouring parcluster identical
→ merge (call result $\mathcal{C}_2$ again)
 - T_x parcluster is a subset of $\mathcal{C}_2$
→ merge (call result $\mathcal{C}_2$ again)
 - T_x parcluster is a subset of $\mathcal{C}_2$
→ merge (call result $\mathcal{C}_2$ again)



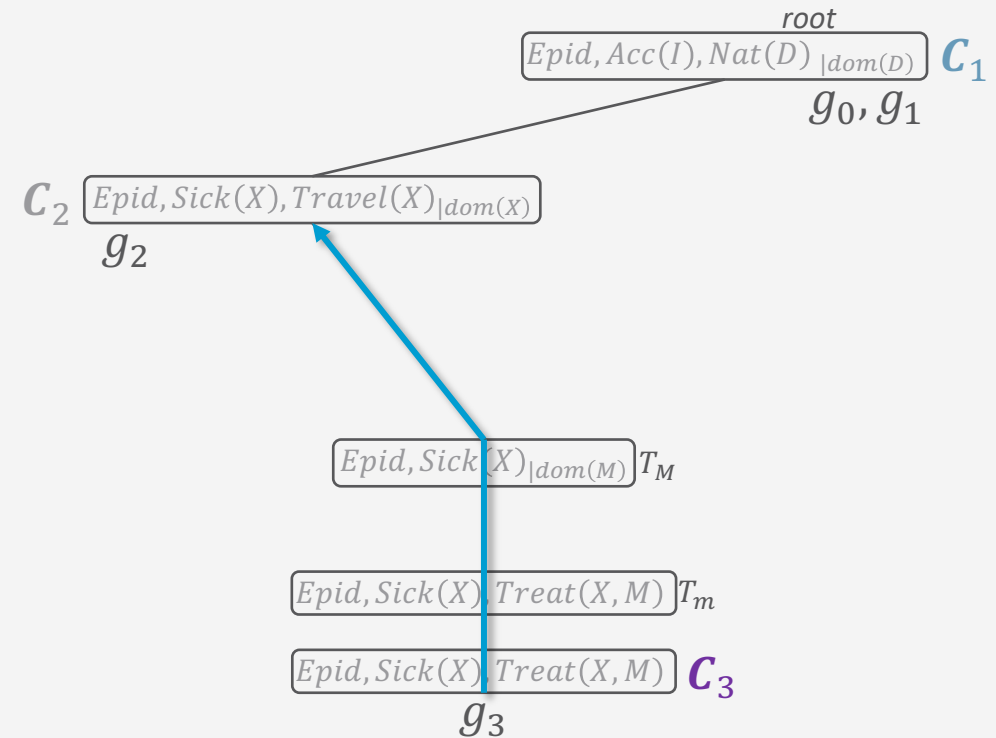
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_2\}$
 - Let us call it $\mathcal{C}_2$
 - Merge **inbound**
 - $\mathcal{C}_2$ and neighbouring parcluster identical
→ merge (call result $\mathcal{C}_2$ again)
 - T_x parcluster is a subset of $\mathcal{C}_2$
→ merge (call result $\mathcal{C}_2$ again)
 - T_X parcluster is a subset of $\mathcal{C}_2$
→ merge (call result $\mathcal{C}_2$ again)
 - Merging cannot move further inbound
 - $\mathcal{C}_1$ is neither a subset nor a superset of $\mathcal{C}_2$
 - Merging *stops*



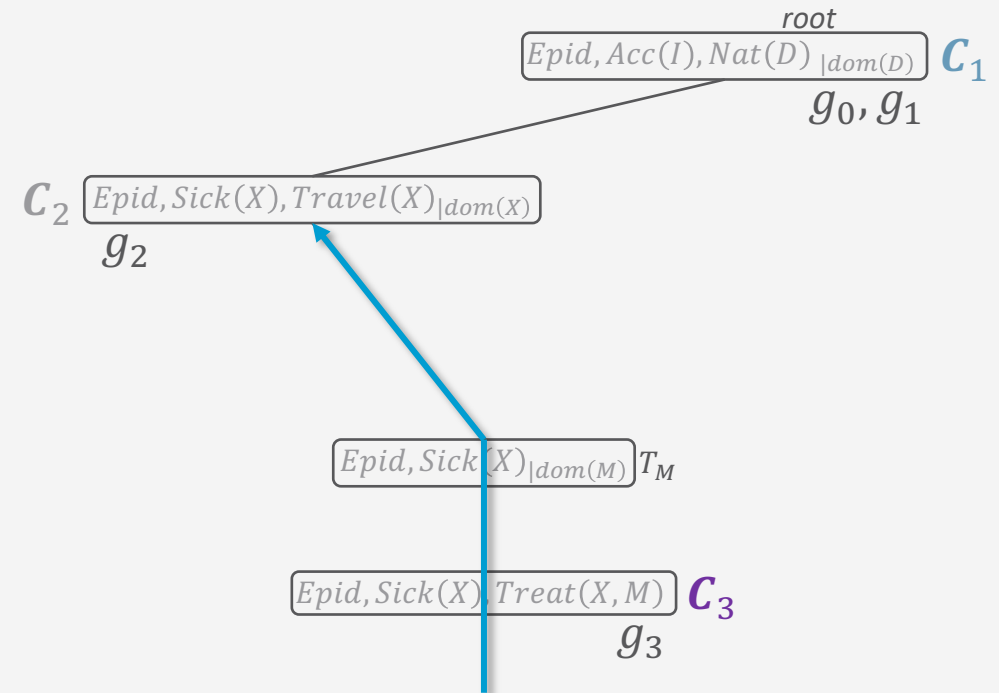
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_3\}$
 - Let us call it $\mathcal{C}_3$
 - Merge **inbound**
 - $\mathcal{C}_3$ and T_m parcluster identical
→ merge (call result $\mathcal{C}_3$ again)



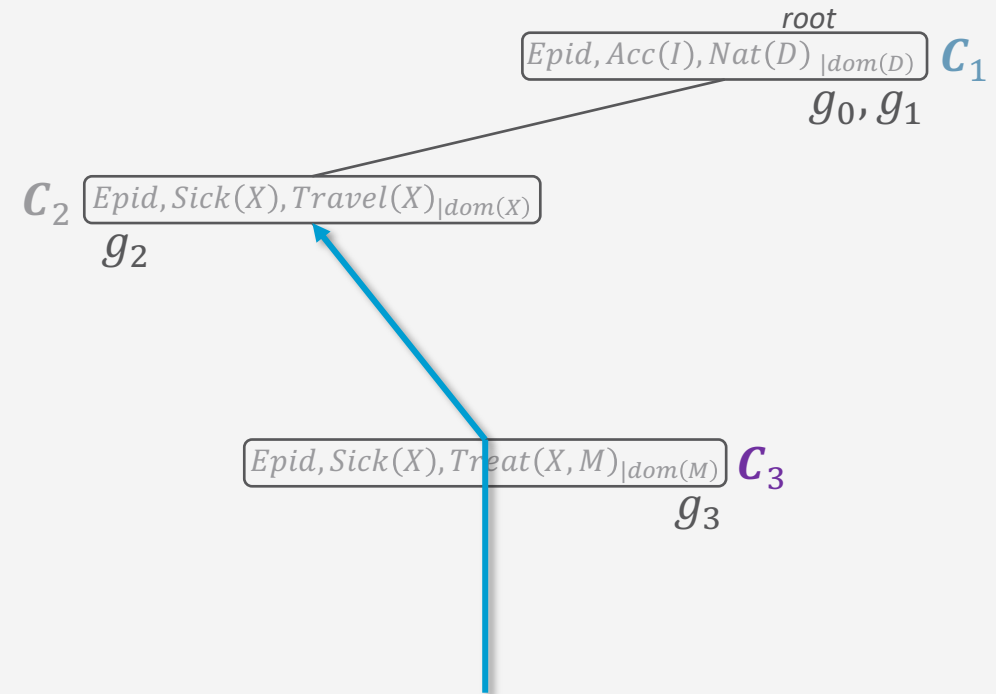
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_3\}$
 - Let us call it $\mathcal{C}_3$
 - Merge **inbound**
 - $\mathcal{C}_3$ and T_m parcluster identical
→ merge (call result $\mathcal{C}_3$ again)
 - T_M parcluster is a subset of $\mathcal{C}_3$
→ merge (call result $\mathcal{C}_3$ again)



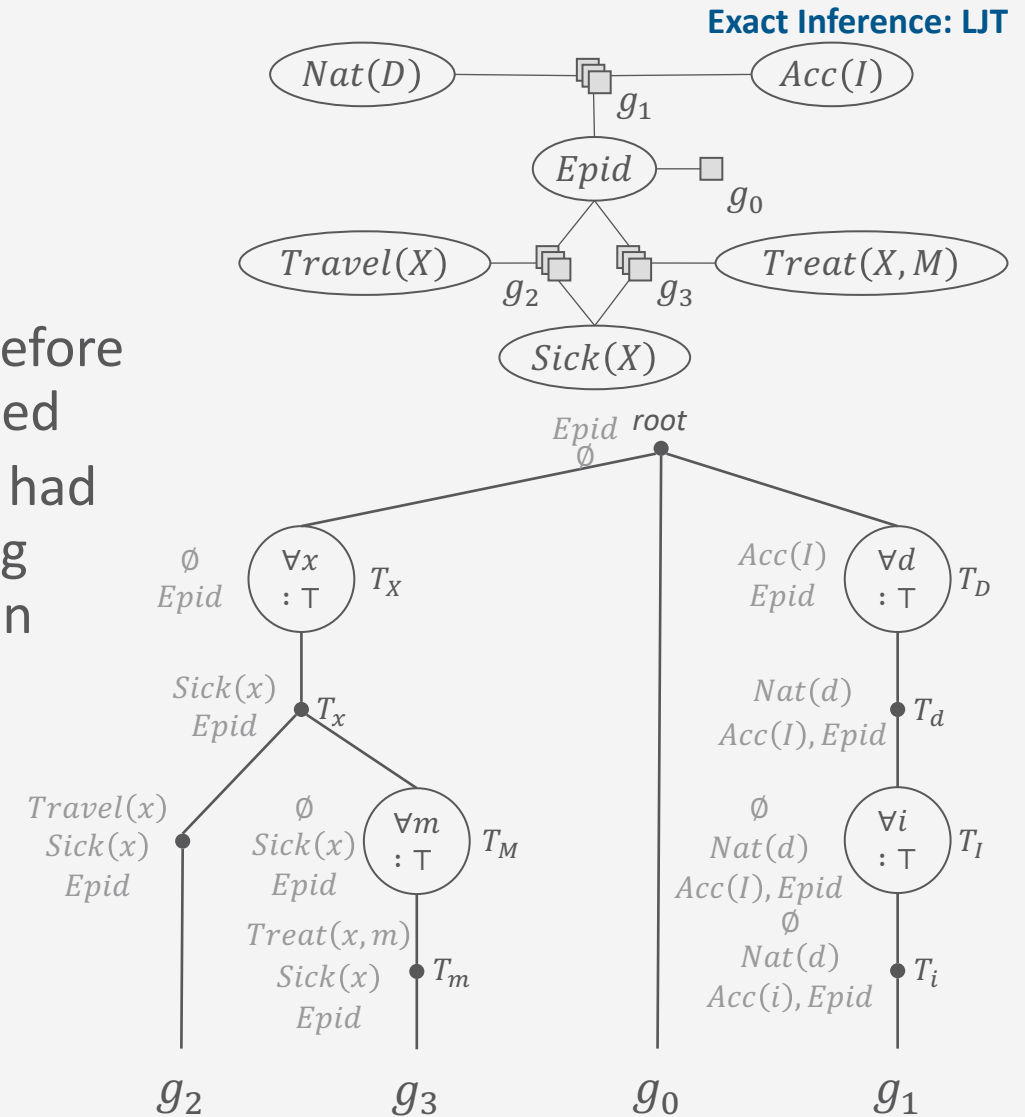
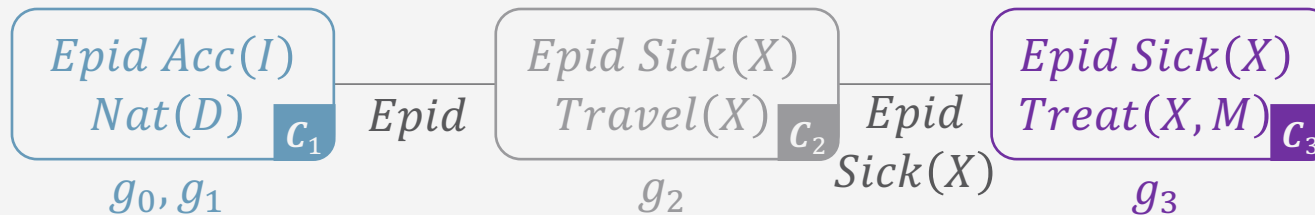
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_3\}$
 - Let us call it $\mathcal{C}_3$
 - Merge **inbound**
 - $\mathcal{C}_3$ and T_m parcluster identical
→ merge (call result $\mathcal{C}_3$ again)
 - T_M parcluster is a subset of $\mathcal{C}_3$
→ merge (call result $\mathcal{C}_3$ again)
 - Merging cannot move further inbound
 - $\mathcal{C}_3$ is neither a subset nor a superset of $\mathcal{C}_2$
 - Merging *stops*



Minimisation: Example Continued

- Resulting FO jtree J from FO dtree T given model G
 - If we had started merging from leaf with g_3 inbound before merging from leaf with g_2 , C_2 and C_3 would be switched
 - g_0 could have made one of the other parclusters if we had started merging from leaf with g_2 or g_3 before merging from leaf with g_0 or by starting at leaf with g_0 and then merging from leaf with g_2 or g_3



FO Jtree Construction

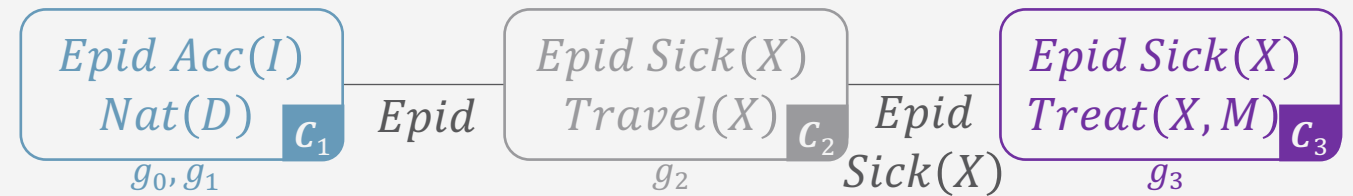
- Given a model G , the following steps are necessary
 1. Bring G into the required normal form for FO dtree construction
 2. Construct an FO dtree T for G
 3. Translate T into an FO jtree J
 4. Apply inverse substitutions to parclusters of descendants of DPG nodes in J
 5. Minimise J

Construction

- Next?
- FO jtrees for query answering
 - Messages need to be passed to ensure independence
 - What about evidence?

Message Passing in FO Jtrees

- Ensure independence between parclusters
- Send messages based on two conditions
 - If a node i has received all messages from neighbours but one, j , node i calculates and sends a message to j
 - If a node i has received all messages, then it calculates and sends messages to all neighbours j that have not received a message yet

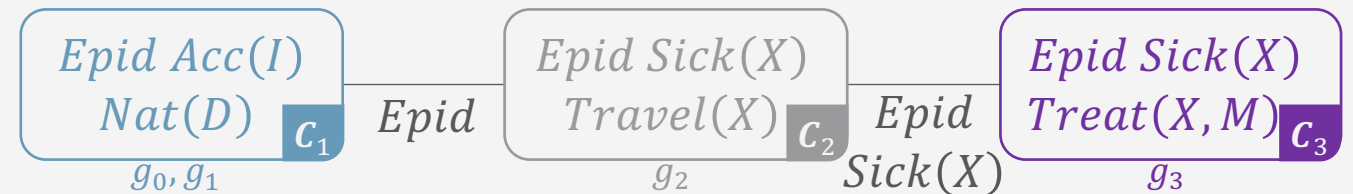


Message Passing in FO Jtrees

- Message m_{ij} from sender C_i to receiver C_j
 - Set of parfactors $\{g_l\}_{l=1}^n$ with $rv(g_l) \subseteq S_{ij}$
 - To calculate
 - Collect necessary information from local model and received messages:

$$G_{ij} = G_i \cup \bigcup_{k \in nbs(i), k \neq j} m_{ki}$$

- Ignore the message that came from C_j (if it already exists)
- Call slightly modified LVE with G_{ij} as input model, S_{ij} as query, and no evidence: LVE–MSG(G_{ij}, S_{ij})
 - Specification of LVE–MSG: next slide



LVE for Message Passing

LVE—MSG(G, S)

$G \leftarrow$ Shatter G on itself

while G contains non-query terms **do**

if a PRV A fulfils the preconditions of sum—out **then**

$G \leftarrow$ Apply sum—out to A in G

else

$G \leftarrow$ Apply an enabling operator on some parfactors in G

return G

No shattering on separator (due to construction) or evidence, no absorption (will have been handled)

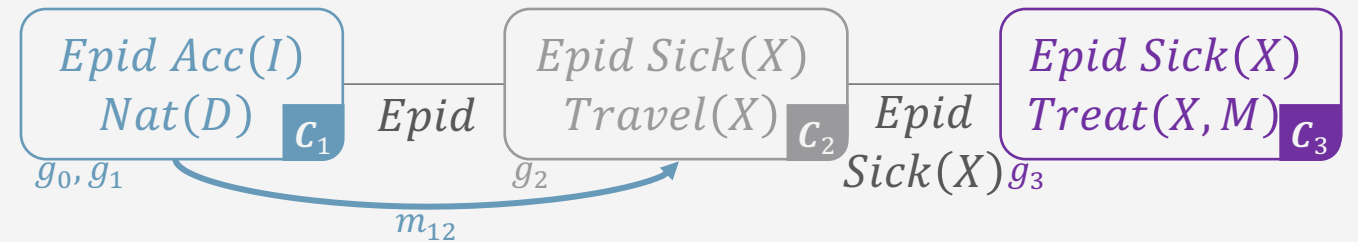
- Model might need to be shattered on itself because of splits introduced by messages

No normalisation (and multiplication of the remaining factors to be able to normalise) at the end

- Interim result returned

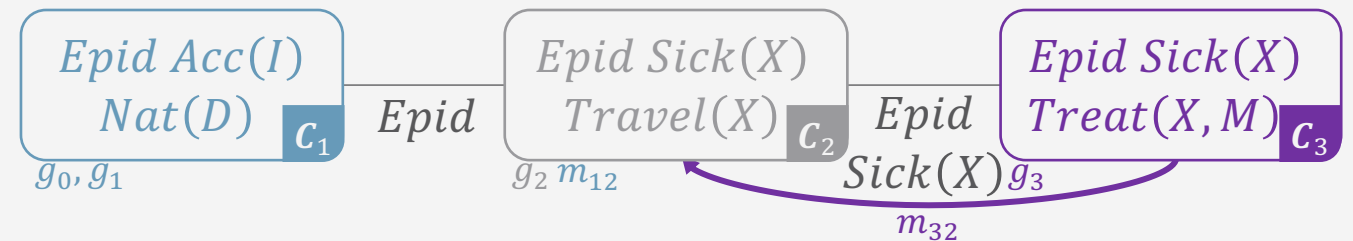
Message Passing in FO Jtrees: Example

- Message m_{12} from C_1 to C_2
 - Collect $G_{12} = \{g_1\} \cup \emptyset$
 - No further neighbours except C_2
 - Call LVE—MSG($\{g_1\}, \{Epid\}, \emptyset$)
 - LVE—MSG eliminates $Nat(D), Acc(I)$ from $\{g_1\}$
 - Count-converting $Nat(D)$ into $\#_D[Nat(D)]$
 - Summing out $Acc(I)$
 - Summing out $\#_D[Nat(D)]$
 - Returning $\{g'_1\}$
 - Send $\{g'_1\}$ as m_{12} to C_2



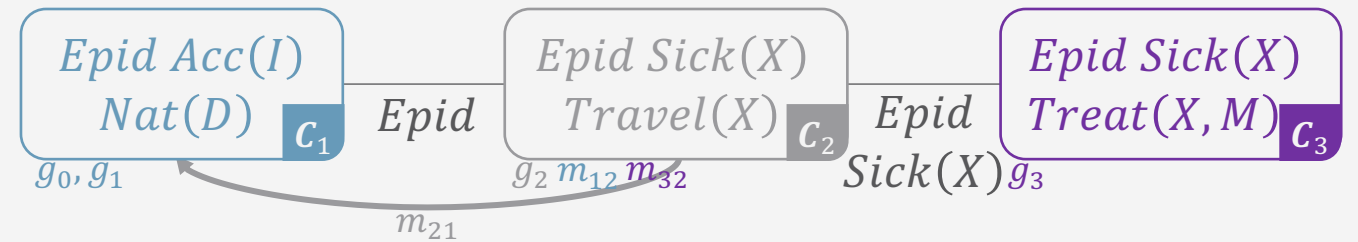
Message Passing in FO Jtrees: Example

- Message m_{32} from C_3 to C_2
 - Collect $G_{32} = \{g_3\} \cup \emptyset$
 - No further neighbours except C_2
 - Call $\text{LVE-MSG}(\{g_3\}, \{\text{Epid}, \text{Sick}(X)\}, \emptyset)$
 - LVE-MSG eliminates $\text{Treat}(X, M)$ from $\{g_3\}$
 - Summing out $\text{Treat}(X, M)$
 - Returning $\{g'_3\}$
 - Send $\{g'_3\}$ as m_{32} to C_2



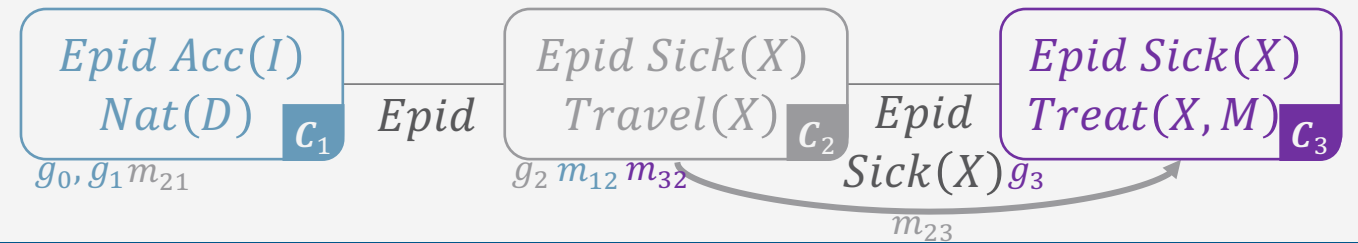
Message Passing in FO Jtrees: Example

- Message m_{21} from C_2 to C_1
 - Collect $G_{21} = \{g_2\} \cup m_{32}$
 - Further neighbour: C_3 , sent message $m_{32} = \{g'_3\}$
 - Call $\text{LVE-MSG}(\{g_2, g'_3\}, \{\text{Epid}\}, \emptyset)$
 - LVE-MSG eliminates $\text{Travel}(X), \text{Sick}(X)$ from $\{g_2, g'_3\}$
 - Summing out $\text{Travel}(X)$ from g_2 , yielding g'_2
 - Summing out $\text{Sick}(X)$ from product of g'_2 and g'_3 , yielding g'_{23}
 - Returning $\{g'_{23}\}$
 - Send $\{g'_{23}\}$ as m_{21} to C_1



Message Passing in FO Jtrees: Example

- Message m_{23} from C_2 to C_3
 - Collect $G_{23} = \{g_2\} \cup m_{12}$
 - Further neighbour: C_1 , sent message $m_{12} = \{g'_1\}$
 - Call $\text{LVE-MSG}(\{g_2, g'_1\}, \{\text{Epid}, \text{Sick}(X)\}, \emptyset)$
 - LVE-MSG eliminates $\text{Travel}(X)$ from $\{g_2, g'_1\}$
 - Summing out $\text{Travel}(X)$ from g_2 , yielding g'_2
 - Returning $\{g'_2, g'_1\}$
 - Send $\{g'_2, g'_1\}$ as m_{23} to C_3



Message Passing: Overview

- Given an FO jtree J , send messages if one of the two conditions is true
 - If a node i has received all messages from neighbours but one, j , node i calculates and sends a message to j
 - If a node i has received all messages, then it calculates and sends messages to all neighbours j that have not received a message yet

- To calculate a message:

- Collect necessary information from local model and received messages:

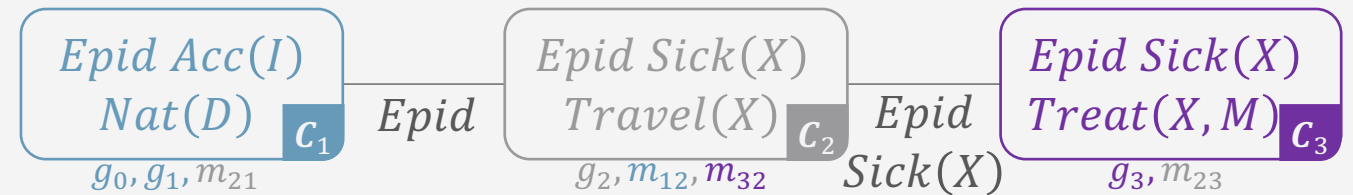
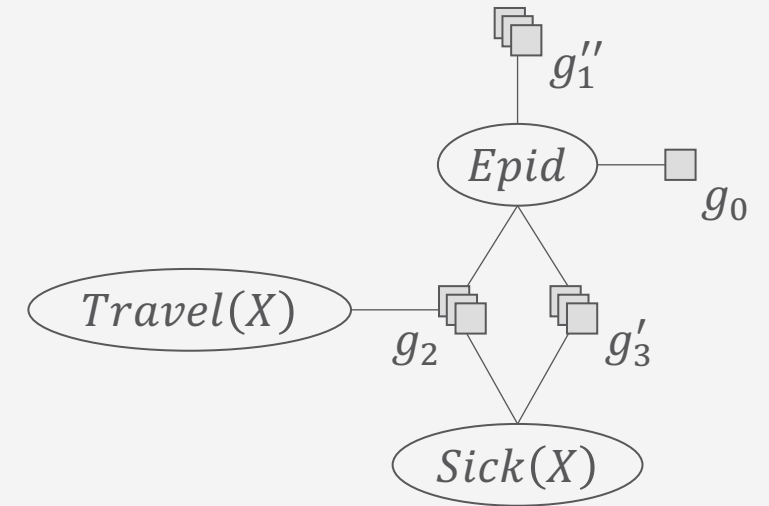
$$G_{ij} = G_i \cup \bigcup_{k \in \text{nbs}(i), k \neq j} m_{ki}$$

- Call LVE—MSG($G_{ij}, \mathcal{S}_{ij}$)

Message Passing

Query Answering in FO Jtrees

- Idea
 - Pick parcluster in which query terms occur
 - Use local model and outside messages as input model for LVE
 - E.g., for $P(Epid)$
 - All parclusters contain *Epid*, choose one at random, e.g., C_2
 - Collect $G_{Epid} = \{g_2\} \cup m_{12} \cup m_{32} = \{g_2, g'_1, g'_3\}$ (depicted right)
 - Call $LVE(\{g_2, g'_1, g'_3\}, Epid, \emptyset)$, yielding a parfactor g containing the probability distribution over *Epid*
- What if query terms occur outside of one parcluster?
 - E.g., $P(Travel(eve), Treat(eve, m_1))$



Query Answering in FO Jtrees

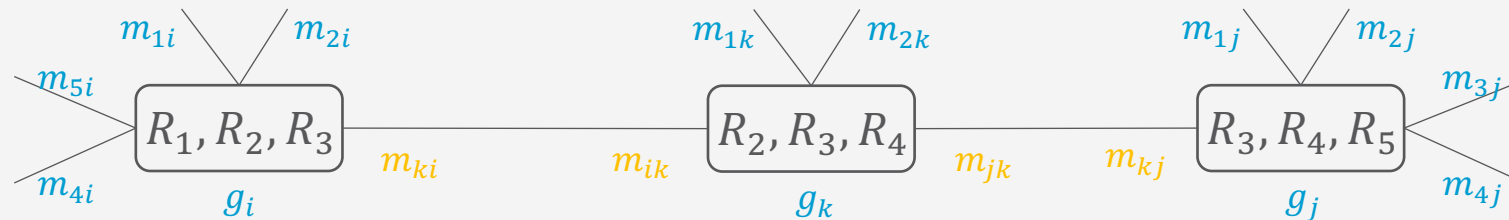
- For query terms Q , possibly contained in more than one parcluster
 - Find a subgraph J' of the FO jtree J such that $Q \subseteq rv(J')$
 - Use local models in J' and messages from outside J' as basis for calling LVE
 - No duplicate information used
 - E.g., query on R_1, R_5 using C_i, C_k, C_j
 - Ignore inside messages $m_{ki}, m_{ik}, m_{jk}, m_{kj}$

How can we find a subgraph?

- Subgraph should be minimal in the number of PRVs in it for optimal performance:

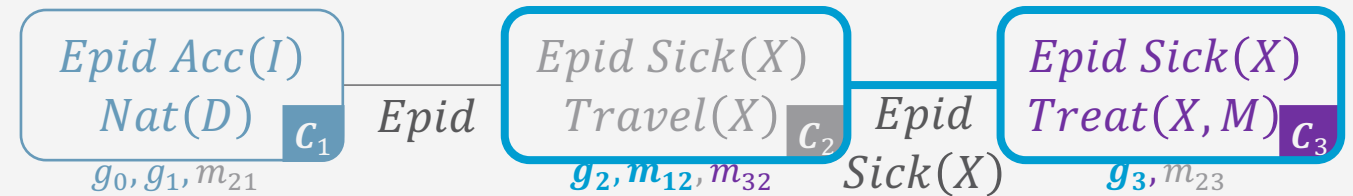
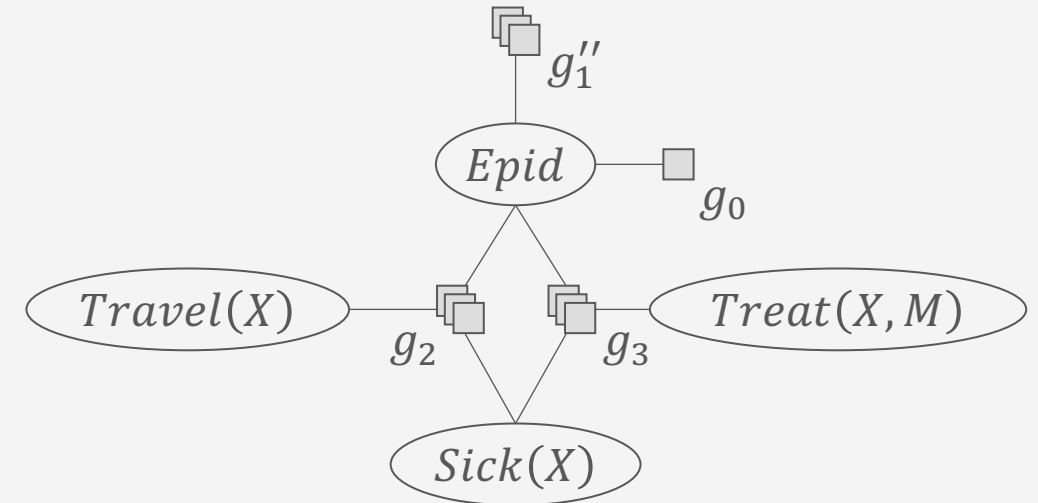
$$\begin{aligned} &\operatorname{argmin} |rv(J')| \\ &\text{s.t. } Q \subseteq rv(J') \end{aligned}$$

- Trade-off between finding a subgraph fast and finding a minimal one
- It is not about the number of parclusters!



Query Answering in FO Jtrees: Example

- E.g., $P(\text{Travel}(\text{eve}), \text{Treat}(\text{eve}, m_1))$
 - Subgraph: C_2, C_3
 - Submodel for query answering: $G_Q = (g_2, g_3, m_{12})$
 - Depicted right
 - Call LVE with G_Q and $Q = \{\text{Travel}(\text{eve}), \text{Treat}(\text{eve}, m_1)\}$
 - Split off query terms
 - Eliminate all non-query terms
 - Normalise the result



Query Answering in FO Jtrees

- After message passing, parclusters independent from each other given messages
 - Prepared for query answering

- For each query with query terms Q

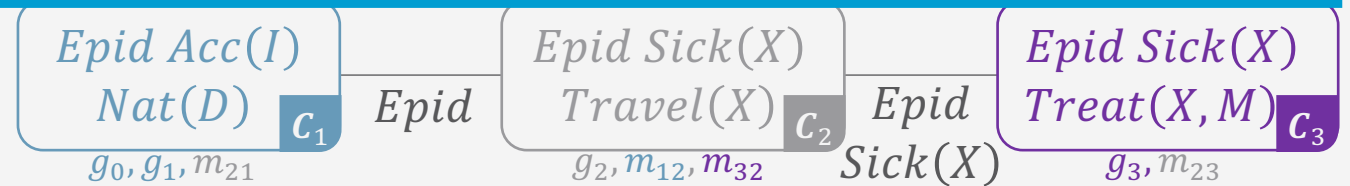
- Find subtree $J' = (V', E')$ s.t. $Q \subseteq rv(J')$
- Collect information from local model and messages, i.e.,

$$G_Q = \bigcup_{i \in V'} G_i \cup \bigcup_{\substack{j \in nbs(i) \\ j \notin V'}} m_{ji}$$

- Call $LVE(G_Q, Q, \emptyset)$ and return or store result of the call

Query Answering

- What about evidence?

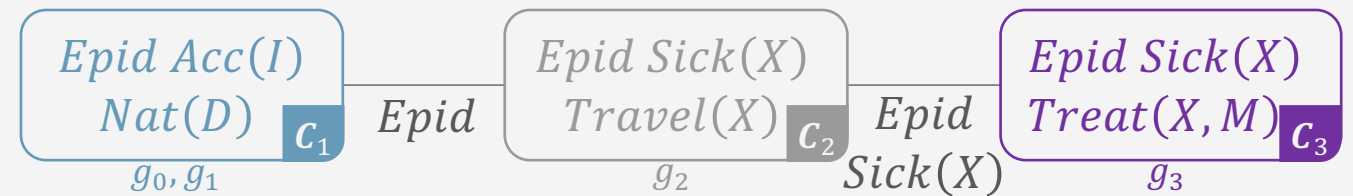


Evidence in FO Jtrees

- Evidence applies to PRVs in some parclusters
 - Changes the distributions in local models
 - Information sent in messages might change
 - Even if summed out and therefore hidden from the other parclusters
- Therefore, handle evidence before sending messages
 - Only then send messages

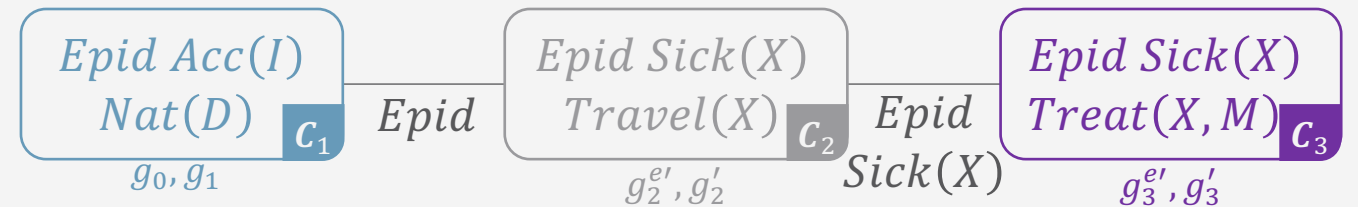
- Given a set of evidence parfactors $\{\phi_e(A_e)_{|C_e}\}_{e=1}^m$
- For each $\phi_e(A_e)_{|C_e}$
 - For *each* parcluster C_i where $A_e \in C_i$
 - Shatter G_i on C_e
 - Absorb $\phi_e(A_e)_{|C_e}$ in G_i

Evidence Entering



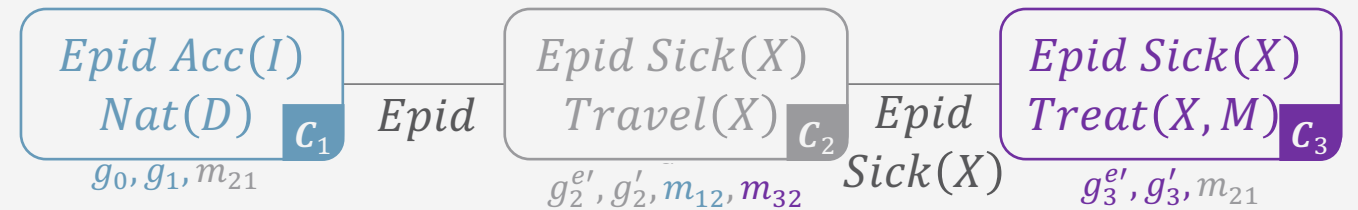
Evidence in FO Jtrees: Example

- Given $Sick(eve) = true$ as evidence g_e
 - In $\mathcal{C}_2$
 - Shatter $G_2 = \{g_2\}$ on $Sick(eve)$, yielding $\{g_2^e, g_2'\}$
 - Absorb g_e in g_2^e , yielding $g_2^{e'}$
 - Result: $G_2 = \{g_2^{e'}, g_2'\}$
 - In $\mathcal{C}_3$
 - Shatter $G_3 = \{g_3\}$ on $Sick(eve)$, yielding $\{g_3^e, g_3'\}$
 - Absorb g_e in g_3^e , yielding $g_3^{e'}$
 - Result: $G_3 = \{g_3^{e'}, g_3'\}$
- After evidence handling, send messages based on the local models that have absorbed the evidence
 - $G_0 = \{g_0, g_1\}$ (unchanged)
 - $G_2 = \{g_2^{e'}, g_2'\}$
 - $G_3 = \{g_3^{e'}, g_3'\}$



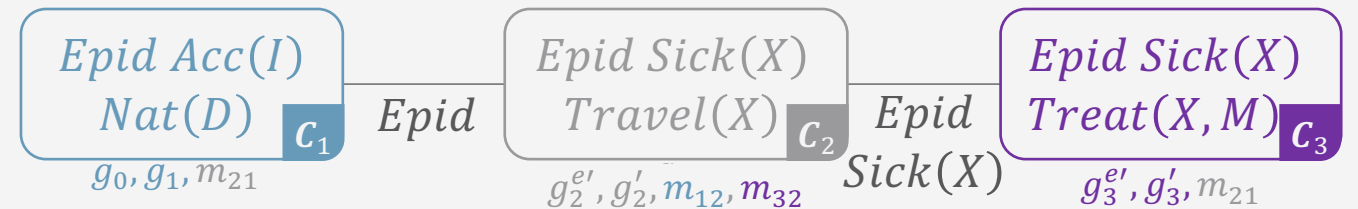
Evidence in FO Jtrees

- E.g., given $Sick(eve) = true$ as evidence in g_e
 - Message m_{12} does not change compared to previous example
 - Message m_{32} calculated based on $\{g_3^{e'}, g_3'\}$
 - Call LVE-MSG($\{g_3^{e'}, g_3'\}, \{Epid, Sick(X)\}, \emptyset$), yielding $\{g_3^{e''}, g_3''\}$
 - Message m_{23} calculated based on $\{g_2^{e'}, g_2'\} \cup m_{12}$
 - Call LVE-MSG($\{g_2^{e'}, g_2', g_1'\}, \{Epid, Sick(X)\}, \emptyset$), yielding $\{g_2^{e''}, g_2'', g_1'\}$
 - Message m_{21} calculated based on $\{g_2^{e'}, g_2'\} \cup m_{32}$
 - Call LVE-MSG($\{g_2^{e'}, g_2', g_3^{e''}, g_3''\}, \{Epid\}, \emptyset$), yielding $\{g_2^{e''}, g_2'', g_3^{e''}, g_3'''\}$



Evidence and Queries in FO Jtrees

- After evidence handling
 - All queries are answered in an FO jtree with handled evidence $\{g_e\}_{e=1}^m$ yield results conditional on $\{g_e\}_{e=1}^m$
 - So, given evidence $\{g_e\}_{e=1}^m$ and query terms $\{Q_i\}_{i=1}^n$ for a model G
 - The posed queries are $P(Q_i \mid \{g_e\}_{e=1}^m)$, $1 \leq i \leq n$, w.r.t. P_G
- FO jtree constructed without specific evidence
 - Reuse for different evidence sets
 - As long as model stays the same
 - Reset the local models before entering new evidence



LJT: Algorithm

$\text{LJT}(G, \{Q_i\}_{i=1}^n, \{g_e\}_{e=1}^m)$

Construct an FO jtree J for G

Enter evidence $\{g_e\}_{e=1}^m$ into J

Pass message in J

Answer queries with query terms $\{Q_i\}_{i=1}^n$ in J

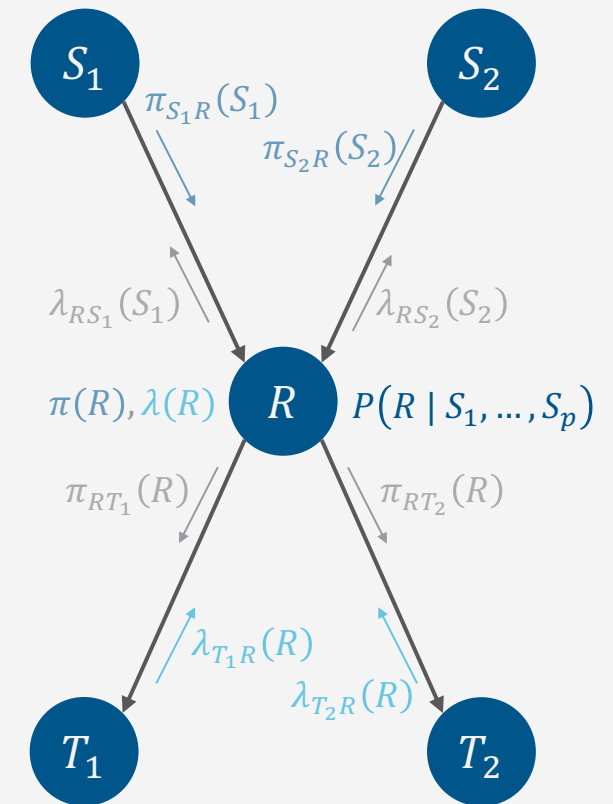
-
- Look for blue boxes on the previous slides to find the descriptions of each step

Step Name

- *Constant overhead* for FO jtree construction
- Payoff if given multiple queries

Foundations of Clustering

- History in propositional probabilistic inference:
 - Based on **probability propagation** introduced by Pearl (1988)
 - If a BN is a **polytree**, i.e., the underlying undirected graph has no trivial cycles, then
 - Treat each node in a BN as a cluster with the random variables of the accompanying conditional probability table as the cluster random variables
 - Send messages along the edges (to parents and children), eliminating random variables not occurring in the parent or child nodes

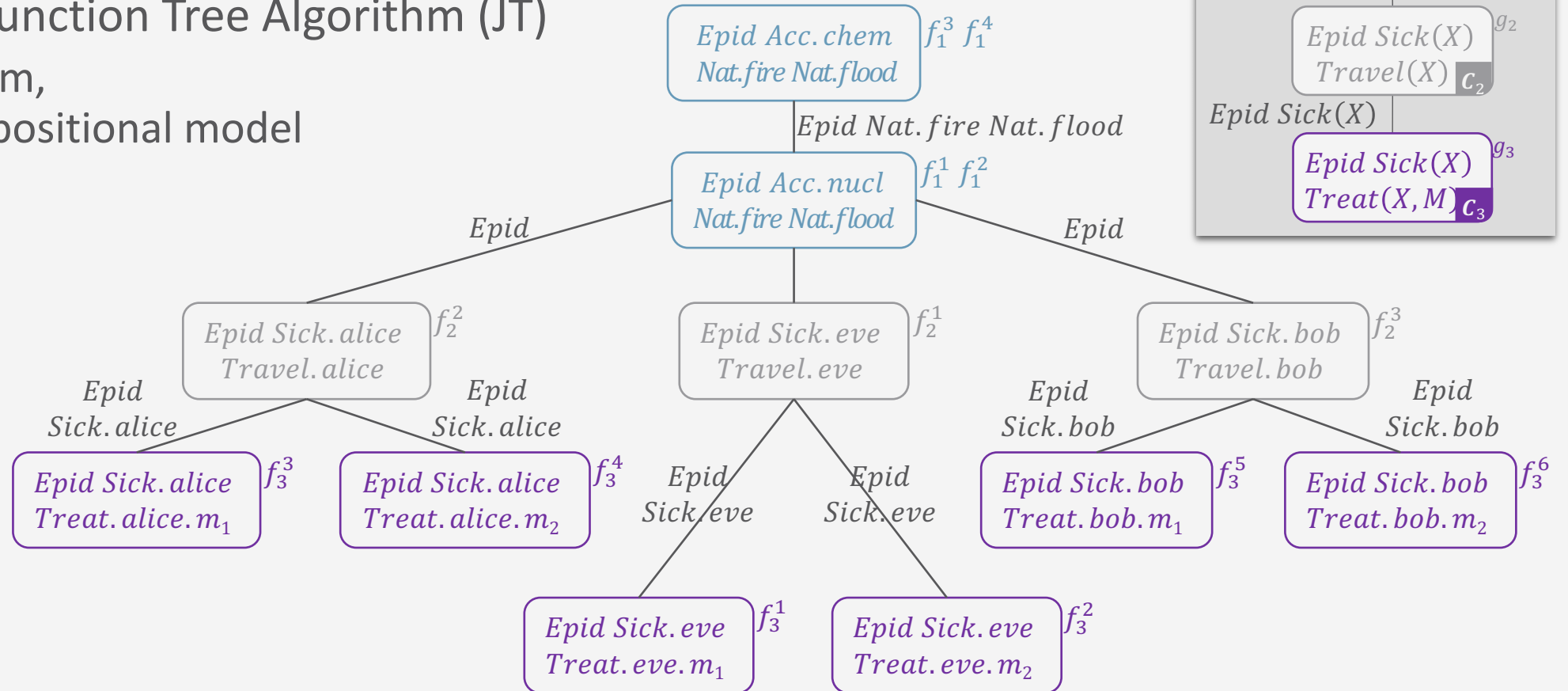


Foundations of Clustering

- History in propositional probabilistic inference:
 - If no polytree, the cycles mess up the message passing along the edges (information arrives multiple times)
 - Send messages nonetheless (exact if polytree, approximate otherwise): called **belief propagation** as an algorithm for *approximate* inference
 - Exact inference required → put the cycles into one cluster
 - Graph formed called a junction tree (**jtree**)
 - A first-order version of a jtree was induced on the previous slides
 - Also known as *clique tree* (since the cycles often form cliques in the model graph) or join tree
 - Propositional version introduced by Lauritzen and Spiegelhalter (1988)
 - Shenoy and Shafer (1989) introduce three axioms of *local computations* to show correctness of doing computations locally

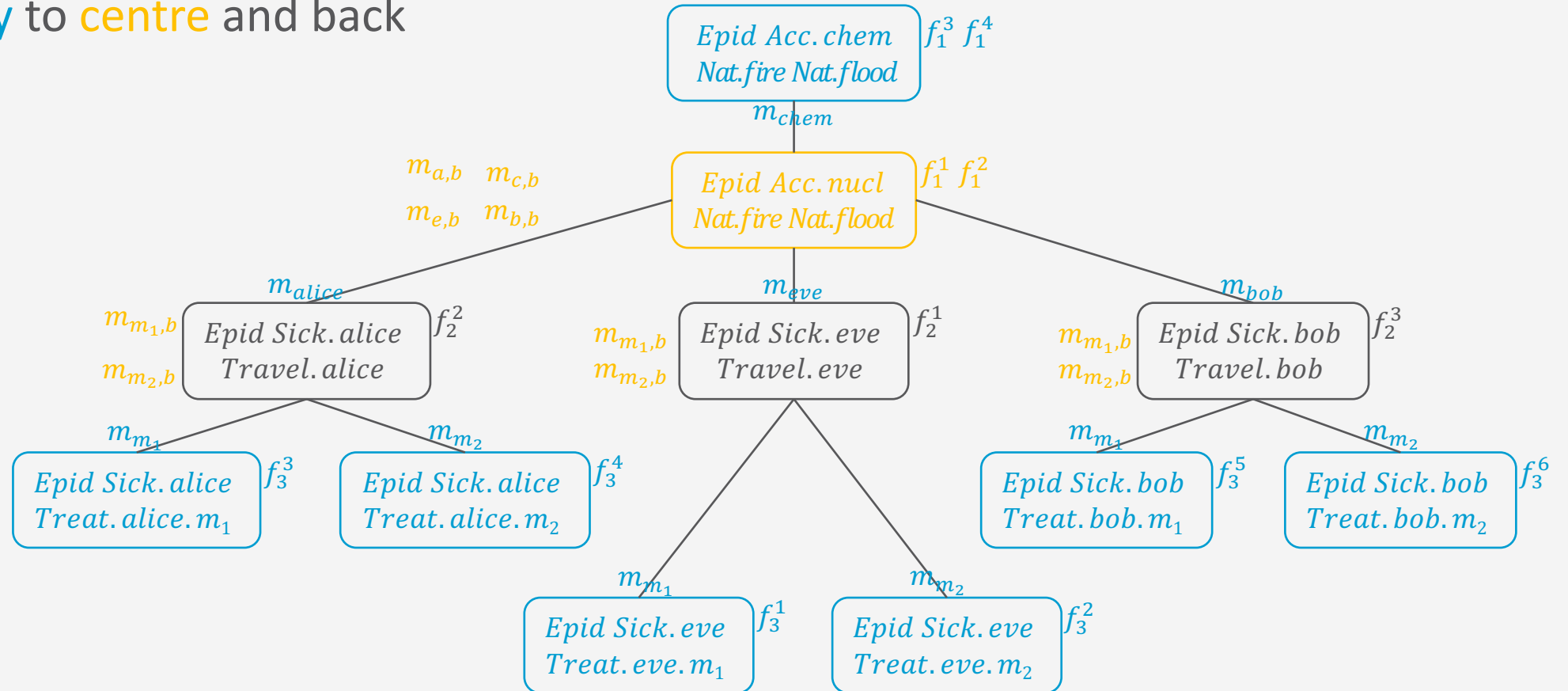
Comparison to Ground Inference

- Propositional Junction Tree Algorithm (JT)
 - Same algorithm,
 - only with propositional model
 - E.g., $gr(G)$



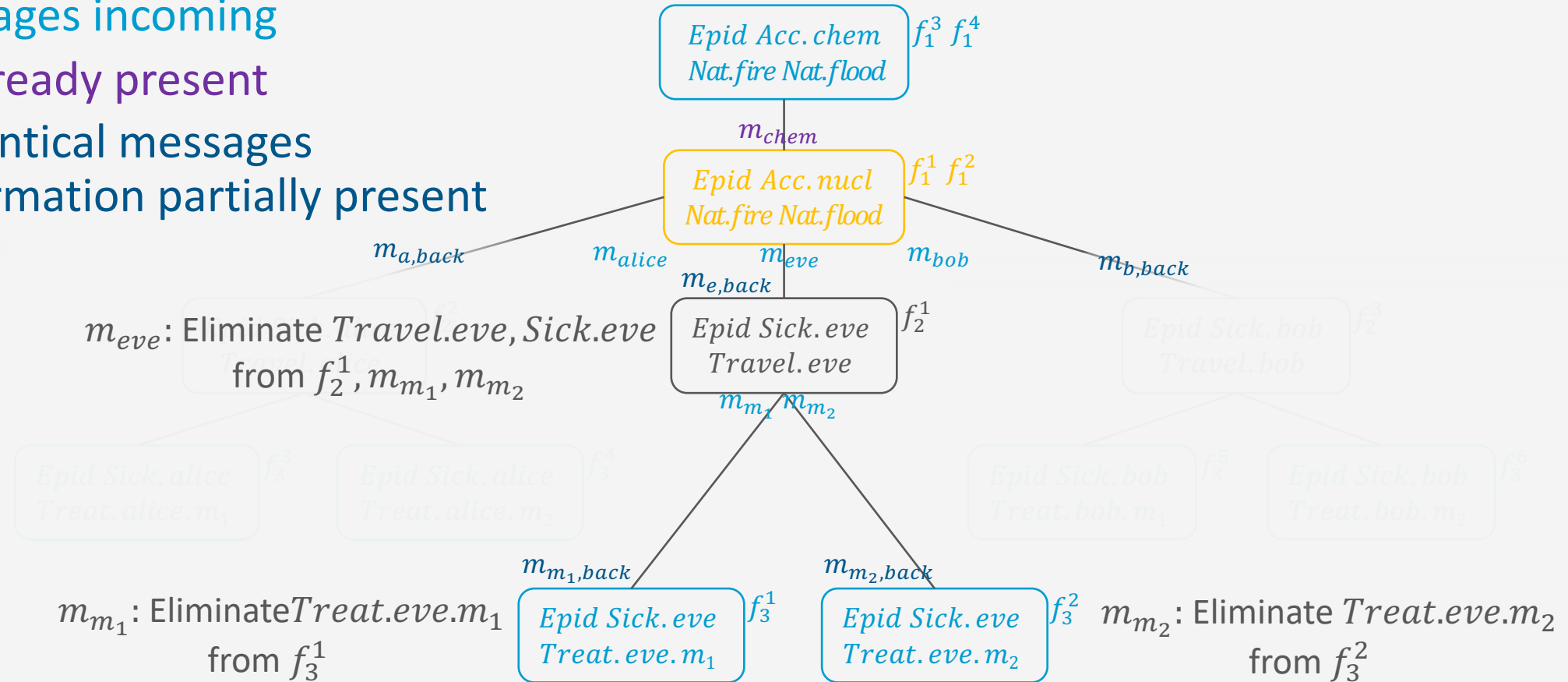
Junction Tree: Messages

- From **periphery** to **centre** and back
 - Inbound
 - Outbound



Junction Tree: Symmetry → Inefficiency

- Identical messages incoming
- Information already present
- Calculating identical messages
+ sending information partially present



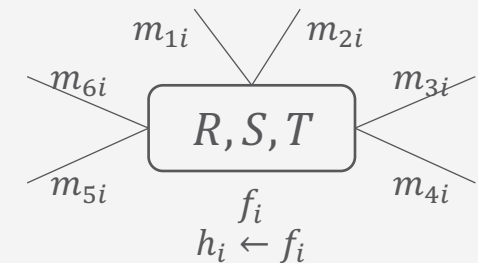
There is also a lifted version of Hugin using a lifted division operator
[Hoffmann et al., 2022]

Exact Inference: LJT

Message Calculation Strategies

- Strategy used so far: so-called **Shafer-Shenoy architecture** [Shafer and Shenoy, 1989]
 - Disadvantage: many operations (multiplications) duplicated
 - Especially in (FO) jtrees with high degree
 - Even if only one factor per parcluster and message
 - Example right: for each outgoing message, only one incoming message changes
- Alternative: **Hugin architecture** [Jensen et al., 1989]
 - Hugin factor $h_i = \phi_i(\mathcal{C}_i)$ per parcluster $\mathcal{C}_i$ as a product of G_i
 - Incoming messages m_{ji} multiplied into h_i (and stored): $h_i \leftarrow h_i \cdot m_{ji}$
 - When calculating message m_{ij} back: $\text{VE-JT}(f_i / m_{ji}, S_{ij}, \emptyset, .)$
 - Divide h_i by message m_{ji} , then sum out non-separators
 - One multiplication and one division instead of multiple multiplications

$$\begin{aligned} m_{i1} &\leftarrow \text{VE-JT}(\{f_i, m_{2i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, \dots) \\ m_{i2} &\leftarrow \text{VE-JT}(\{f_i, m_{1i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, \dots) \\ m_{i3} &\leftarrow \text{VE-JT}(\{f_i, m_{1i}, m_{2i}, m_{4i}, m_{5i}, m_{6i}\}, \dots) \\ m_{i4} &\leftarrow \text{VE-JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{5i}, m_{6i}\}, \dots) \\ m_{i5} &\leftarrow \text{VE-JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{6i}\}, \dots) \\ m_{i6} &\leftarrow \text{VE-JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{5i}\}, \dots) \end{aligned}$$



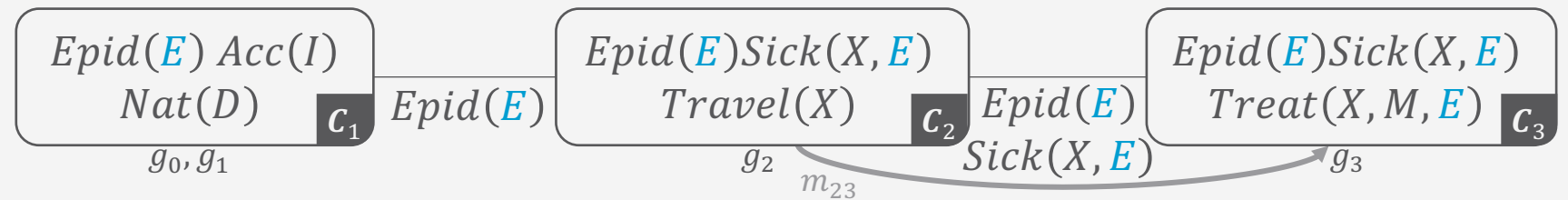
$$\begin{aligned} h_i &\leftarrow h_i \cdot m_{1i} & h_i / m_{1i} &\rightarrow \text{VE-JT} \\ h_i &\leftarrow h_i \cdot m_{2i} & h_i / m_{2i} &\rightarrow \text{VE-JT} \\ h_i &\leftarrow h_i \cdot m_{3i} & h_i / m_{3i} &\rightarrow \text{VE-JT} \\ h_i &\leftarrow h_i \cdot m_{4i} & h_i / m_{4i} &\rightarrow \text{VE-JT} \\ h_i &\leftarrow h_i \cdot m_{5i} & h_i / m_{5i} &\rightarrow \text{VE-JT} \\ h_i &\leftarrow h_i \cdot m_{6i} & h_i / m_{6i} &\rightarrow \text{VE-JT} \end{aligned}$$

In terms of Lifting: Is it that simple?

- Algorithm-induced **groundings** due to message passing
 - For message calculation, non-separator PRVs are eliminated
 - Separator as the query terms containing *logical variables*
 - Non-separator PRVs have to fulfil sum-out preconditions
 - Preconditions 1 + 3 fulfilled by construction
 - Maybe that Precondition 2 is **not** fulfilled $\rightarrow$ can cause groundings
- E.g., logical variable **E** added to PRVs $Epid, Sick(X), Treat(X, M)$
 - When calculating m_{23} , one has to eliminate $Travel(X)$
 - But: does not contain both X and **E**, count conversion does not apply (**E** occurs in two PRVs) $\rightarrow$ **ground E**

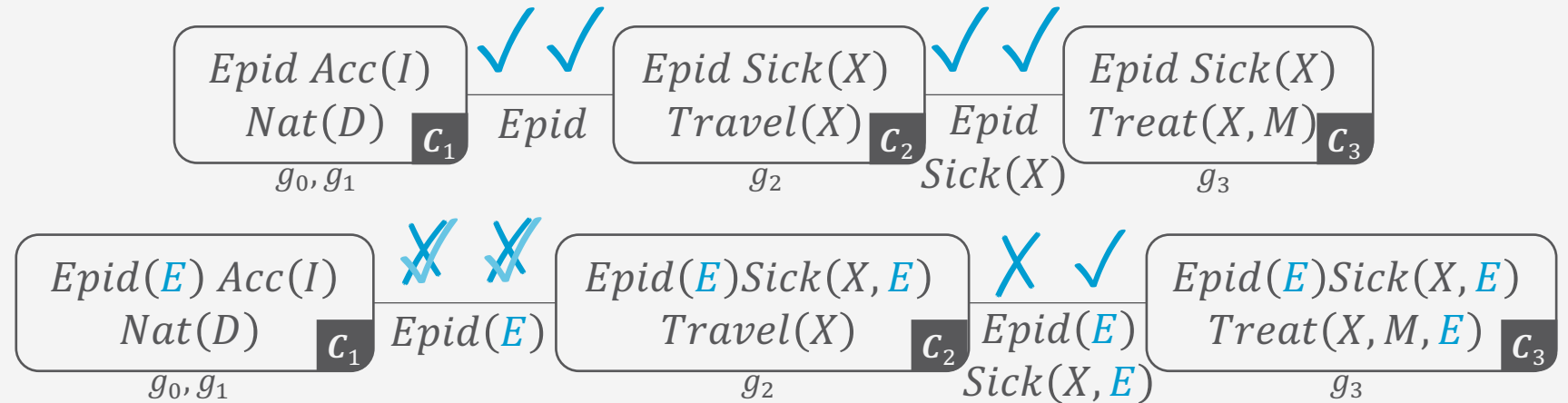
Preconditions:

- $\forall B \in rv(G \setminus \{g\}) : gr(B|_C) \cap gr(A_i|(X, C_X)) = \emptyset$
- $\forall X \in \{X \mid |\pi_X(C_X)| > 1\} : X \in lv(A_i)$
- $X^{excl} = lv(A_i) \setminus (X \setminus lv(A_i))$ count-normalised w.r.t



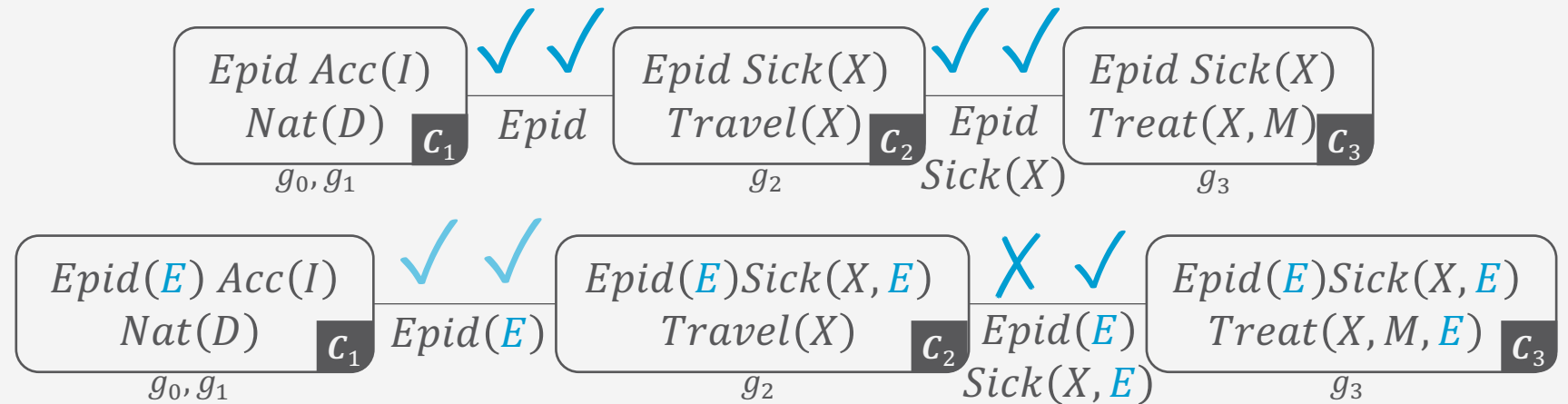
Conditions on Groundings

- For a lifted calculation of message m_{ij} , it necessarily has to hold that
 - for each PRV $A \in (C_i \setminus S_{ij})$, i.e., A has to be eliminated:
 - for each separator PRV $S \in S_{ij} : lv(S) \subseteq lv(A)$ (Cond. 1)
- If Cond. 1 does not hold, i.e., $lv(S) \not\subseteq lv(A)$, one may induce Cond. 1 by count conversion
 - If $lv(S) \setminus lv(A)$ are countable in G_{ij} (Cond. 2)



Conditions on Groundings

- Problem with Cond. 1 induced using count conversions on logical variables $lv(S) \setminus lv(A)$:
 - Logical variables that were previously not counted are now counted
 - All receiving parclusters need to be able to handle counted versions, which needs to be checked
 - If newly counted logical variable arrives at parcluster C_k , it has to be **countable in G_k** as well (Cond. 3)
 - For further calculations, since they refer to the same set of randvars, they have to occur in the same form, i.e., at one point the logical variable has to be counted in G_k as well

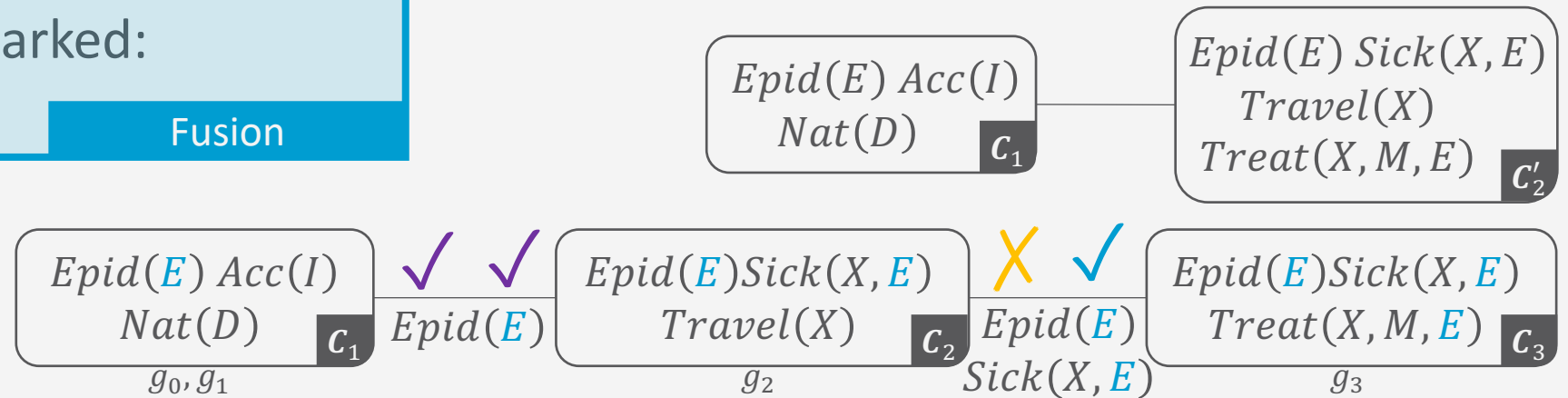


Fusion

- Test each message m_{ij} for each PRV A to eliminate and each separator PRV S
 - If Cond. 1 holds: continue (no groundings)
 - Else if Cond. 2 and Cond. 3 holds: continue
 - Else: mark m_{ij} (**grounding**); continue with next m_{ij}
- For each message m_{ij} marked:
 - Merge parclusters C_i, C_j

Fusion

- Fusion an additional step after construction to guarantee lifted calculations for liftable models
- E.g., testing marks m_{23}
→ merge C_2, C_3 (as in minimisation)



LJT: Complexity

- Uses also the notion of lifted width $w_T = (w_g, w_{\#})$
 - w_g largest ground width
 - $w_{\#}$ largest counting width
 - As FO jtree constructed from FO dtree, w_T identical between LVE and LJT
 - Fusion may change w_T in terms of the FO jtree
 - But in terms of the LVE calculations in the merged parcluster, w_T is still the same with multiple nodes being combined into one
 - For simplicity, let us consider models that all fulfil Cond. 1 in fusion such that w_T is identical for both LJT and LVE

LJT: Complexity

- LJT complexity based on complexity of LVE:

$$O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$$
- Complexity of individual steps
 - Construction: linear in number of nodes, no calculations; negligible compared to later steps
 - Evidence entering: $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$
 - Absorbing evidence complexity: $O(\log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#} w_{\#}})$
 - Visits $\frac{1}{r} \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}}$ lines, possibly exponentiates the potentials
 - At each node $\rightarrow n_J \cdot O(\log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#} w_{\#}})$
 - n_J number of nodes in FO jtree J
 - For each e evidence parfactors $\rightarrow e \cdot O(n_J \cdot \log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#} w_{\#}})$
 - Assuming $e \ll n_J \rightarrow O(n_J \cdot \log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#} w_{\#}})$
 - First two steps accumulated: $O(n_J \cdot \log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#} w_{\#}})$

LJT: Complexity

- Complexity of individual steps
 - First two steps accumulated: $O(n_J \cdot \log_2(n) \cdot r^{w_g-1} \cdot n^{r_{\#}w_{\#}})$
 - Message passing: $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
 - Calculating one message = answering one query on a parcluster
 - Worst-case parfactor size at parcluster: $r^{w_g} \cdot n^{r_{\#}w_{\#}}$
 - Elimination of $|\mathcal{C}_i \setminus \mathcal{S}_{ij}|$ PRVs goes through each line, potentials may be exponentiated $\rightarrow O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
 - Two messages per edge, $n_J - 1$ edges in $J \rightarrow n_J \cdot O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
 - Query answering: $O(m \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
 - Each query answered in one parcluster $\rightarrow O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
 - With m query terms $\rightarrow m \cdot O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}})$
- All four steps accumulated:

$$O\left((n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#}w_{\#}}\right)$$

Comparison to LVE

- LVE complexity of one query = LJT complexity of message passing
 - $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$ vs. $O(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$
 - Actual number of calculations:
 - In LVE: c_{LVE}
 - For message pass: $2 \cdot c_{LVE}$
- For m queries
 - LVE: $O(m \cdot n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$
 - LJT: $O((n_J + m) \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$
 - Difference in $m \cdot n_T$ vs. $(n_J + m)$
 - LVE has complexity of $O(n_T \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$ for one query
 - LJT only complexity of $O(\log_2(n) \cdot r^{w_g} \cdot n^{r_{\#} w_{\#}})$ for one query

LJT only pays off if $m > 1$, most likely, starting with third query (two queries in LVE = one message pass)