WWU MÜNSTER

Wir tragen Maske!
We wear masks!

Lifted Inference: Exact Inference

Statistical Relational Artificial Intelligence (StaRAI)

living.knowledge

Tanya Braun

Contents

Outline: 4. Lifted Inference

A. *Exact Inference*

i. Lifted Variable Elimination for Parfactor Models
   - Idea, operators, algorithm, complexity

ii. Lifted Junction Tree Algorithm
   - Idea, helper structure: junction tree, algorithm

iii. First-order Knowledge Compilation for MLNs
   - Idea, helper structure: circuit, algorithm

B. *Approximate Inference: Sampling*

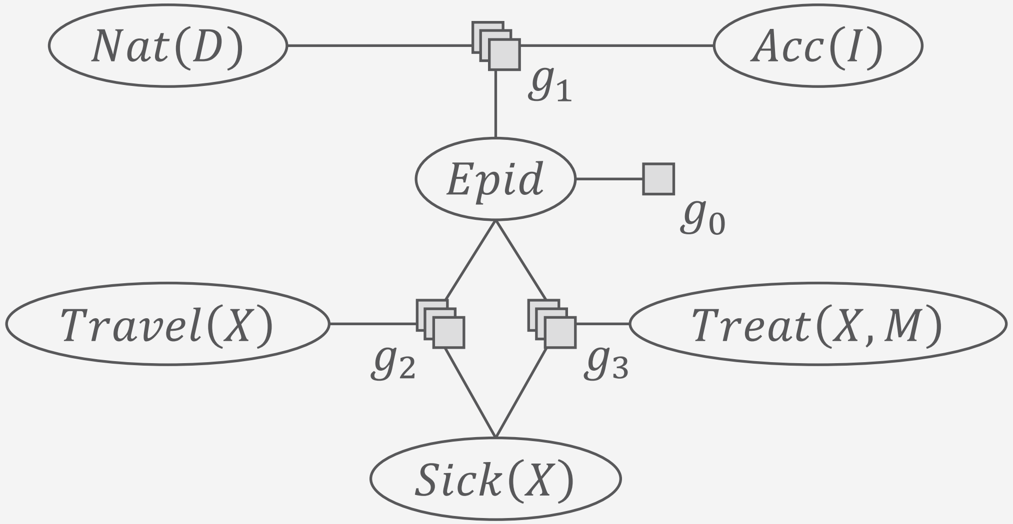
- Rejection sampling
- (Lifted) likelihood sampling
- (Lifted) Markov Chain Monte Carlo sampling

Problem: Many Queries

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

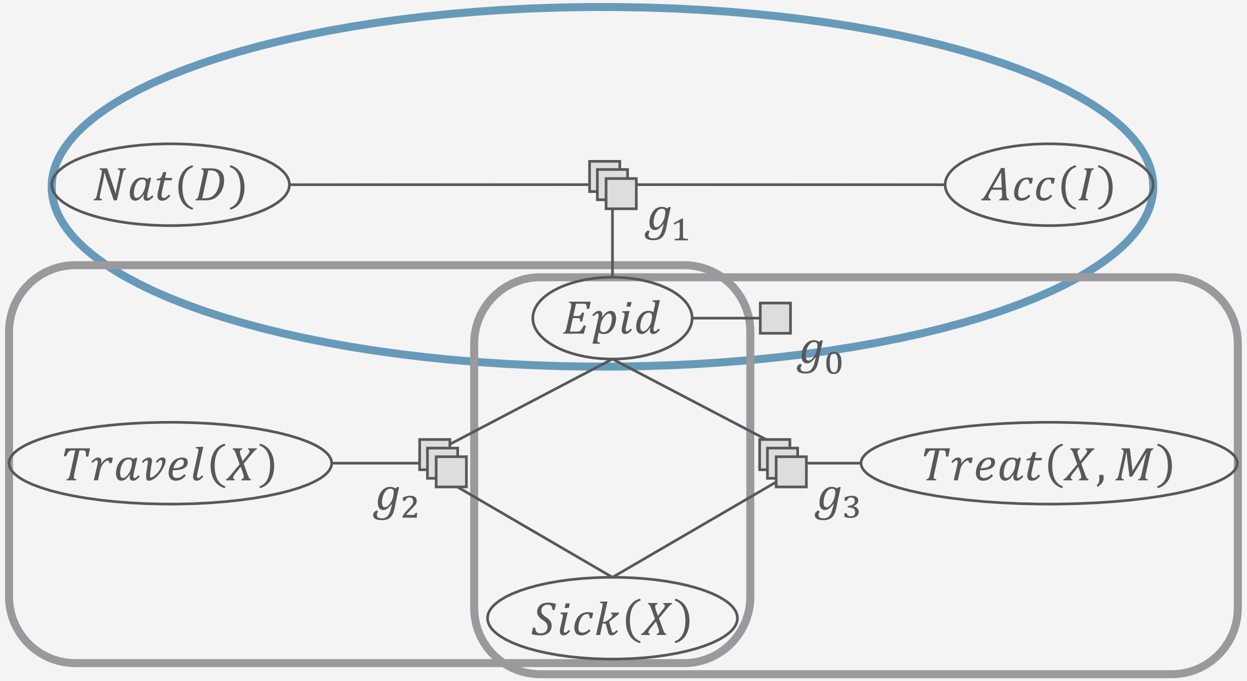
- LVE restarts with initial model for each query

Build a helper structure to precompute parts



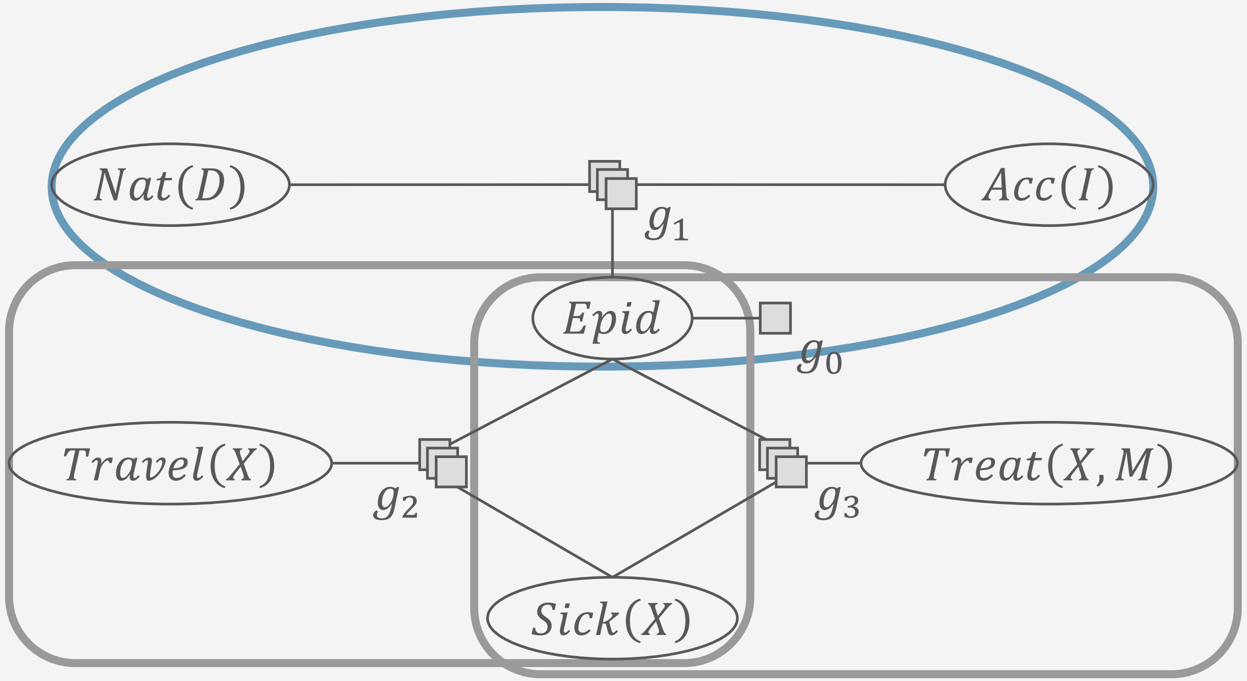
Clustering of Models

- 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



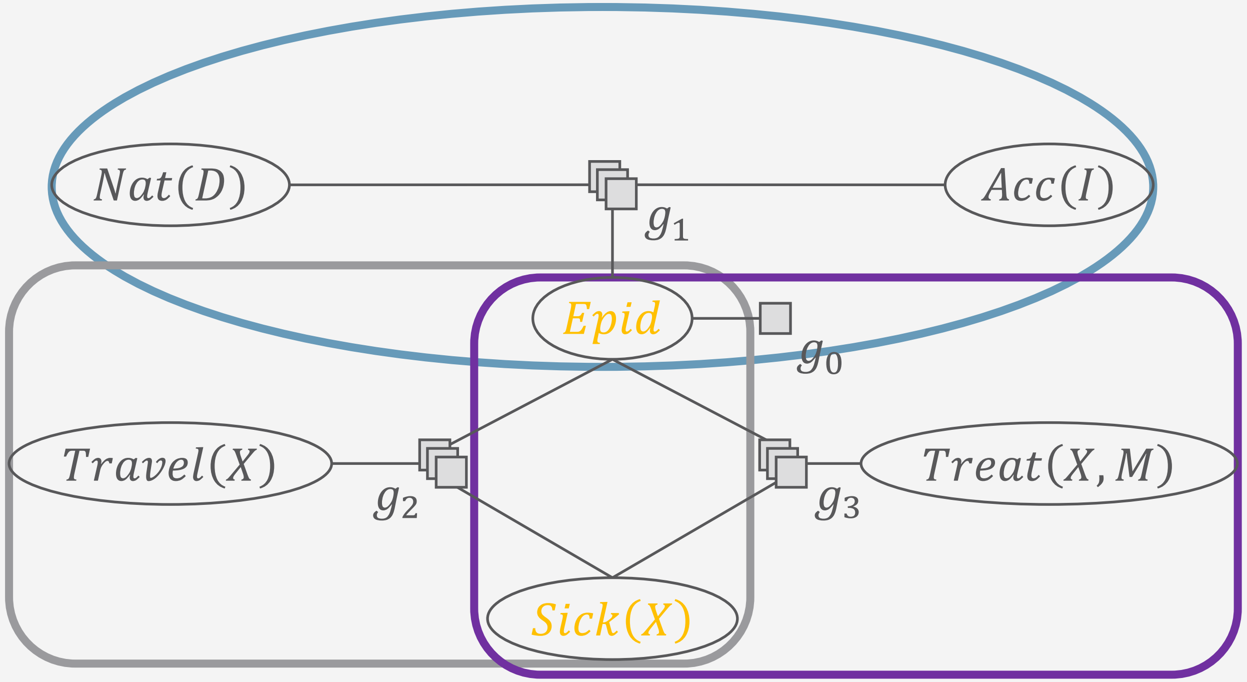
Clustering of Models

- 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?



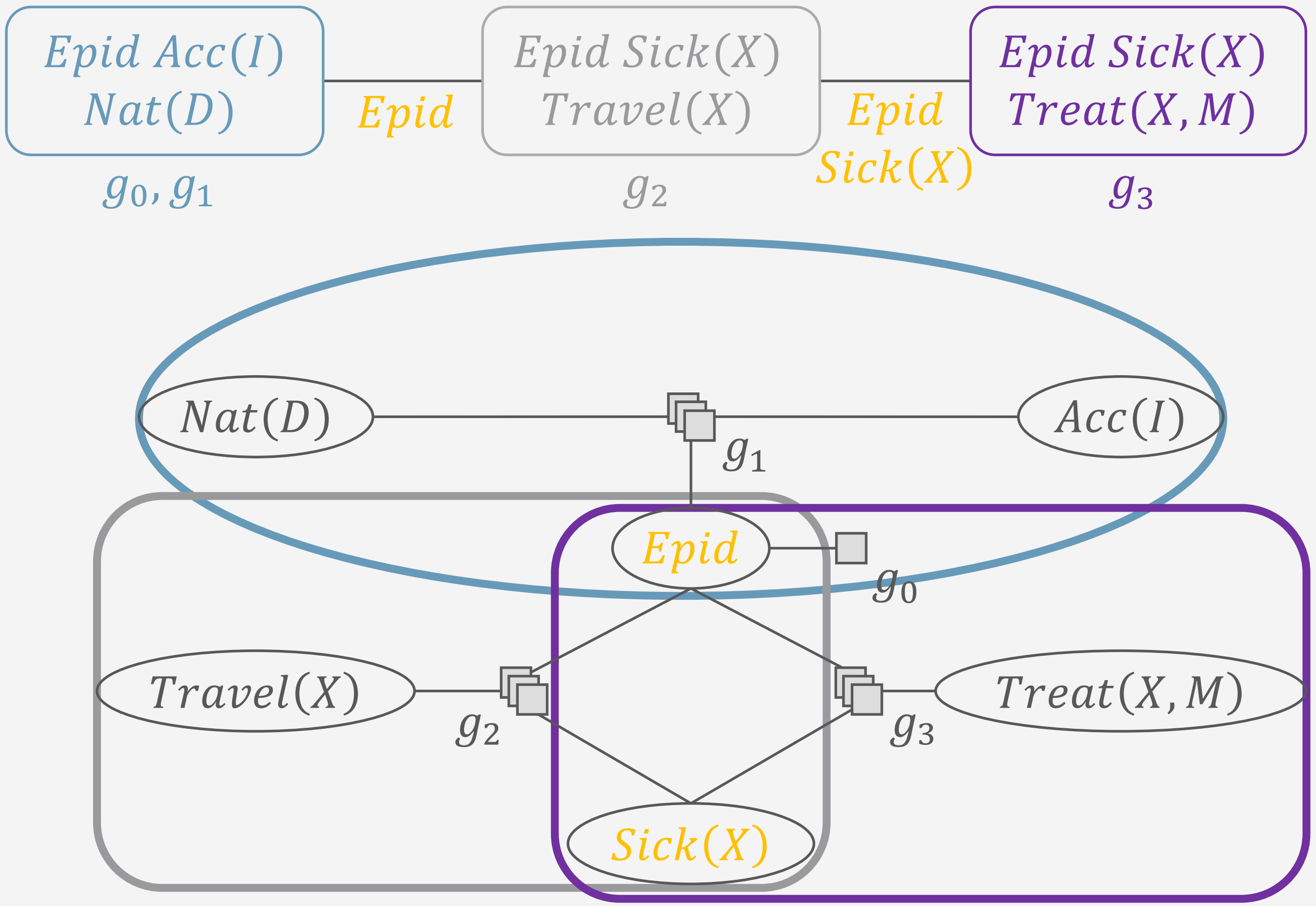
Clustering of Models

- Factorised models encode independences:
  - Any two subsets of variables are conditionally independent given a *separating* subset $\boldsymbol{S}$
    - *Separating* subset $\boldsymbol{S}$: All paths from one subset to the other run through $\boldsymbol{S}$
      - 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 $Epid, Sick(X)$

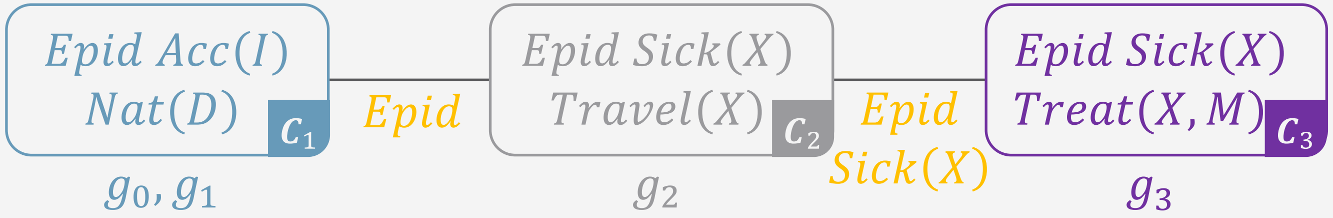


Clustering of Models

- 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

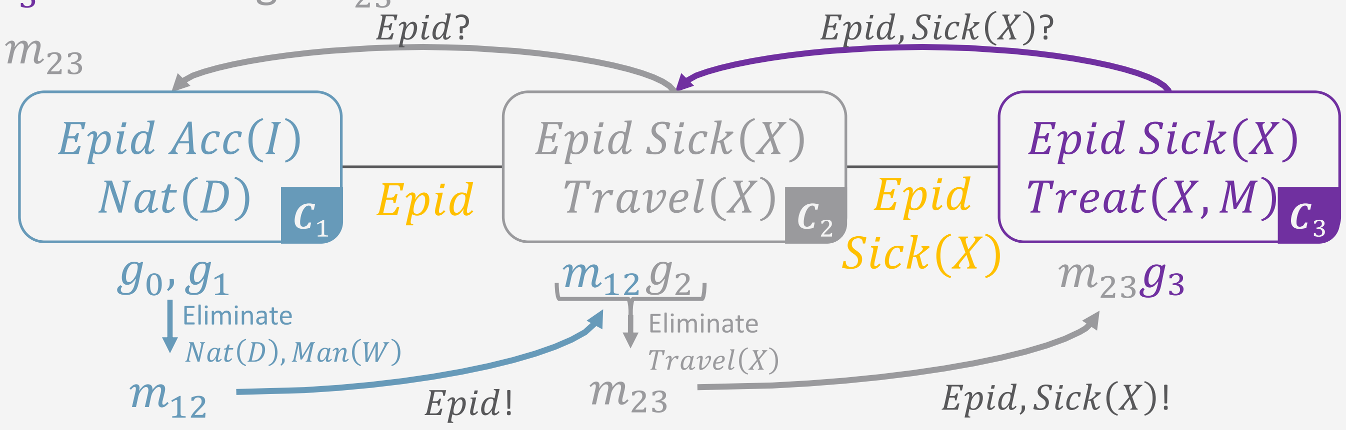


- Next: Make clusters actually independent of each other
  - Each cluster i asks its neighbours $j \in nbs(i)$ for information about the separator $\boldsymbol{S}_{ij}$ between them
    - Other clusters have to collect all the information from the model that lies behind the separator on its part, eliminate 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., $\boldsymbol{C}_3$: g_3 → independent of the rest given $Epid, Sick(X)$
    - Asks neighbour C_2 for information on $Epid, Sick(X)$
      - C_2 asks neighbour $\boldsymbol{C}_1$ for information on $Epid$
      - $\boldsymbol{C}_1$ sends information on $Epid$ in a message m_{12}
      - Eliminates $Nat(D), Acc(I)$ from g_0, g_1 for m_{12}
      - C_2 sends information on $Epid, Sick(X)$ to $\boldsymbol{C}_3$ in a message m_{23}
      - Eliminates $Travel(X)$ from g_2 and m_{12} for m_{23}
    - With m_{23}, $\boldsymbol{C}_3$ is independent from its neighbour C_2 and therefore also from $\boldsymbol{C}_1$
      - As C_2 is independent given m_{12} from $\boldsymbol{C}_1$

The same has to be done for C_2 and C_1

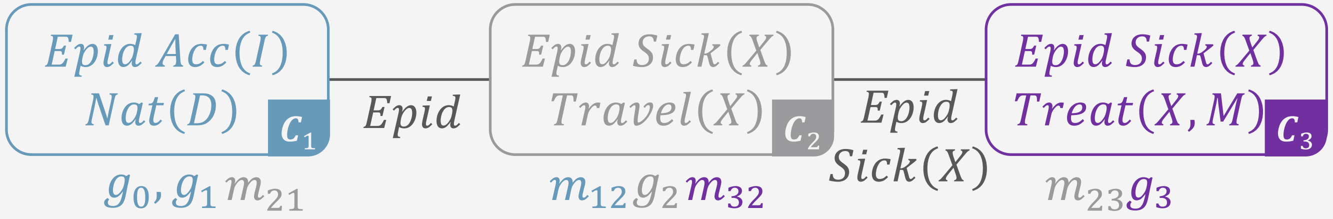


Clustering of Models

- 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., $\boldsymbol{C}_3$: g_3 → independent of the rest given $Epid, Sick(X)$
    - Based on g_3 and m_{23}, $\boldsymbol{C}_3$ can answer queries about $Epid, Sick(X), Treat(X, M)$ such as

$$P\big(Sick(X)\big),$$
$$P\big(Treat(eve, m_2)\big),$$
$$P\big(Epid, Sick(alice)\big)$$

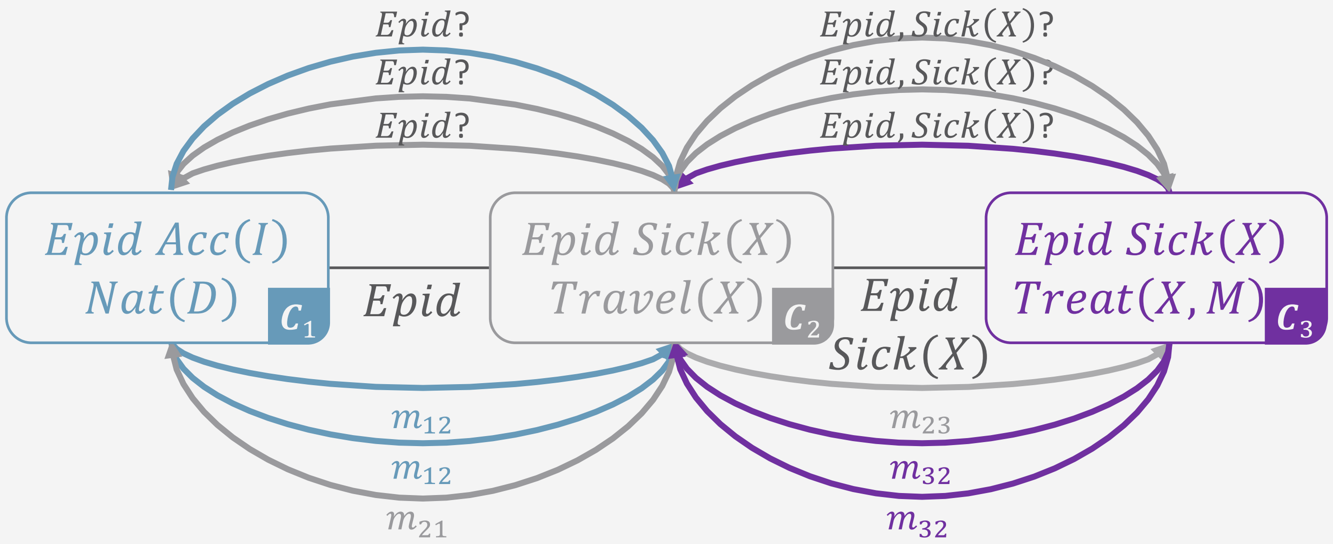
      - Cannot answer any queries about $Nat(D), Acc(I), Travel(X)$ but $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, respectively, can



Clustering of Models

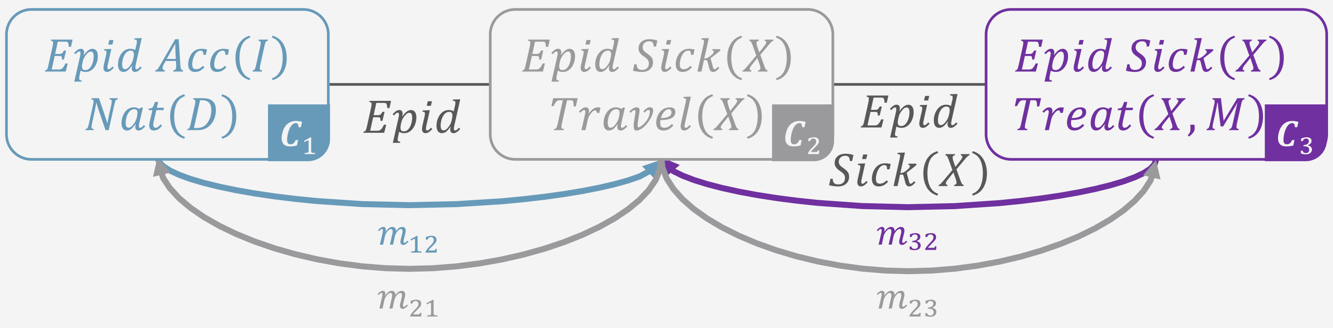
- Problem left: If each cluster asks for information on separators, some messages are sent multiple times
  - E.g.,
    - $\boldsymbol{C}_3$ asks C_2, which asks $\boldsymbol{C}_1$
      - Messages calculated and sent: m_{12}, m_{23}
    - C_2 asks $\boldsymbol{C}_1$ and $\boldsymbol{C}_3$
      - Messages calculated and sent: m_{12}, m_{32}
    - $\boldsymbol{C}_1$ asks C_2, which asks $\boldsymbol{C}_3$
      - Messages calculated and sent: m_{32}, m_{21}

Organise in way that messages are calculated only once



Clustering of Models

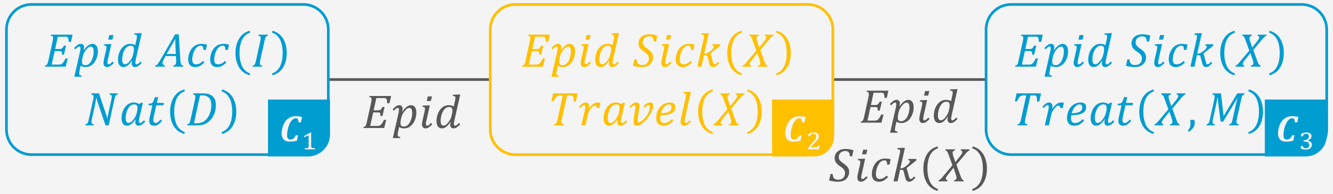
- Use dynamic programming to organise the order of asking or rather sending information in messages:
  - → If a node i has received all information from neighbours but one, j, node i sends a message with its information on the separator $\mathbf{S}_{ij}$ to j
  - → If a node i has received all messages, 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 $\boldsymbol{S}_{ij}$ 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



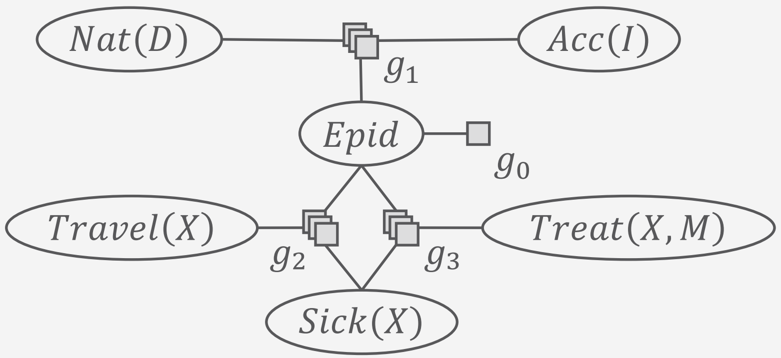
Two passes from periphery to centre and back suffice to distribute all information and make the clusters independent from each other*

* Shown by Steffen L. Lauritzen and David J. Spiegelhalter: Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. In: *Journal of the Royal Statistical Society. Series B: Methodological*, 1988.

Lifted Junction Tree Algorithm (LJT)

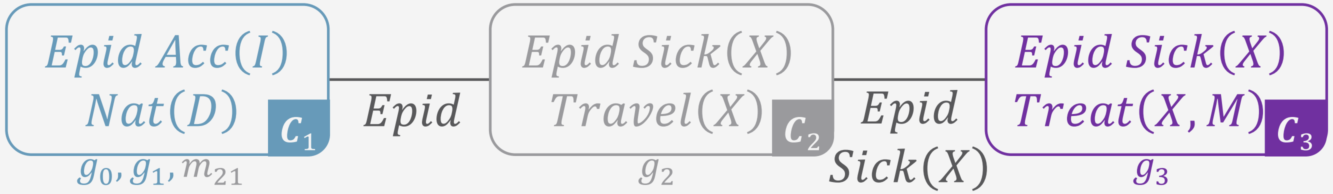
- Inputs
  - Model G
  - Evidence $\boldsymbol{e}$ as evidence parfactors
  - Set of query terms $\{\boldsymbol{Q}_i\}_{i=1}^{m}$
    - Queries on G: $P(\boldsymbol{Q}_i \mid \boldsymbol{e}), i \in \{1, \dots, m\}$

- LJT consists of four steps
  1. Build FO jtree J for model G
  2. Enter evidence $\boldsymbol{e}$ in J
  3. Pass messages in J
  4. Answer queries $\{\boldsymbol{Q}_i\}_{i=1}^{m}$



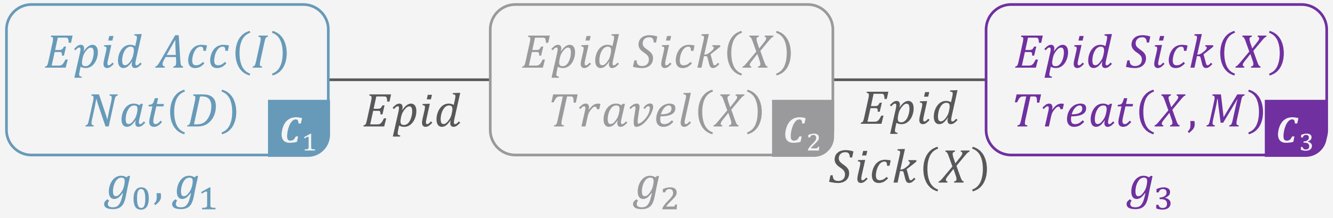
Evidence:
$sick(eve)$

Queries:
$\{\{Epid\}, \{Travel(eve), Treat(eve, m_1)\}\}$



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 $\boldsymbol{X}$ be a set of logical variables, $\boldsymbol{A}$ a set of PRVs with $lv(\boldsymbol{A}) \subseteq \boldsymbol{X}$, and $(\mathcal{X}, C_{\mathcal{X}})$ a constraint on $\boldsymbol{X}$
- Then, a parcluster $\boldsymbol{C}$ is given by

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

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

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

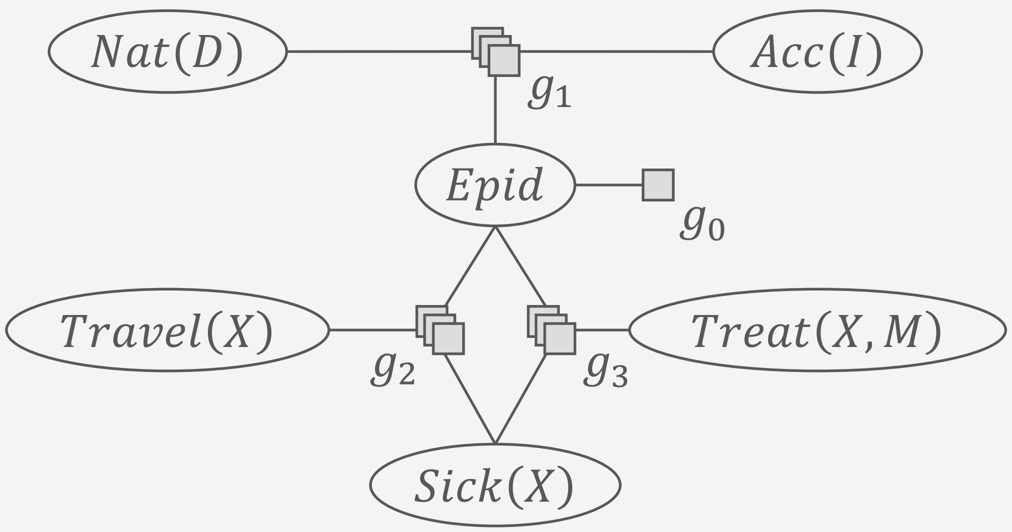
$$= \{Epid, Sick(X), Travel(X)\}_{|(X, \mathcal{D}(X))}$$

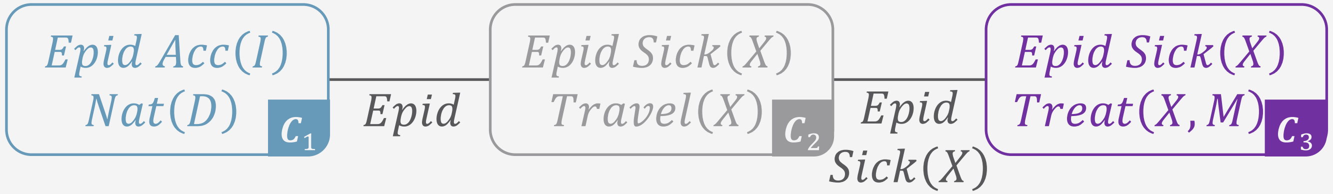
$$= \{Epid, Sick(X), Travel(X)\}$$

$Epid\ Sick(X)$
$Travel(X)$

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

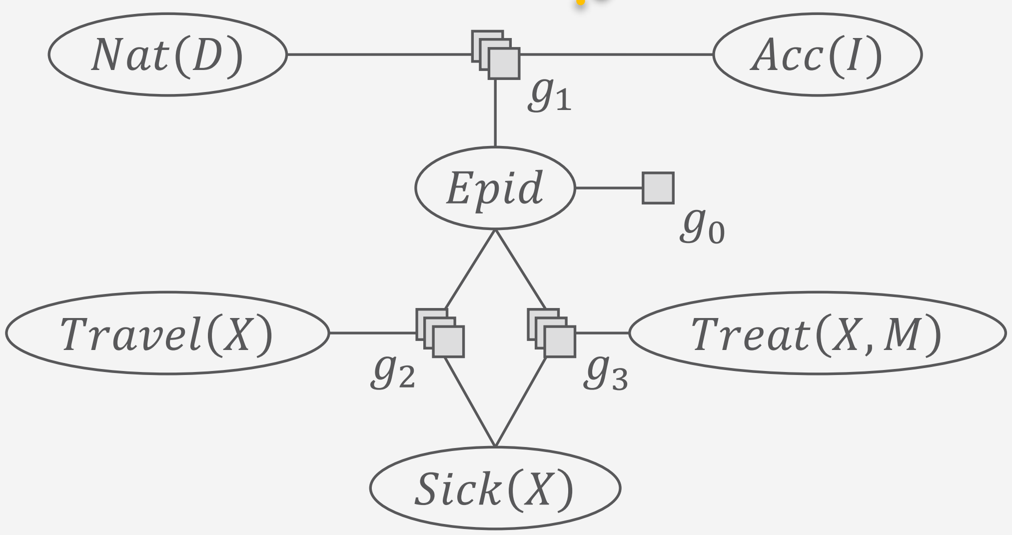


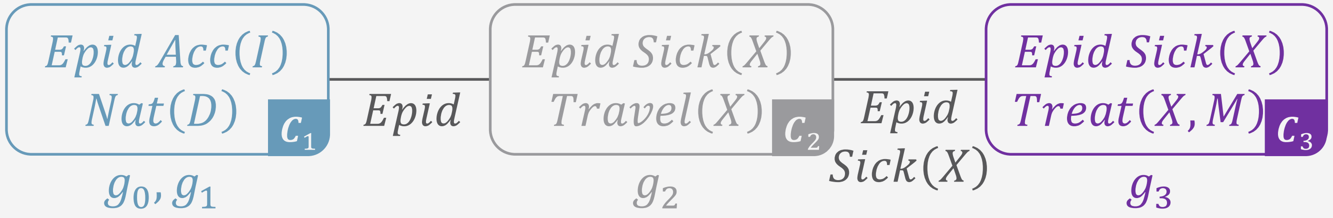


FO Jtree

Other valid FO jtrees to build?

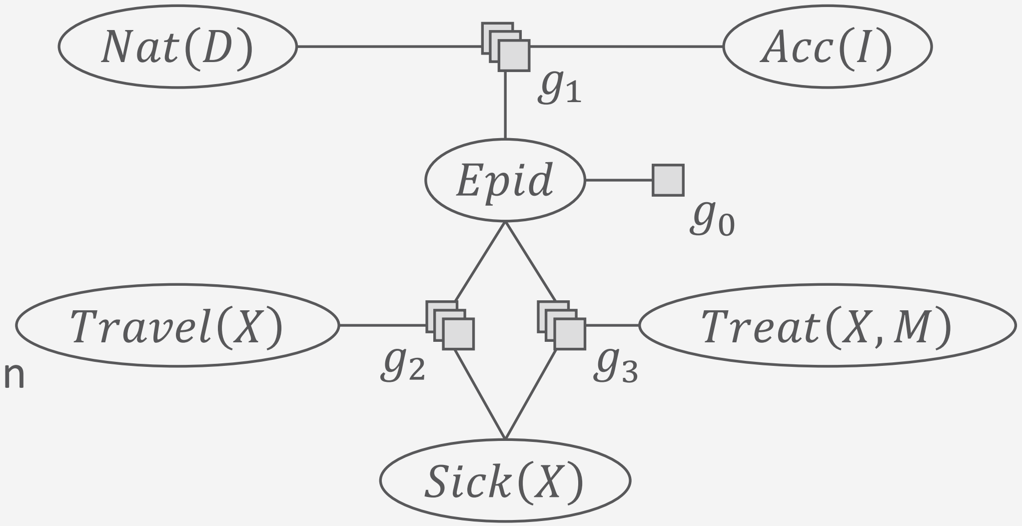
- An FO jtree J for a model G is a cycle-free graph (V, E)
  - Has to satisfy three properties:
    1. $\forall \boldsymbol{C} \in V : \boldsymbol{C} \subseteq rv(G)$
       - Every parcluster consists of PRVs from G
    2. $\forall g \in G : \exists \boldsymbol{C} \in V : rv(g) \subseteq \boldsymbol{C}$
       - Arguments of every parfactor in G occur in a parcluster
    3. If $\exists A \in rv(G) : A \in \boldsymbol{C}_i \land A \in \boldsymbol{C}_j$ with $\boldsymbol{C}_i, \boldsymbol{C}_j \in V$, then $\forall \boldsymbol{C}_k \in V$ on the path between $\boldsymbol{C}_i, \boldsymbol{C}_j : A \in \boldsymbol{C}_k$ (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

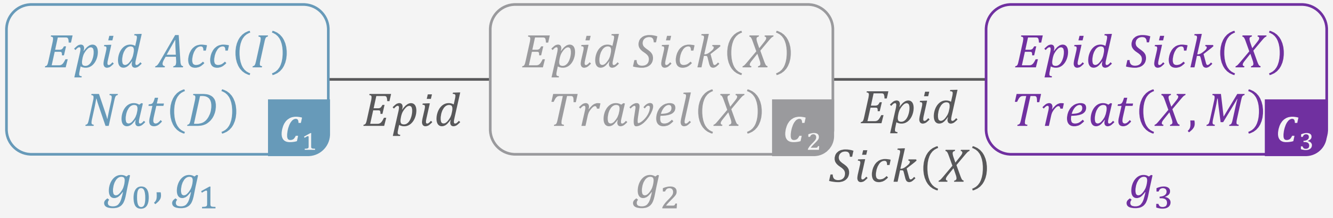




FO Jtree

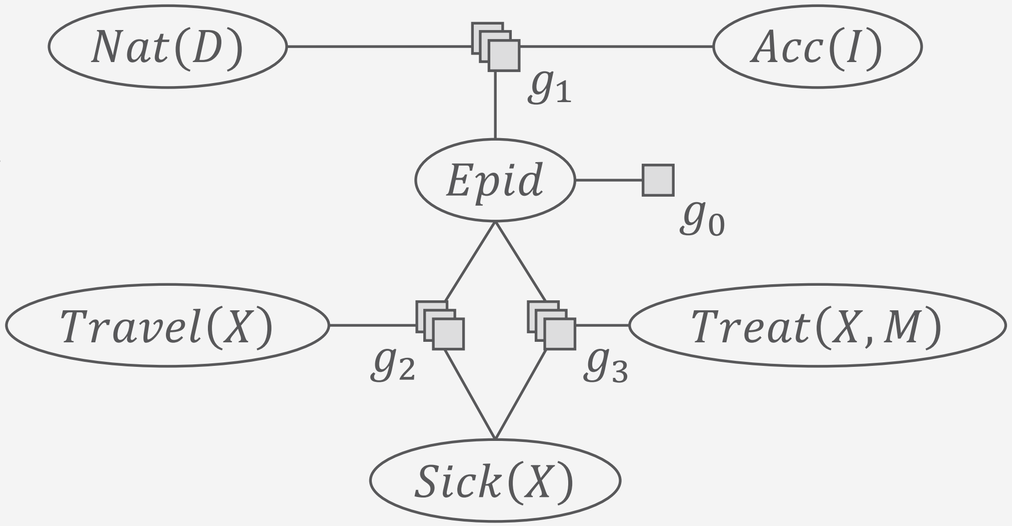
- An FO jtree J for a model G is a cycle-free graph (V, E)
  - Is minimal if by removing a PRV from a parcluster, the FO jtree ceases to be an FO jtree
    - I.e., no longer fulfils at least one property
  - E.g., depicted on the left
    - Cannot remove any PRV from any parcluster
      - Otherwise,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

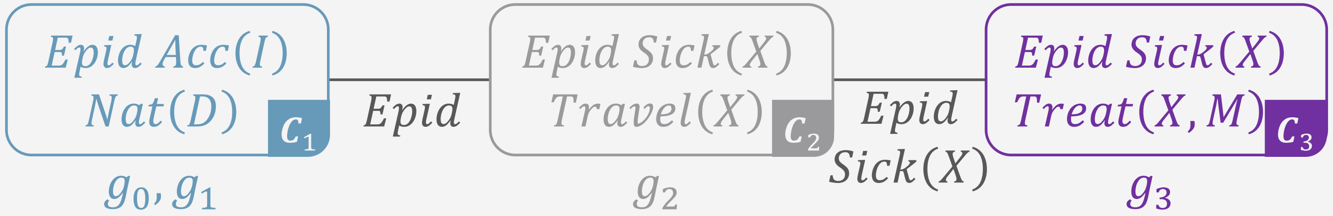




FO Jtree

- An FO jtree J for a model G is a cycle-free graph (V, E)
  - Set $\boldsymbol{S}_{ij}$ called separator of edge $\{i, j\} \in E$, defined by
    $$\boldsymbol{S}_{ij} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$$
  - Term $nbs(i)$ refers to the neighbours of $\boldsymbol{C}_i$, defined by
    $$nbs(i) = \{j \mid \{i, j\} \in E\}$$
  - Each $\boldsymbol{C}_i$ has a local model G_i and
    $$\forall g \in G_i : rv(g) \subseteq \boldsymbol{C}_i$$
    - Local models G_i partition G, i.e.,
    $$G = \bigcup_{i \in V} G_i, \forall i, j \in V, i \neq j : G_i \cap G_j = \emptyset, G_i \neq \emptyset$$

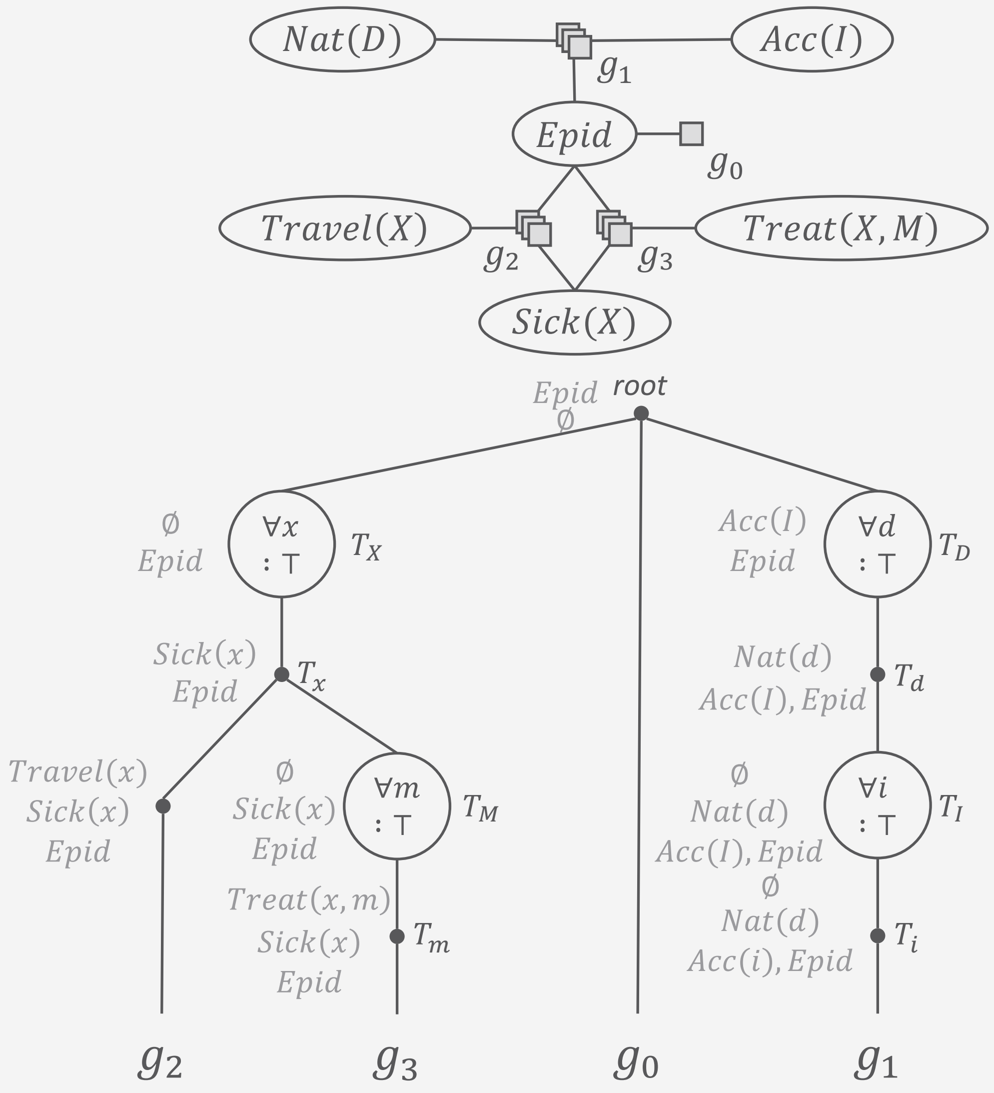


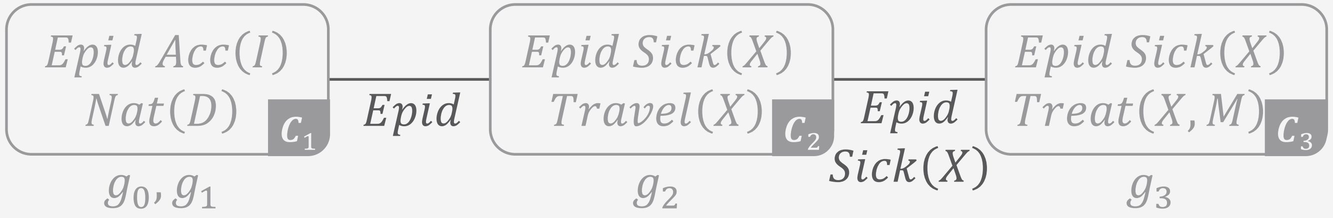


Construction

- 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?

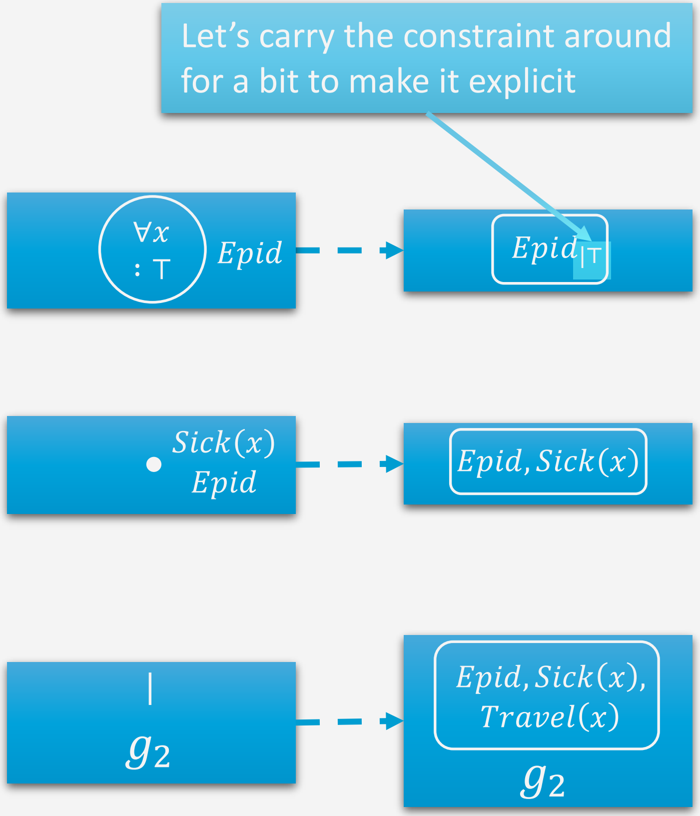
→ Clusters of an FO dtree
+ undirected dtree edges
+ minimisation
= FO jtree





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 $\boldsymbol{C}_j = \{A_1, \dots, A_n\}_{|C}$
    - Local model $G_j = \emptyset$
- Given a cluster $\{A_1, \dots, A_n\}$ of a VE node
  - Resulting parcluster $\boldsymbol{C}_j = \{A_1, \dots, A_n\}_{|\top}$
    - Local model $G_j = \emptyset$
- Given a cluster $\{A_1, \dots, A_n\}$ from a leaf node with parfactor g_i
  - Resulting parcluster $\boldsymbol{C}_j = \{A_1, \dots, A_n\}_{|\top}$
    - Local model $G_j = \{g_i\}$

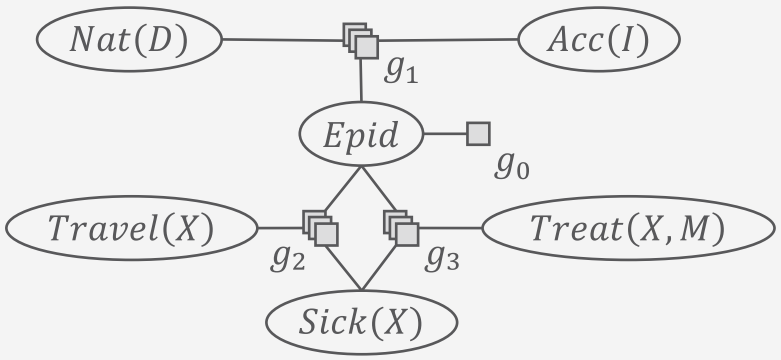


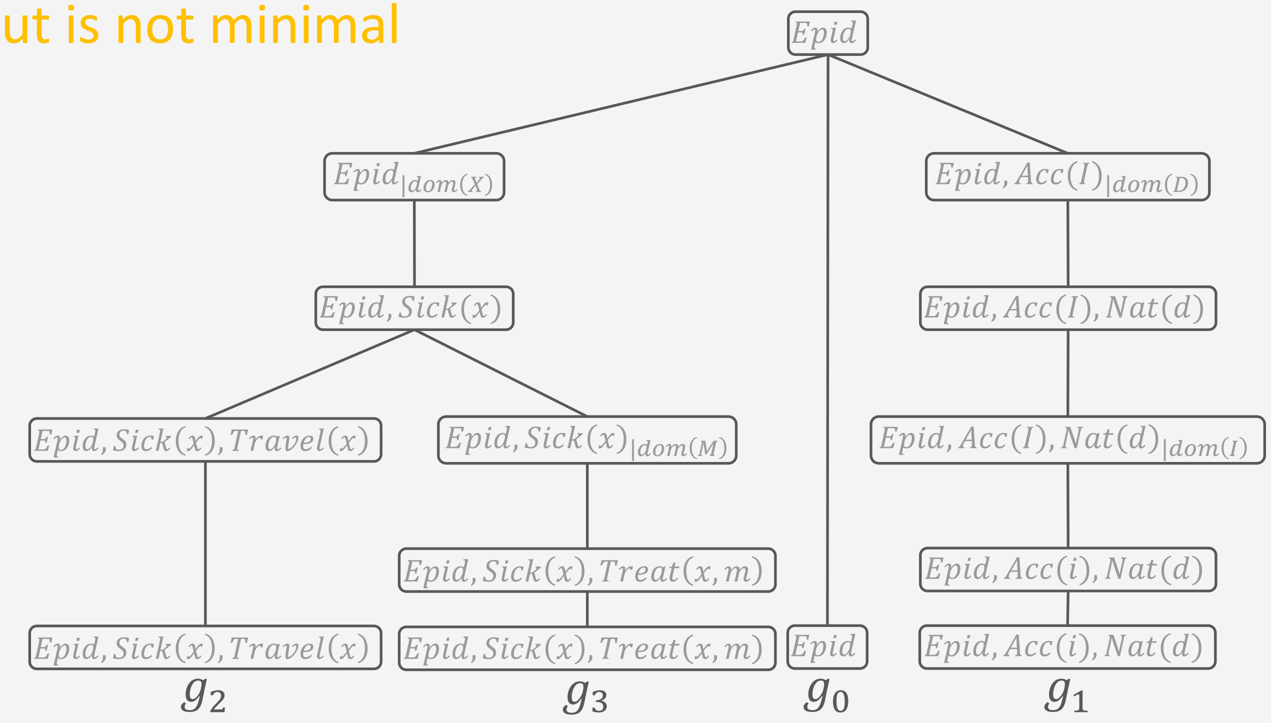
FO Dtree → FO Jtree

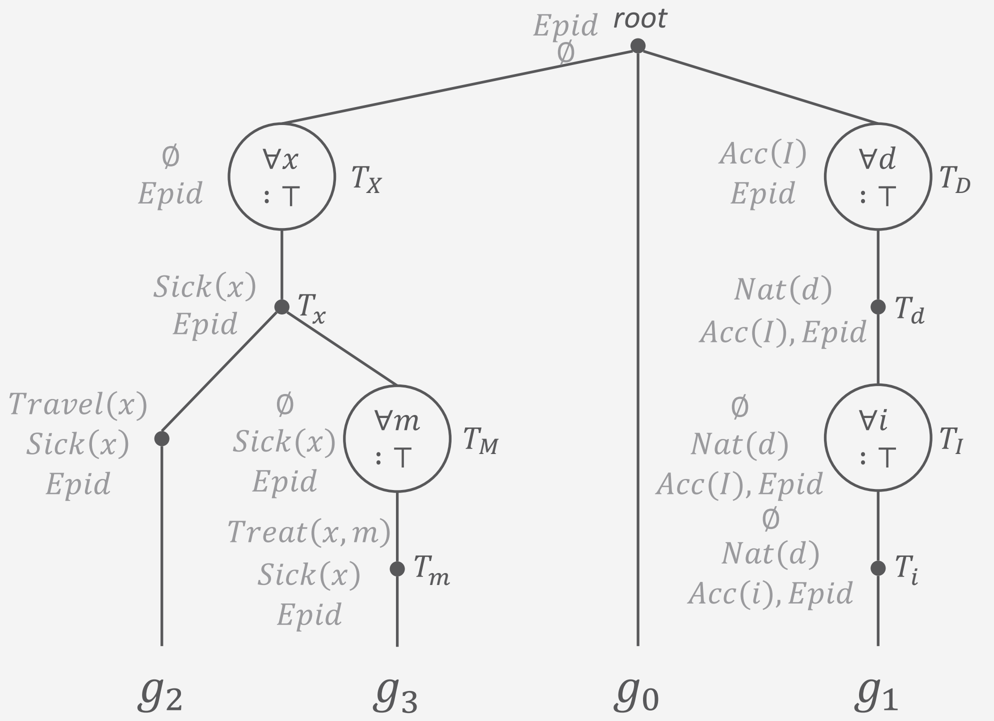
- Forming an FO jtree J from an FO dtree T of a model G
- Nodes of J
  - Parclusters resulting from clusters of T as shown on previous slide
    - Each parcluster has a source node in T
- Edges of J
  - Add an edge between two parclusters whenever there is an edge between the source nodes of the two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

FO Dtree → FO Jtree

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



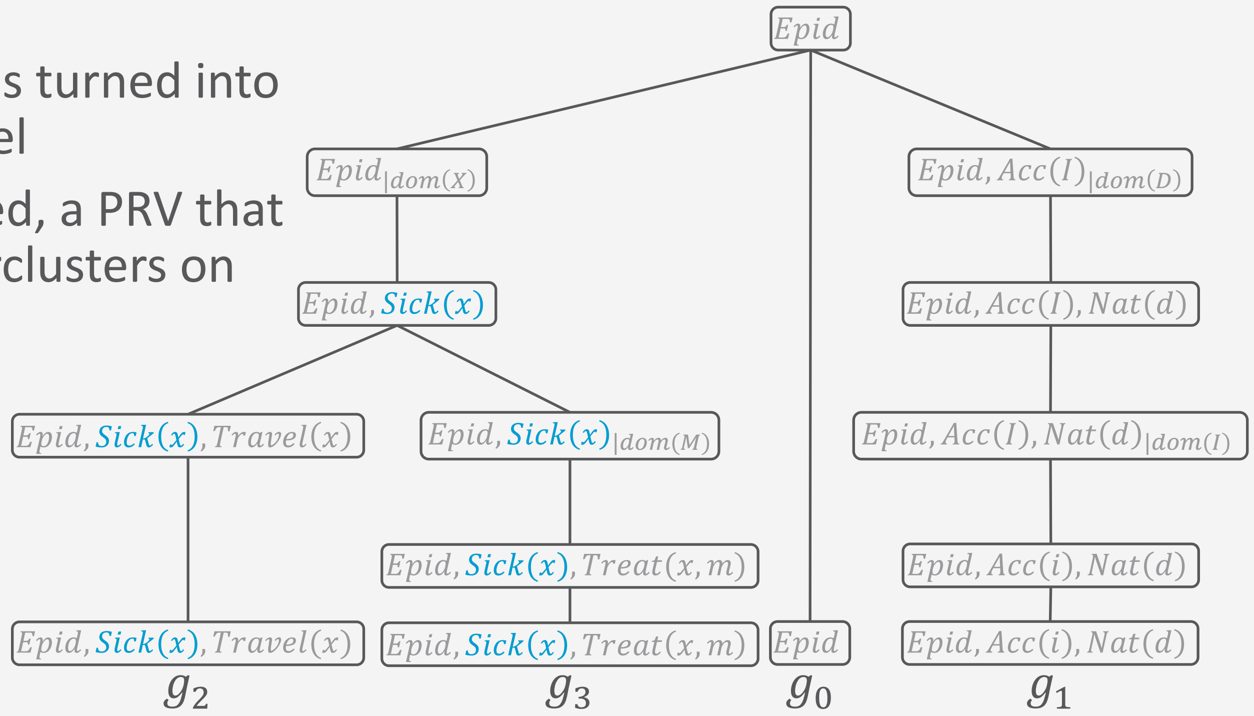




FO Dtree → FO Jtree

1. $\forall \boldsymbol{C} \in V : \boldsymbol{C} \subseteq rv(G)$
2. $\forall g \in G : \exists \boldsymbol{C} \in V : rv(g) \subseteq \boldsymbol{C}$
3. If $\exists A \in rv(G) : A \in \boldsymbol{C}_i \land A \in \boldsymbol{C}_j$ with $\boldsymbol{C}_i, \boldsymbol{C}_j \in V$, then $\forall \boldsymbol{C}_k \in V$ on the path between $\boldsymbol{C}_i, \boldsymbol{C}_j : A \in \boldsymbol{C}_k$

- Transformation result fulfils the three jtree properties
  - Properties hold by construction of the FO dtree
    1. Parclusters can only contain model PRVs
    2. Each parfactor occurs at a dtree leaf, which is turned into a parcluster with the parfactor as local model
    3. 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., $Sick(X)$

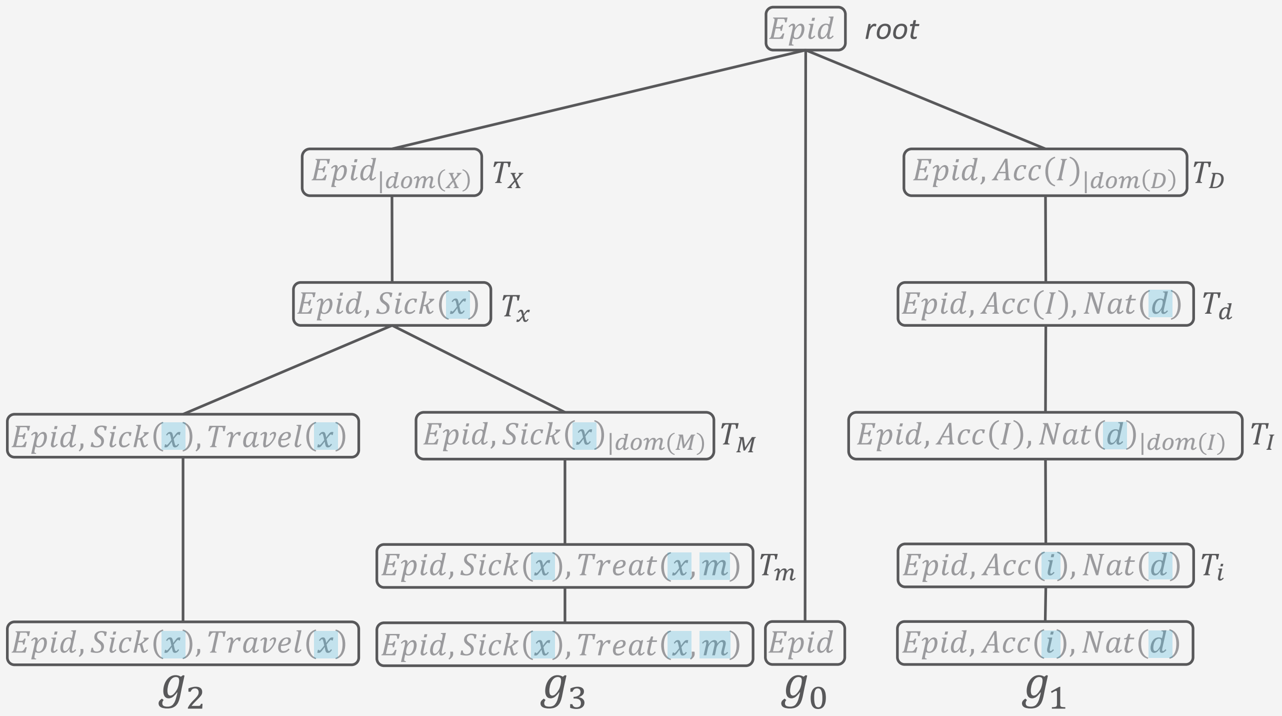


* Proof for jtrees: Adnan Darwiche: Recursive Conditioning. In: *Artificial Intelligence*, 2001.
Proof for FO jtrees: Tanya B: Rescued from a Sea of Queries: Exact Inference in Probabilistic Relational Models. PhD thesis, 2020.

FO Dtree → FO Jtree

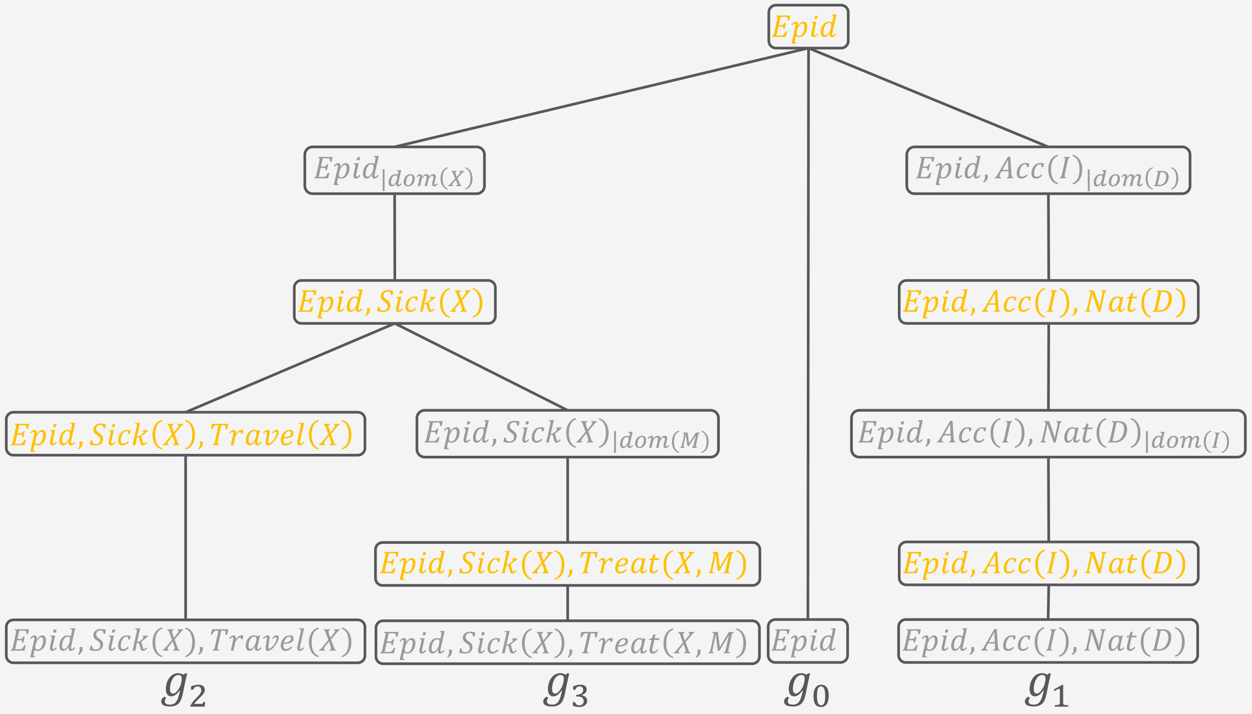
- But: Parclusters may contain a logical variable X or its representative x
- For each source DPG node T_X
  - Apply the inverse substitution θ^{-1} to the one applied during FO dtree construction to all parclusters that come from descendants of T_X:

$$\theta^{-1} = \{X \rightarrow x\}^{-1} = \{x \rightarrow X\}$$



FO Dtree → 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



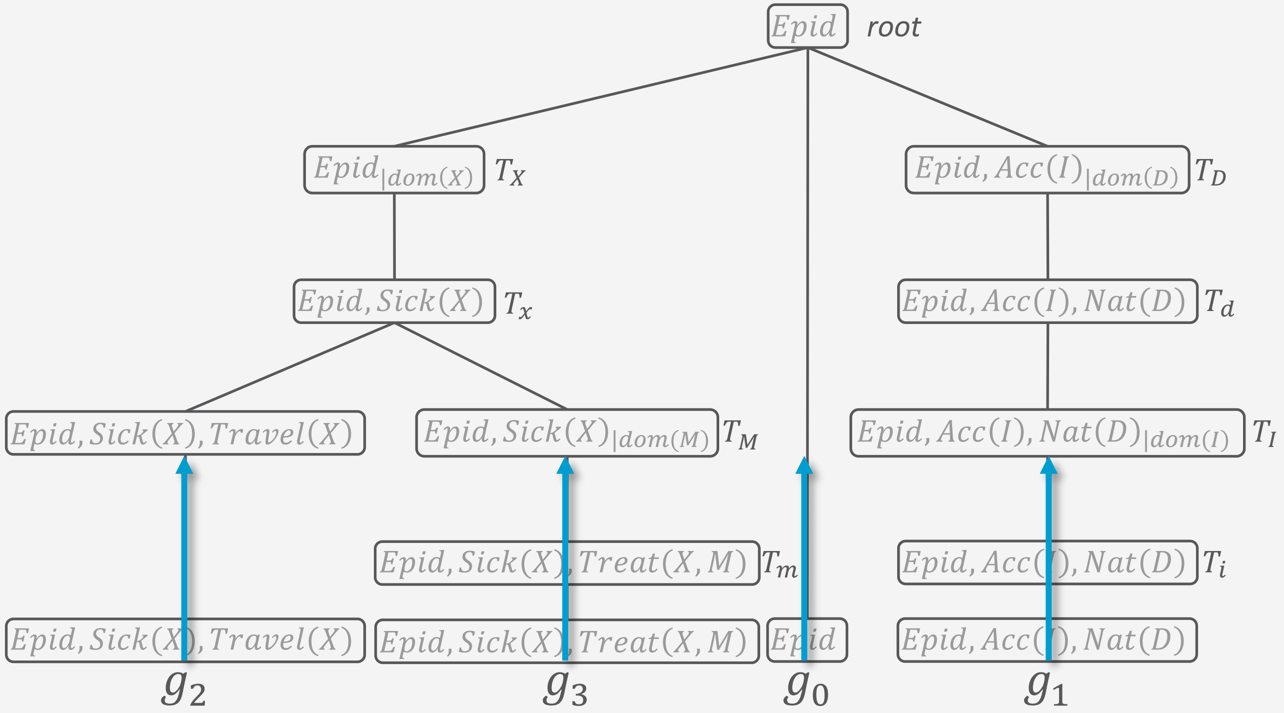
Minimisation

- Merge parclusters $\boldsymbol{C}_i, \boldsymbol{C}_j$ with local models G_i, G_j iff $gr(\boldsymbol{C}_i) \subseteq gr(\boldsymbol{C}_j) \vee gr(\boldsymbol{C}_j) \subseteq gr(\boldsymbol{C}_i)$
  - Assuming ⊤ constraints and same names for logical variables that reference the same domain (from normal form of FO dtree), then the following suffices:
    $$\boldsymbol{C}_i \subseteq \boldsymbol{C}_j \vee \boldsymbol{C}_j \subseteq \boldsymbol{C}_i$$
    - Checking on a PRV and logical variable level instead of a grounded level
- Merging parclusters $\boldsymbol{C}_i, \boldsymbol{C}_j$ into parcluster $\boldsymbol{C}_k$
  - $\boldsymbol{C}_k = \boldsymbol{C}_i \cup \boldsymbol{C}_j$
  - $G_k = G_i \cup G_j$
  - Changes in FO jtree (V, E)
    - $V = V \setminus \{\boldsymbol{C}_i, \boldsymbol{C}_j\} \cup \boldsymbol{C}_k$
    - $E = E \setminus \{\{i, l\} \mid l \in nbs(i)\} \setminus \{\{j, l\} \mid l \in nbs(j)\}$
      $\cup \{\{k, l\} \mid l \in nbs(i) \vee l \in nbs(j), l \neq i, l \neq j\}$

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

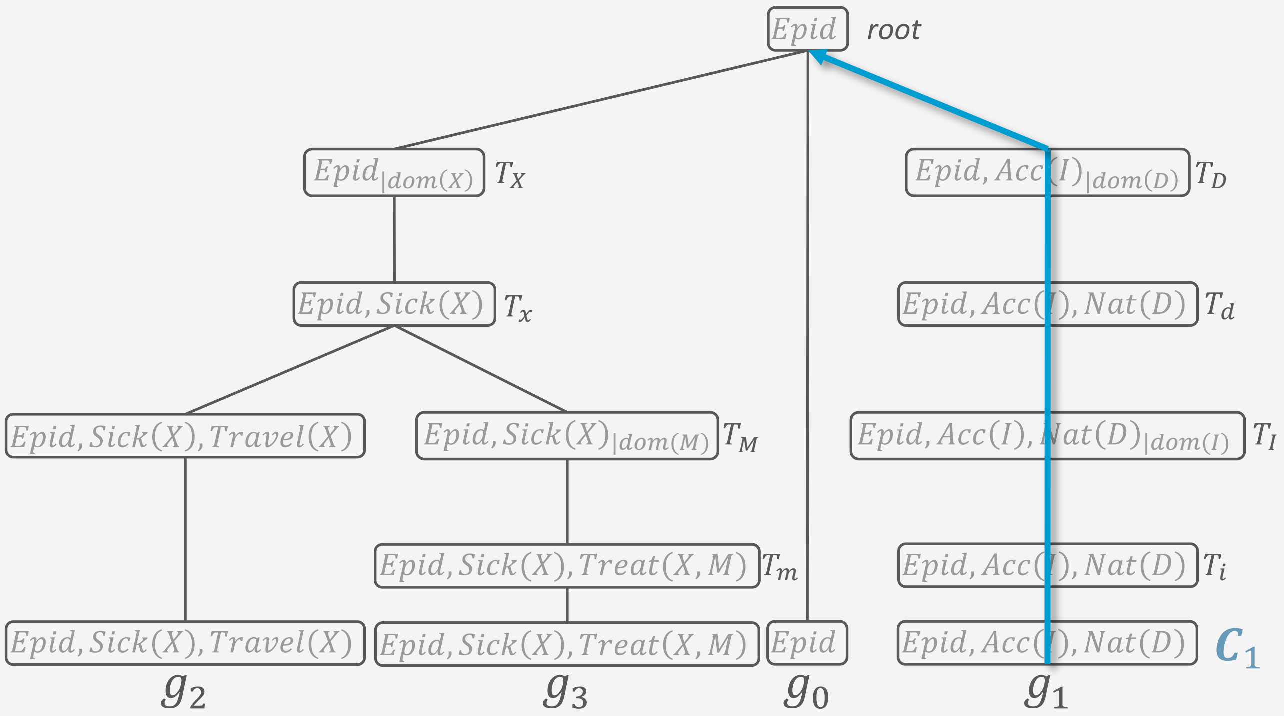
Minimisation

- 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 $\{g_3\}$



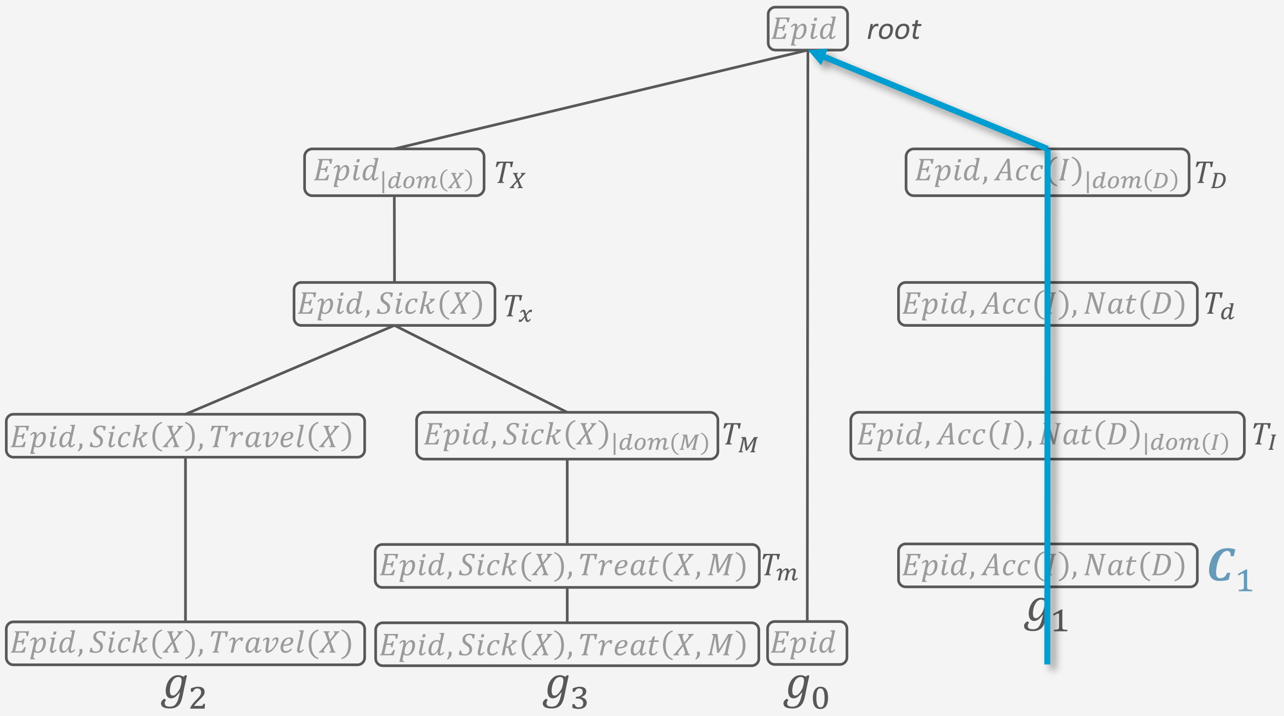
Minimisation: Example Continued

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



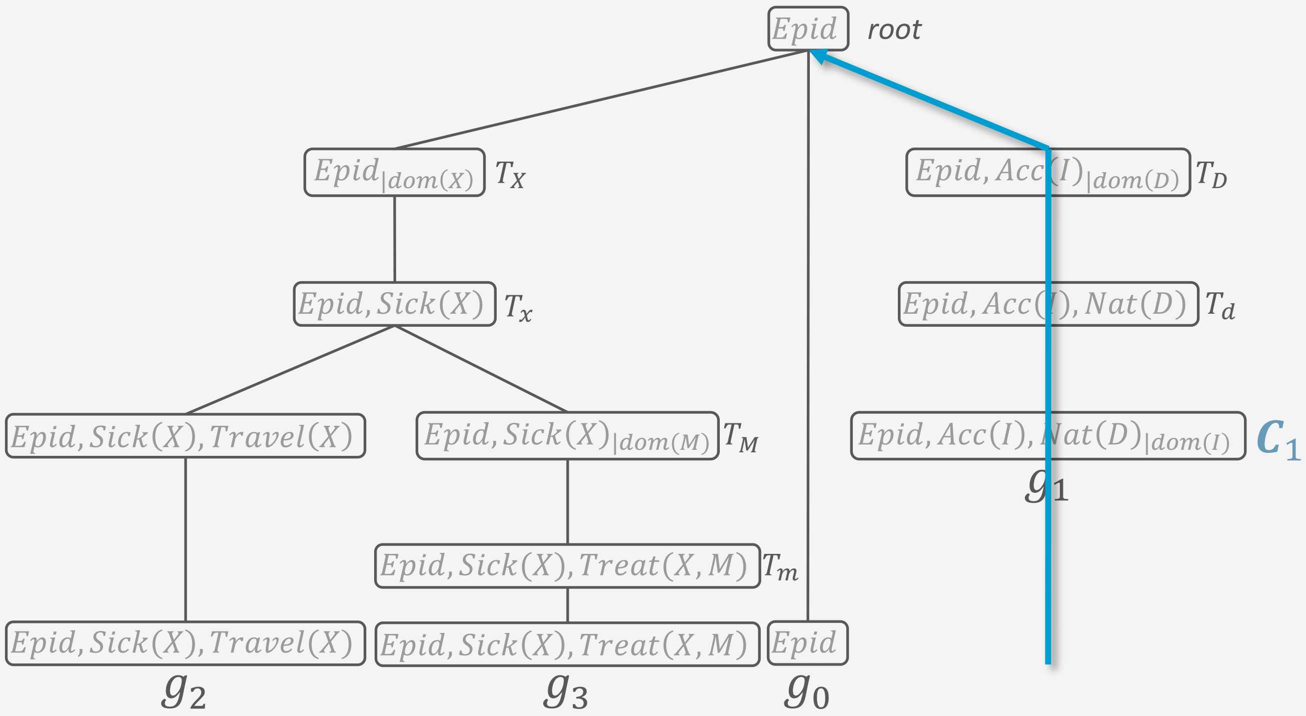
Minimisation: Example Continued

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



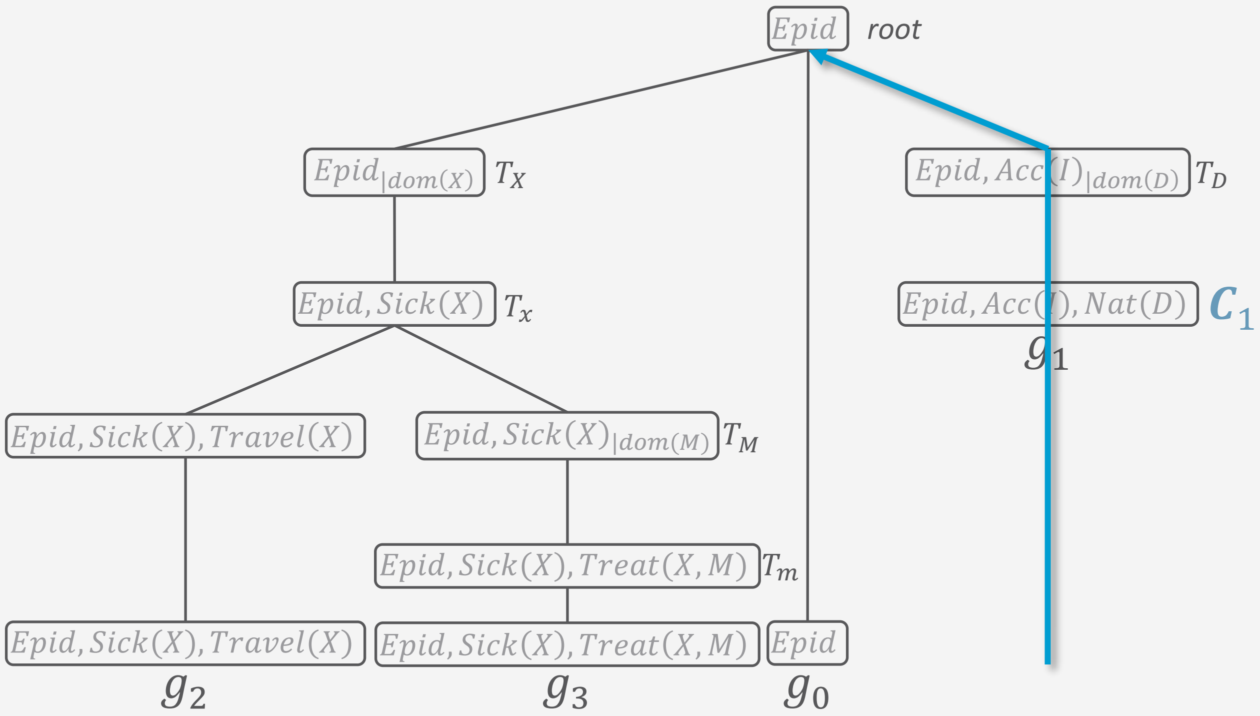
Minimisation: Example Continued

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



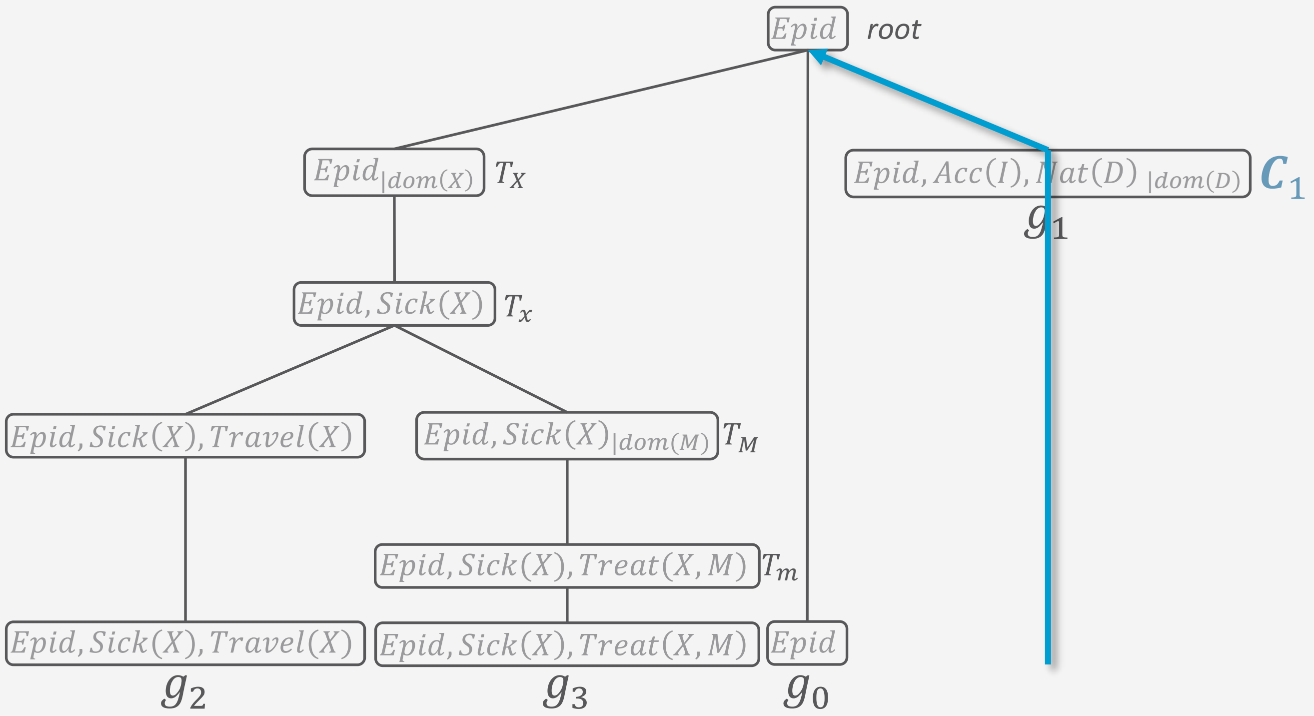
Minimisation: Example Continued

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



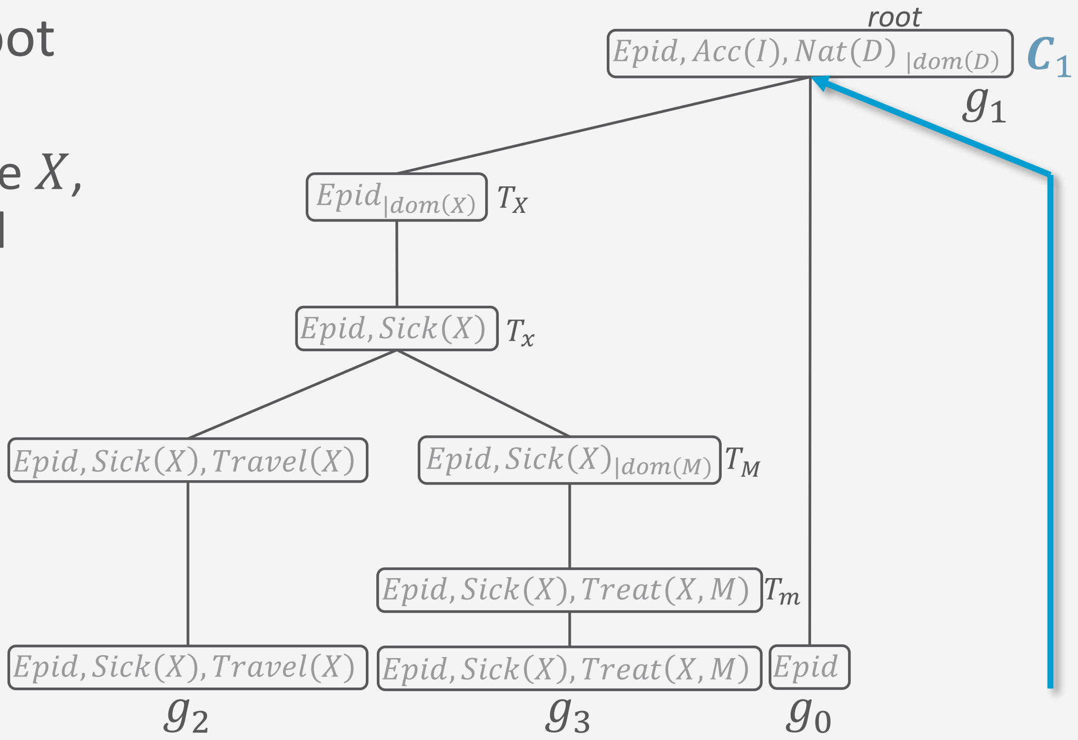
Minimisation: Example Continued

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



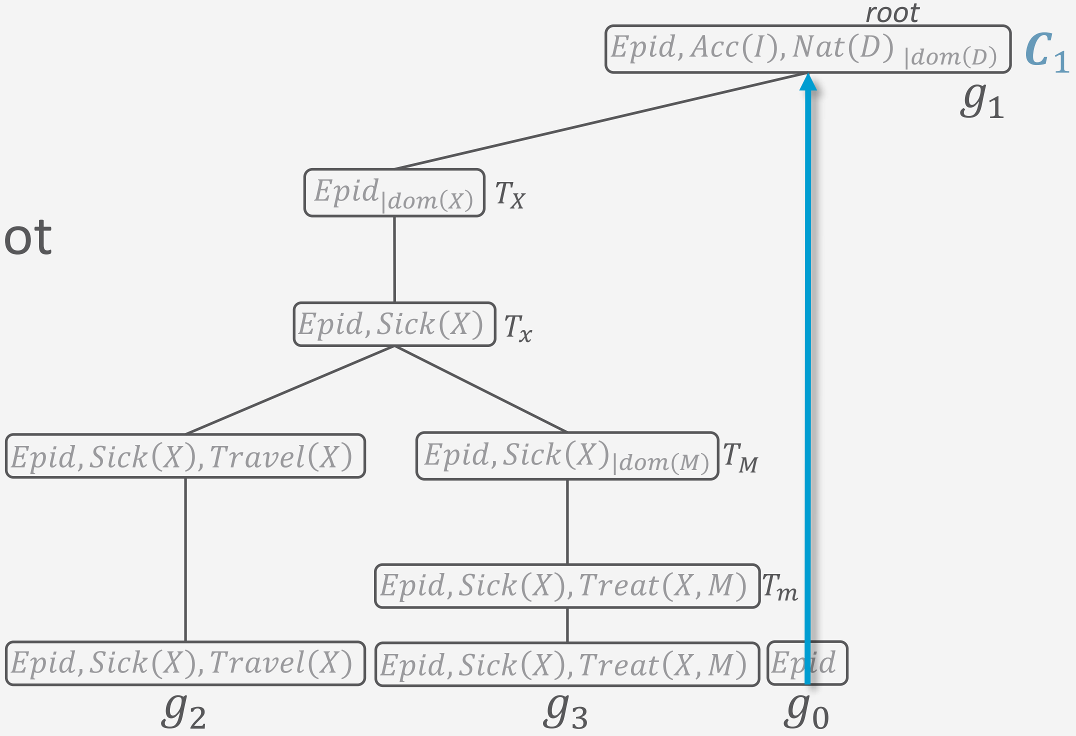
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_1\}$
  - Let us call it $\boldsymbol{C}_1$
  - 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 $\boldsymbol{C}_1$ (D, I)
    - Merging *stops*



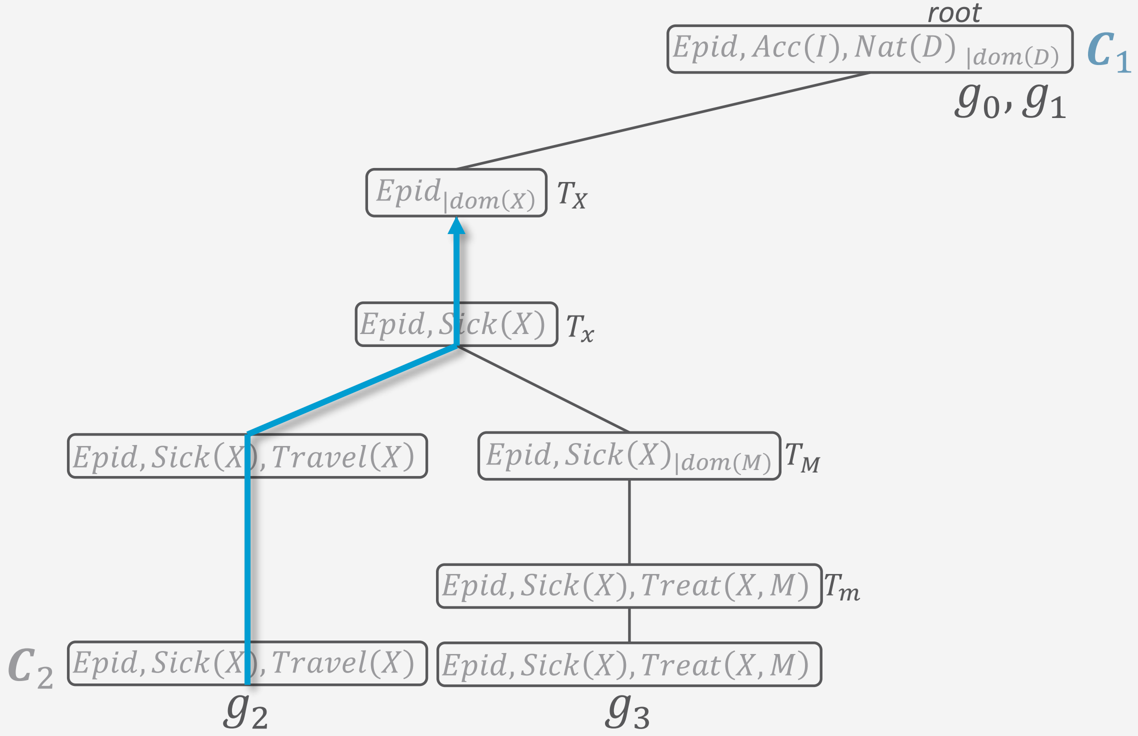
Minimisation: Example Continued

- Consider leaf parcluster with local model $\{g_0\}$
  - Let us call it $\boldsymbol{C}_0$
  - Merge inbound
    - $\boldsymbol{C}_0$ subset of $\boldsymbol{C}_1$
      → merge (call result $\boldsymbol{C}_1$ 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 $\boldsymbol{C}_1$ (D, I)
    - Merging *stops*



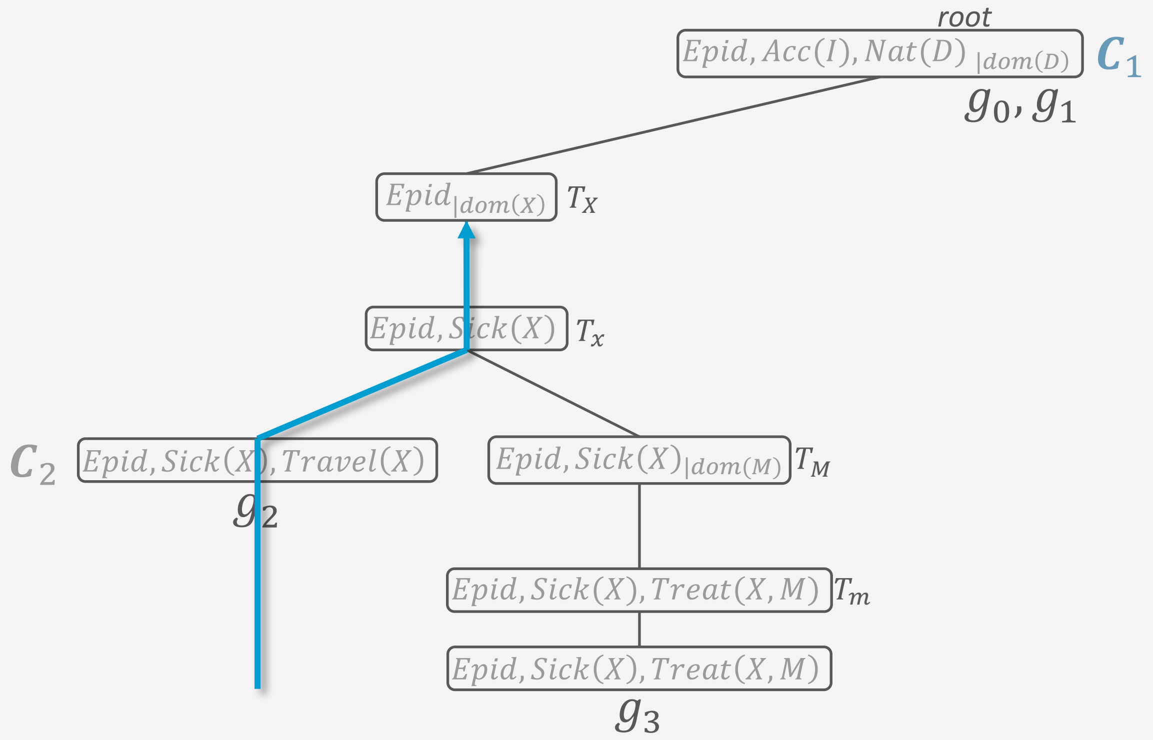
Minimisation: Example Continued

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



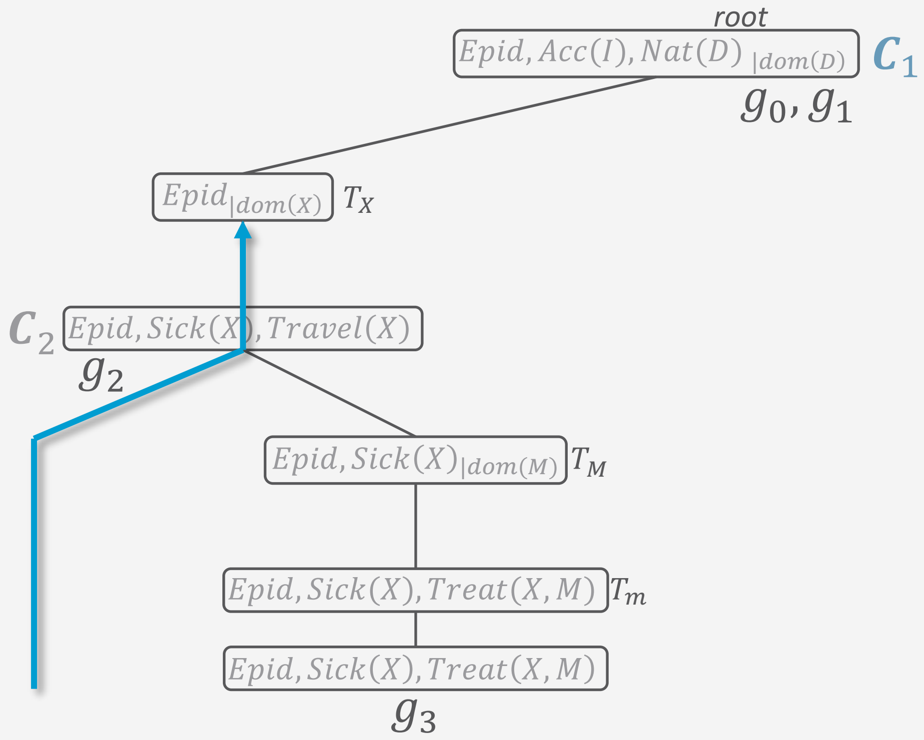
Minimisation: Example Continued

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



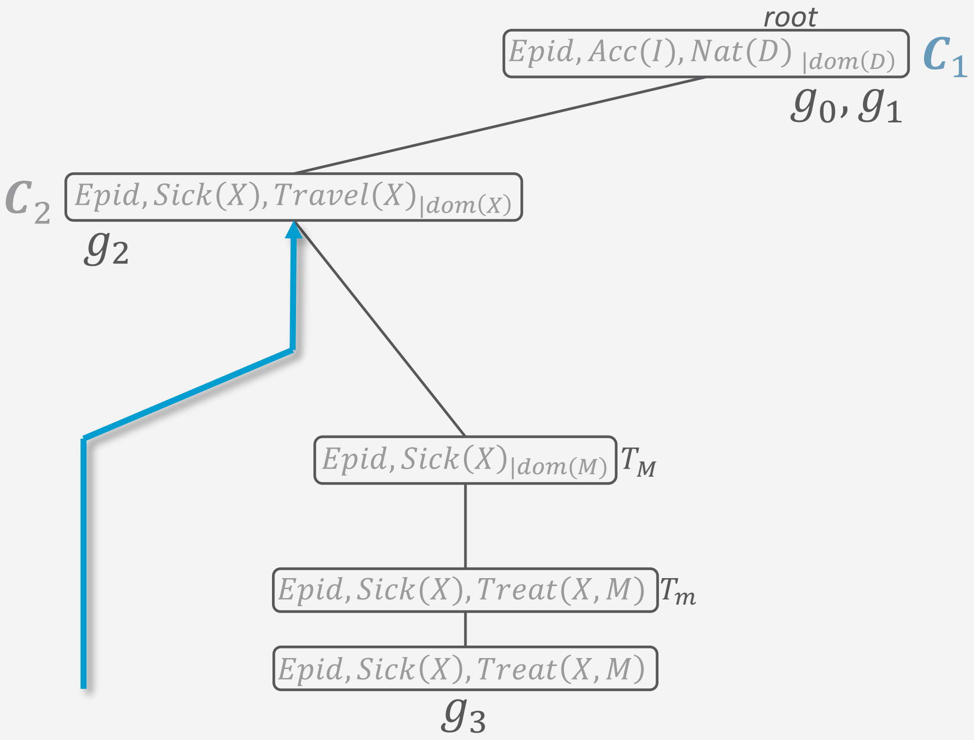
Minimisation: Example Continued

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



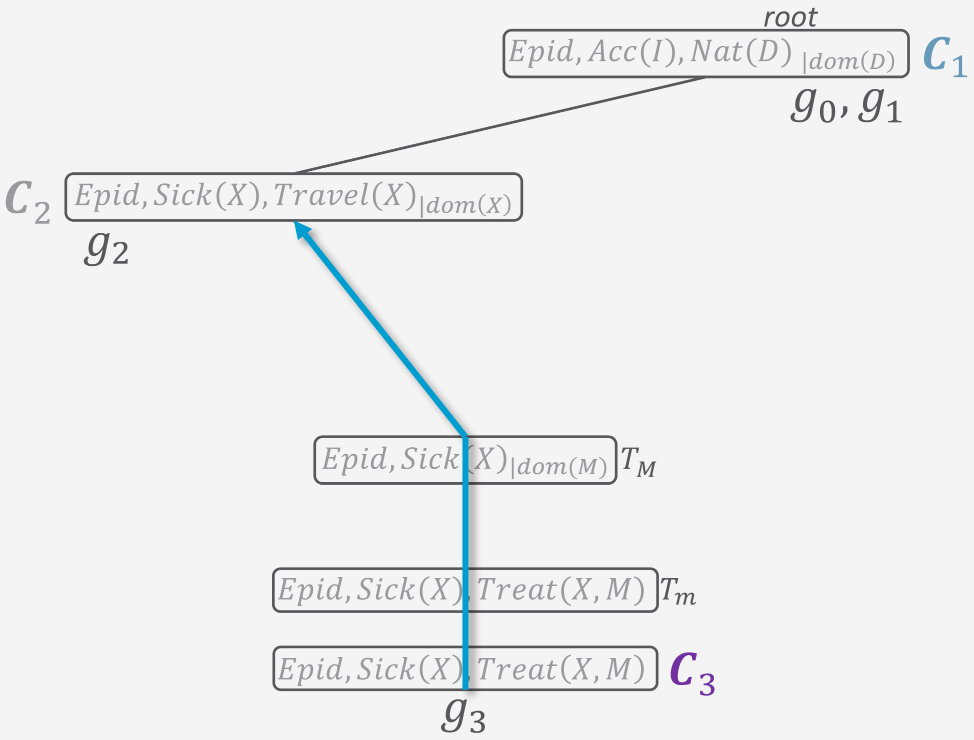
Minimisation: Example Continued

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



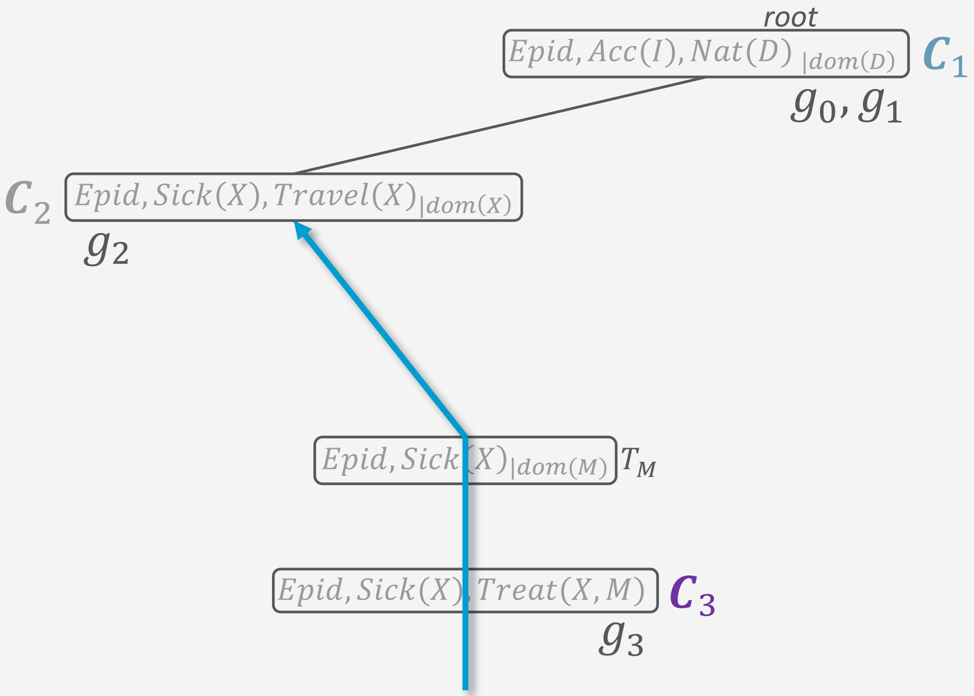
Minimisation: Example Continued

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



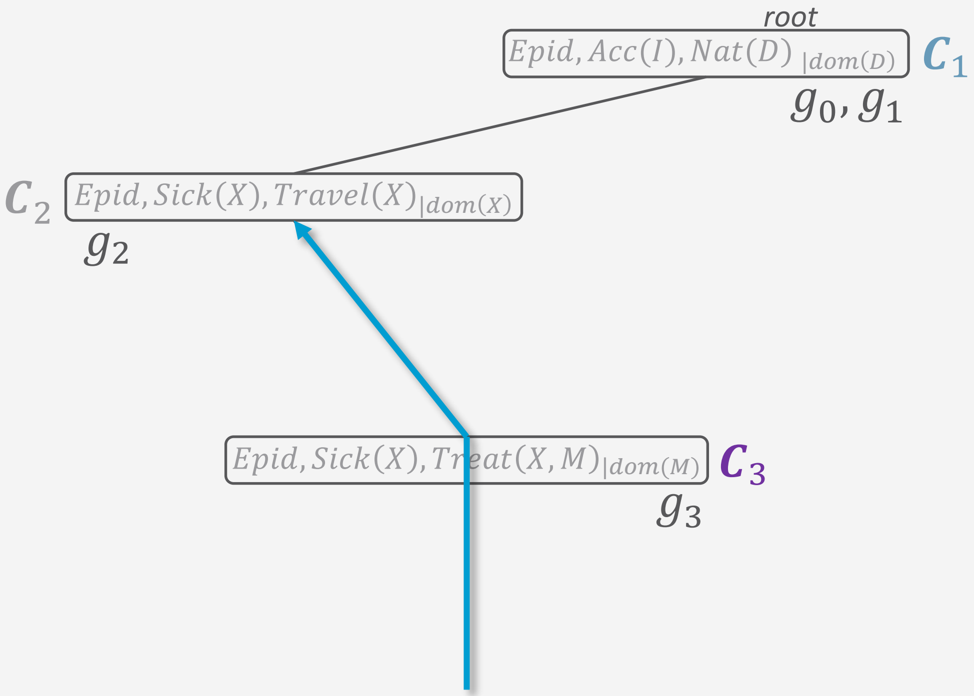
Minimisation: Example Continued

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



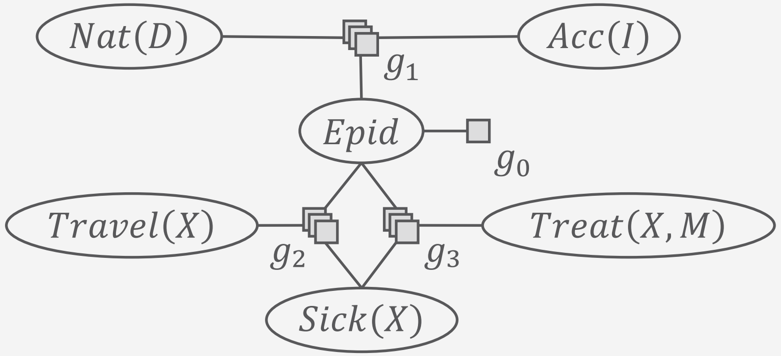
Minimisation: Example Continued

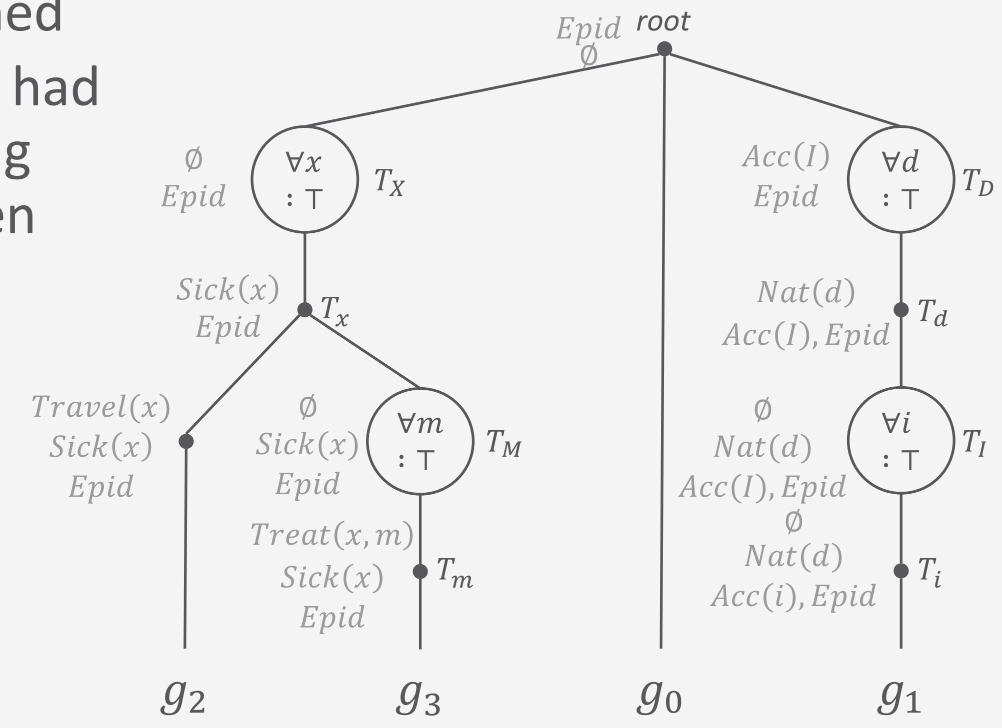
- Consider leaf parcluster with local model $\{g_3\}$
  - Let us call it $\boldsymbol{C}_3$
  - Merge inbound
    - $\boldsymbol{C}_3$ and T_m parcluster identical
      $\rightarrow$ merge (call result $\boldsymbol{C}_3$ again)
    - T_m parcluster is a subset of $\boldsymbol{C}_3$
      $\rightarrow$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

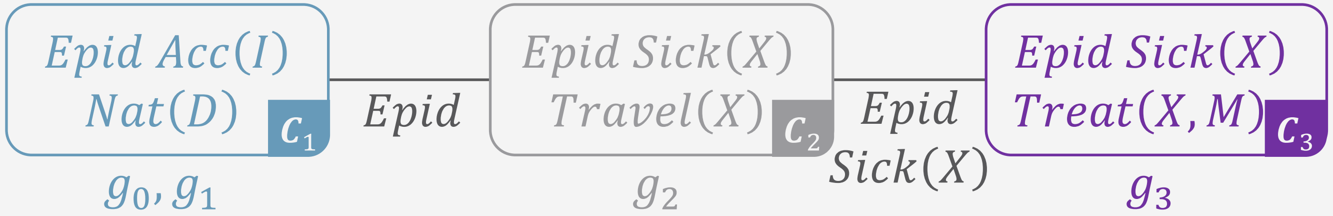


Minimisation: Example Continued

- 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, $\boldsymbol{C}_2$ and $\boldsymbol{C}_3$ 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

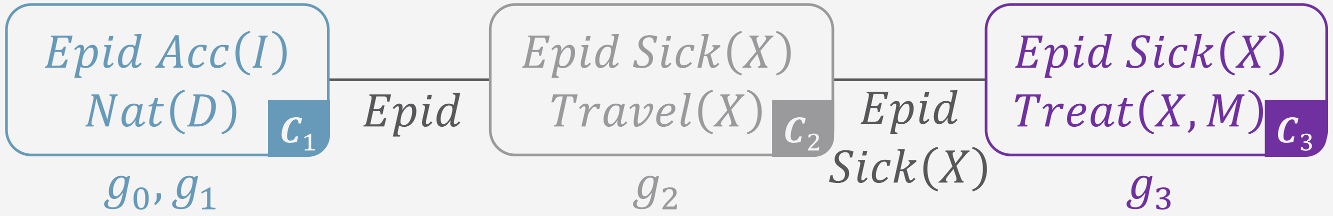
- Given a model G, the following steps are necessary
  1. Bring G into the required normal form for FO dtree construction
  2. Construct an FO dtree T for G
  3. Translate T into an FO jtree J
  4. Apply inverse substitutions to parclusters of descendants of DPG nodes in J
  5. Minimise J

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

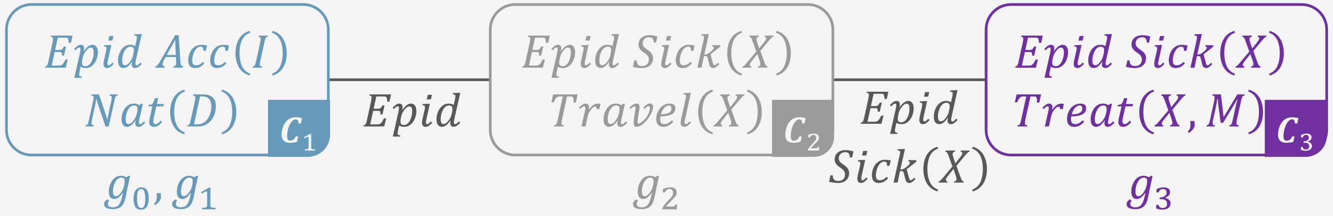


Message Passing in FO Jtrees

- Message m_{ij} from sender $\boldsymbol{C}_i$ to receiver $\boldsymbol{C}_j$
  - Set of parfactors $\{g_l\}_{l=1}^{n}$ with $rv(g_l) \subseteq \boldsymbol{S}_{ij}$
  - 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 $\boldsymbol{C}_j$ (if it already exists)
    - Call slightly modified LVE with G_{ij} as input model, $\boldsymbol{S}_{ij}$ as query, and no evidence: $\mathrm{LVE{-}MSG}(G_{ij}, \boldsymbol{S}_{ij})$
      - Specification of $\mathrm{LVE{-}MSG}$: next slide



LVE for Message Passing

$\mathbf{LVE{-}MSG}(G, \boldsymbol{S})$

$G \leftarrow$ Shatter G on itself

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

if a PRV A fulfils the preconditions of $\text{sum}{-}\text{out}$ **then**

$G \leftarrow$ Apply $\text{sum}{-}\text{out}$ 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)

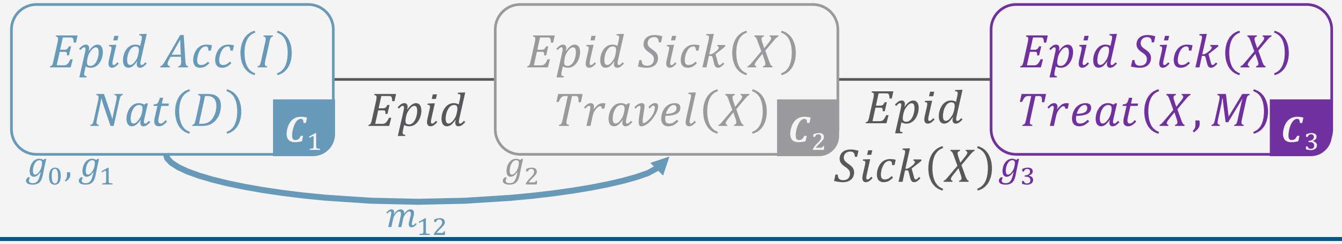
- Model might need to be shattered on itself because of splits introduced by messages

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

- Interim result returned

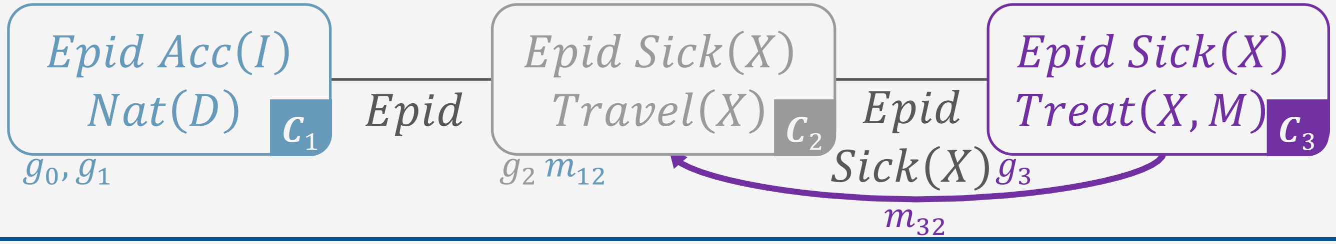
Message Passing in FO Jtrees: Example

- Message m_{12} from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$
  - Collect $G_{12} = \{g_1\} \cup \emptyset$
    - No further neighbours except $\boldsymbol{C}_2$
  - Call $\text{LVE-MSG}(\{g_1\}, \{Epid\}, \emptyset)$
    - 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 $\boldsymbol{C}_2$



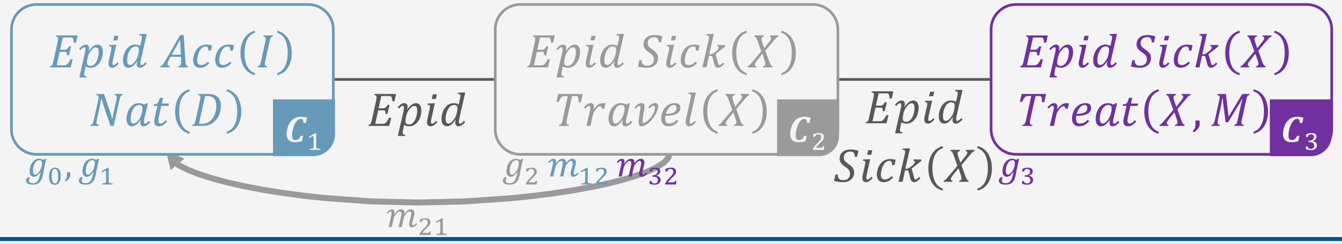
Message Passing in FO Jtrees: Example

- Message m_{32} from $\boldsymbol{C}_3$ to $\boldsymbol{C}_2$
  - Collect $G_{32} = \{g_3\} \cup \emptyset$
    - No further neighbours except $\boldsymbol{C}_2$
  - Call $\mathrm{LVE-MSG}(\{g_3\}, \{Epid, Sick(X)\}, \emptyset)$
    - $\mathrm{LVE-MSG}$ eliminates $Treat(X, M)$ from $\{g_3\}$
      - Summing out $Treat(X, M)$
      - Returning $\{g_3'\}$
  - Send $\{g_3'\}$ as m_{32} to $\boldsymbol{C}_2$



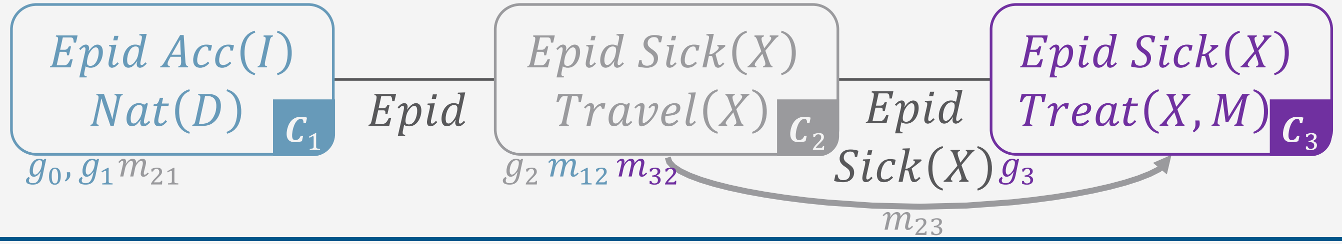
Message Passing in FO Jtrees: Example

- Message m_{21} from $\boldsymbol{C}_2$ to $\boldsymbol{C}_1$
  - Collect $G_{21} = \{g_2\} \cup m_{32}$
    - Further neighbour: $\boldsymbol{C}_3$, sent message $m_{32} = \{g_3'\}$
  - Call $\text{LVE-MSG}(\{g_2, g_3'\}, \{Epid\}, \emptyset)$
    - LVE-MSG eliminates $Travel(X), Sick(X)$ from $\{g_2, g_3'\}$
      - Summing out $Travel(X)$ from g_2, yielding g_2'
      - Summing out $Sick(X)$ from product of g_2' and g_3', yielding g_{23}'
      - Returning $\{g_{23}'\}$
  - Send $\{g_{23}'\}$ as m_{21} to $\boldsymbol{C}_1$



Message Passing in FO Jtrees: Example

- Message m_{23} from $\boldsymbol{C}_2$ to $\boldsymbol{C}_3$
  - Collect $G_{23} = \{g_2\} \cup m_{12}$
    - Further neighbour: $\boldsymbol{C}_1$, sent message $m_{12} = \{g_1'\}$
  - Call $\mathrm{LVE}^*(\{g_2, g_1'\}, \{Epid, Sick(X)\}, \emptyset)$
    - LVE^* eliminates $Travel(X)$ from $\{g_2, g_1'\}$
      - Summing out $Travel(X)$ from g_2, yielding g_2'
      - Returning $\{g_2', g_1'\}$
  - Send $\{g_2', g_1'\}$ as m_{23} to $\boldsymbol{C}_3$



Message Passing: Overview

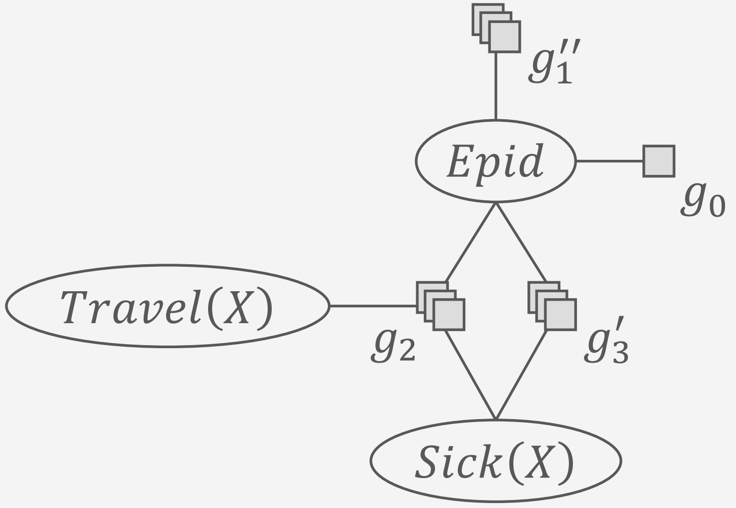
- Given an FO jtree J, send messages if one of the two conditions is true
  → If a node i has received all messages from neighbours but one, j, node i calculates and sends a message to j
  → If a node i has received all messages, then it calculates and sends messages to all neighbours j that have not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

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

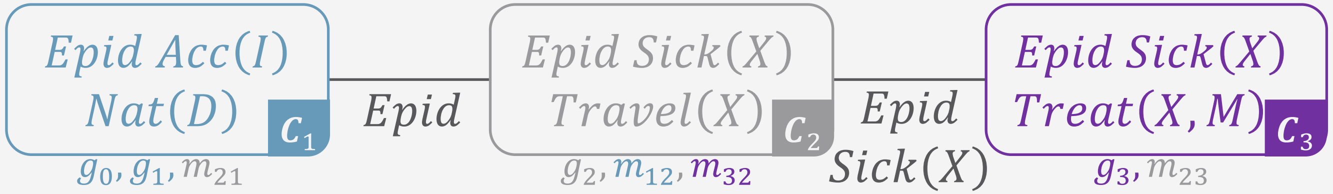
  - Call $\mathrm{LVE{-}MSG}\left(G_{ij}, \boldsymbol{S}_{ij}\right)$

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., $\boldsymbol{C}_2$
    - Collect $G_{Epid} = \{g_2\} \cup m_{12} \cup m_{32} = \{g_2, g_1', g_3'\}$ (depicted right)
    - Call $\text{LVE}(\{g_2, g_1', g_3'\}, Epid, \emptyset)$, yielding a parfactor g containing the probability distribution over $Epid$
- What if query terms occur outside of one parcluster?
  - E.g., $P\big(Travel(eve), Treat(eve, m_1)\big)$



Query Answering in FO Jtrees

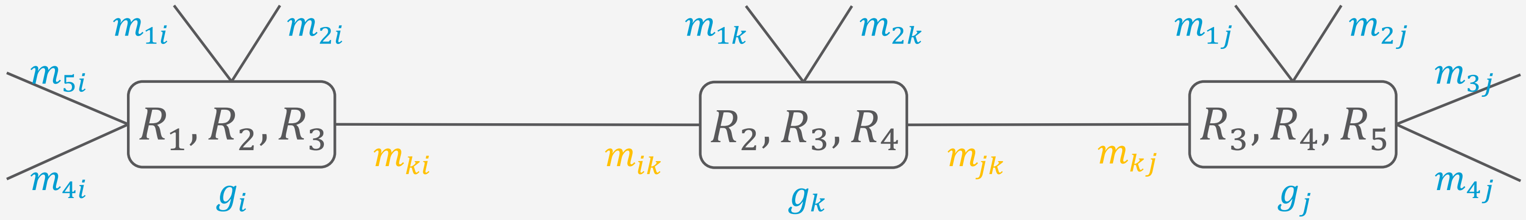
How can we find a subgraph?

- For query terms $\boldsymbol{Q}$, possibly contained in more than one parcluster
  - Find a subgraph J' of the FO jtree J such that $\boldsymbol{Q} \subseteq rv(J')$
  - 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 $\boldsymbol{C}_i, \boldsymbol{C}_k, \boldsymbol{C}_j$
    - Ignore inside messages $m_{ki}, m_{ik}, m_{jk}, m_{kj}$

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

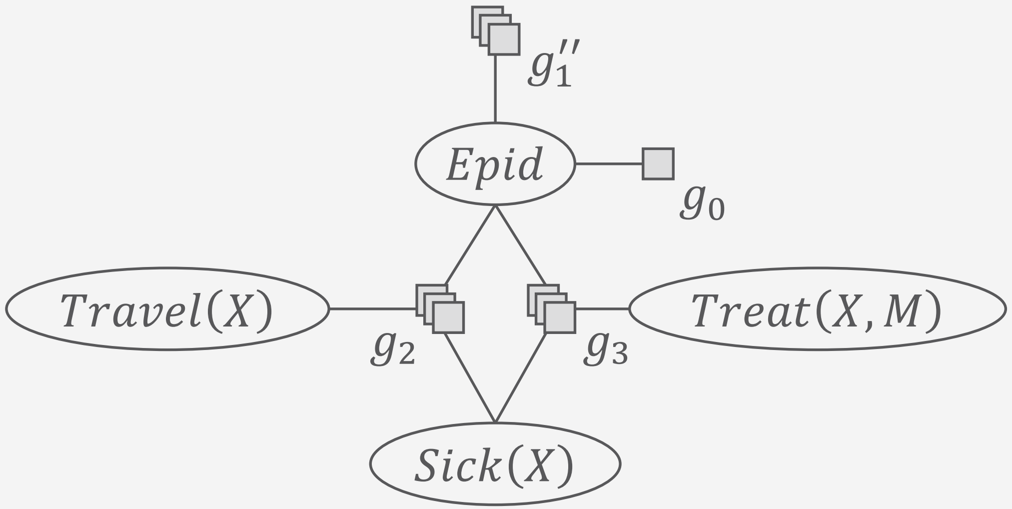
$$\underset{J'}{\operatorname{argmin}} |rv(J')|$$
$$\text{s.t. } \boldsymbol{Q} \subseteq rv(J')$$

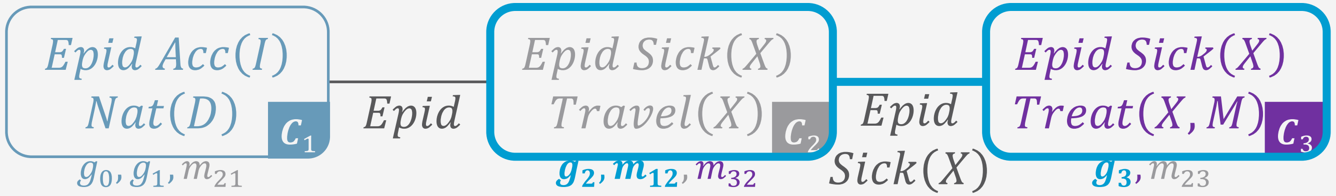
  - Trade-off between finding a subgraph fast and finding a minimal one
  - It is not about the number of parclusters!



Query Answering in FO Jtrees: Example

- E.g., $P\big(Travel(eve), Treat(eve, m_1)\big)$
  - Subgraph: $\boldsymbol{C}_2, \boldsymbol{C}_3$
  - Submodel for query answering: $G_{\boldsymbol{Q}} = (g_2, g_3, m_{12})$
    - Depicted right
  - Call LVE with $G_{\boldsymbol{Q}}$ and $\boldsymbol{Q} = \{Travel(eve), Treat(eve, m_1)\}$
    - Split off query terms
    - Eliminate all non-query terms
    - Normalise the result





Query Answering in FO Jtrees

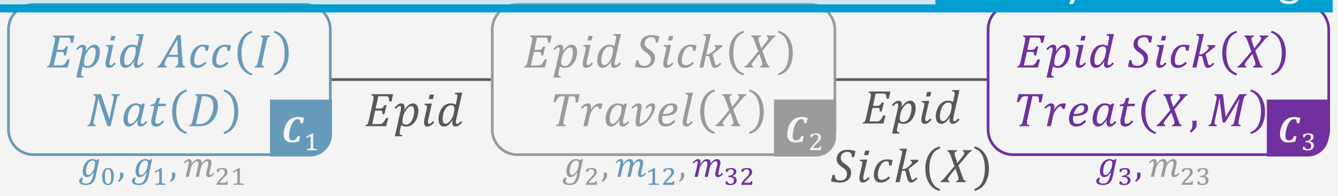
- After message passing, parclusters independent from each other given messages
  - Prepared for query answering
- For each query with query terms $\boldsymbol{Q}$
  - Find subtree $J' = (V', E')$ s.t. $\boldsymbol{Q} \subseteq rv(J')$
  - Collect information from local model and messages, i.e,

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

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

Query Answering

- What about evidence?

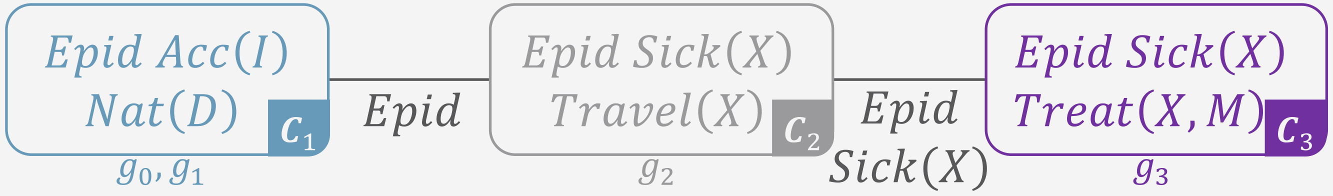


Evidence in FO Jtrees

- 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

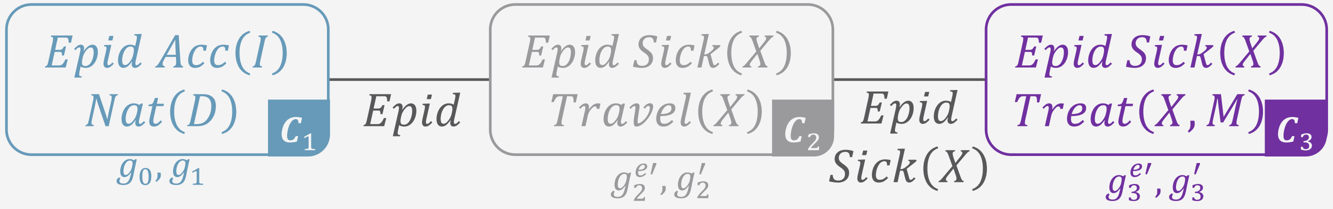
Evidence Entering

- 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 $\boldsymbol{C}_i$ where $A_e \in \boldsymbol{C}_i$
    - Shatter G_i on C_e
    - Absorb $\phi_e(A_e)_{|C_e}$ in G_i



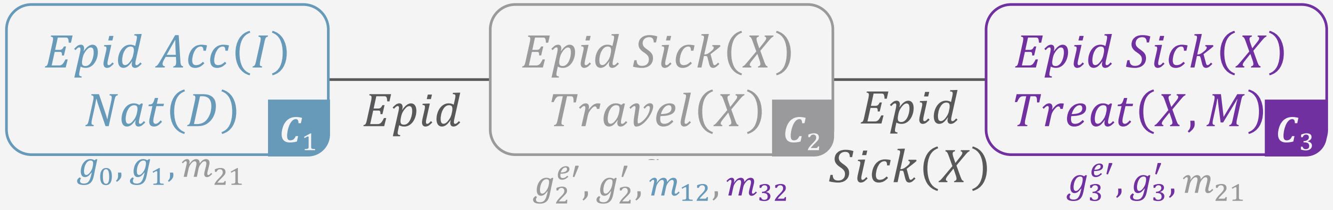
Evidence in FO Jtrees: Example

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



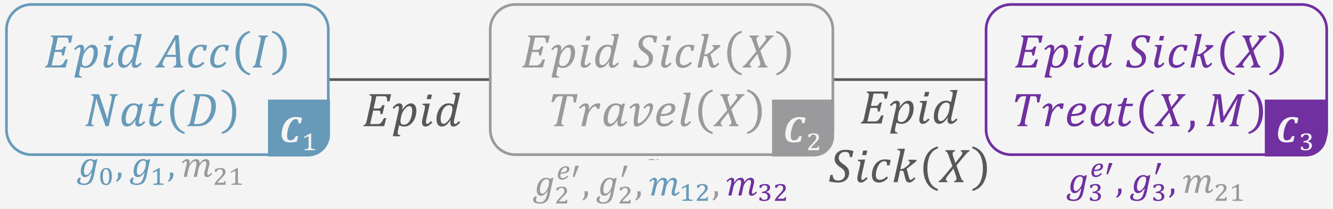
Evidence in FO Jtrees

- 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 $\{g_3^{e\prime}, g_3^{\prime}\}$
    - Call LVE−MSG($\{g_3^{e\prime}, g_3^{\prime}\}, \{Epid, Sick(X)\}, \emptyset$), yielding $\{g_3^{e\prime\prime}, g_3^{\prime\prime}\}$
  - Message m_{23} calculated based on $\{g_2^{e\prime}, g_2^{\prime}\} \cup m_{12}$
    - Call LVE−MSG($\{g_2^{e\prime}, g_2^{\prime}, g_1^{\prime}\}, \{Epid, Sick(X)\}, \emptyset$), yielding $\{g_2^{e\prime\prime}, g_2^{\prime\prime}, g_1^{\prime}\}$
  - Message m_{21} calculated based on $\{g_2^{e\prime}, g_2^{\prime}\} \cup m_{32}$
    - Call LVE−MSG($\{g_2^{e\prime}, g_2^{\prime}, g_3^{e\prime\prime}, g_3^{\prime\prime}\}, \{Epid\}, \emptyset$), yielding $\{g_2^{e\prime\prime}, g_2^{\prime\prime}, g_3^{e\prime\prime}, g_3^{\prime\prime\prime}\}$



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 $\{\boldsymbol{Q}_i\}_{i=1}^n$ for a model G
    - The posed queries are $P(\boldsymbol{Q}_i \mid \{g_e\}_{e=1}^m), 1 \leq i \leq n$, 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

$\mathbf{LJT}(G, \{\boldsymbol{Q}_i\}_{i=1}^{n}, \{g_e\}_{e=1}^{m})$

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 $\{\boldsymbol{Q}_i\}_{i=1}^{n}$ in J

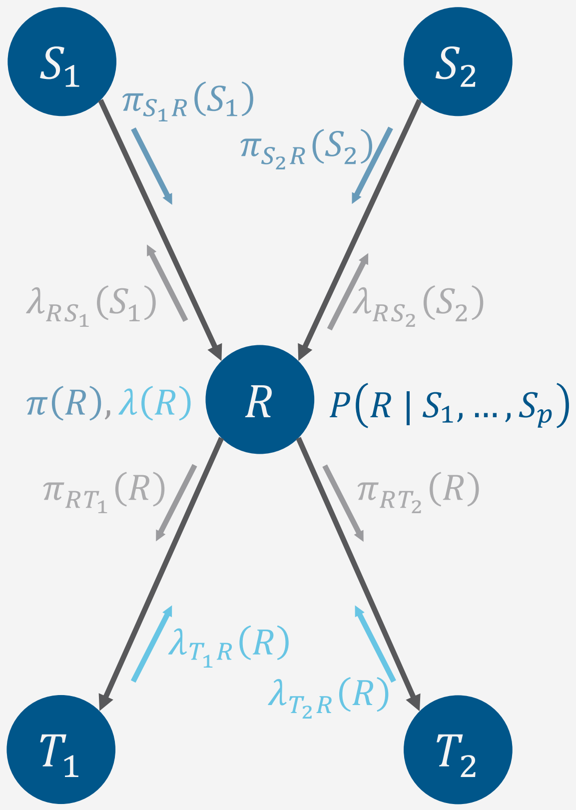
- Look for blue boxes on the previous slides to find the descriptions of each step

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



Judea Pearl: Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In: *AAAI-82 Proceedings of the 2nd National Conference on Artificial Intelligence*, 1982.

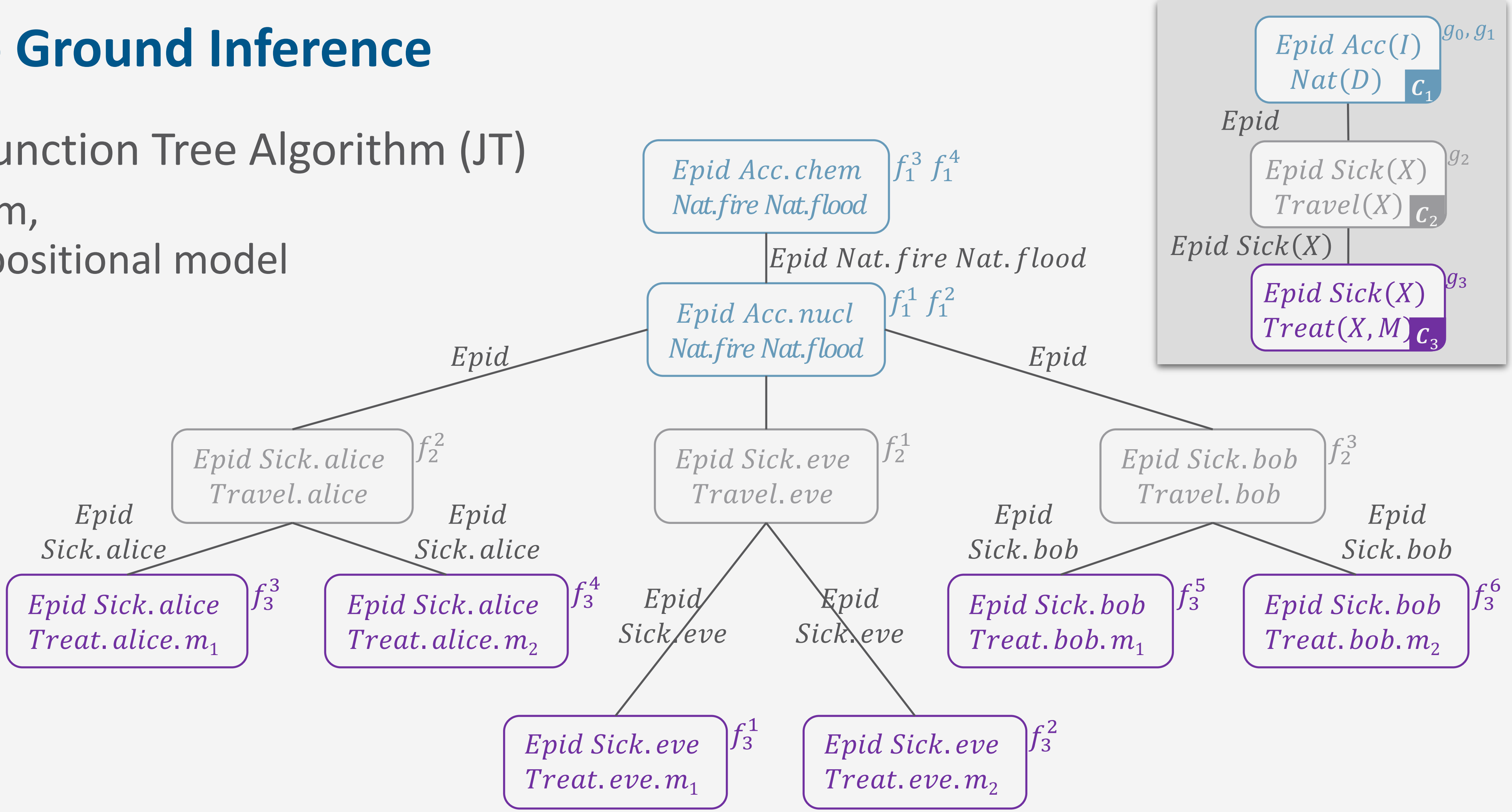
Foundations of Clustering

- 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

Steffen L. Lauritzen and David J. Spiegelhalter: Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. In: *Journal of the Royal Statistical Society. Series B: Methodological*, 1988.
Prakash P. Shenoy and Glenn R. Shafer: Axioms for Probability and Belief-Function Propagation. In: *Uncertainty in Artificial Intelligence 4*, 1990.

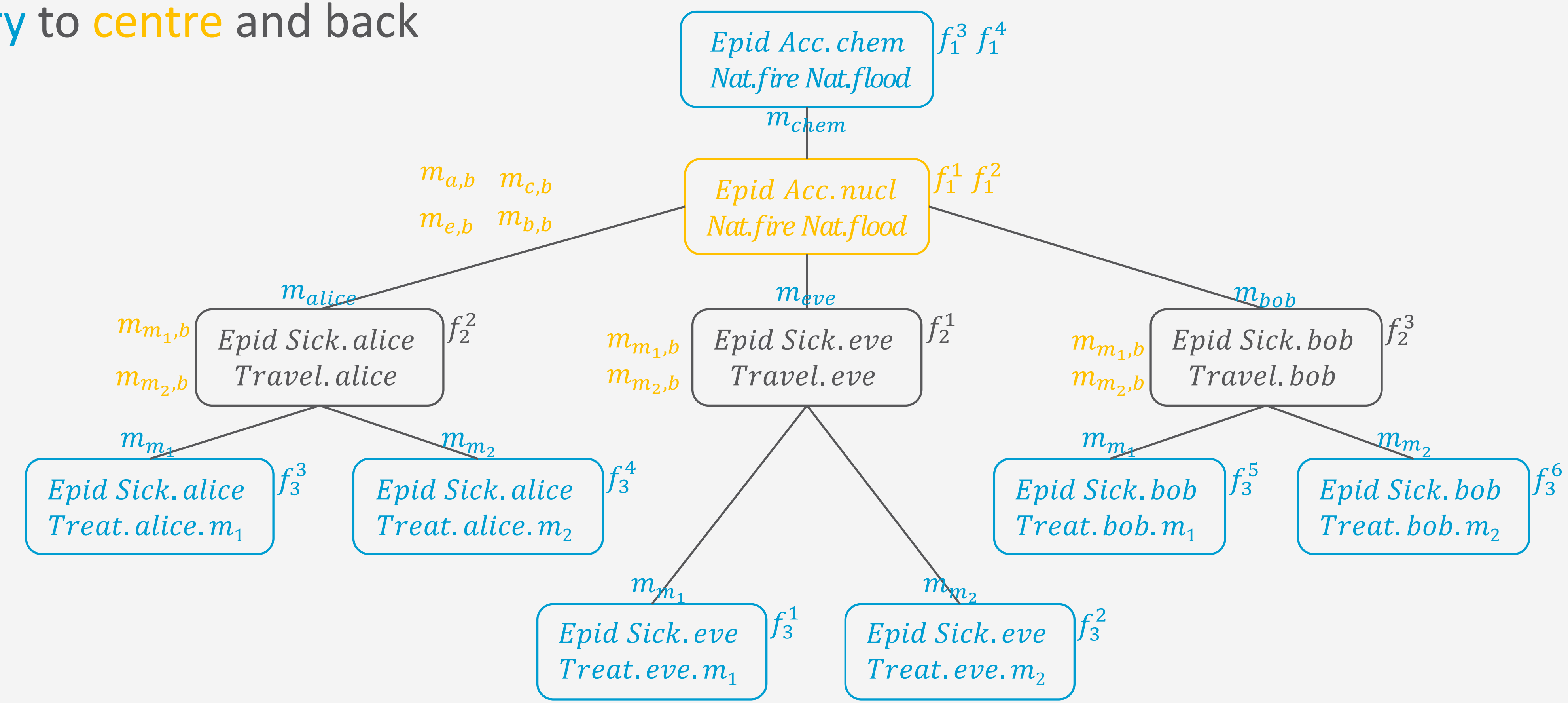
Comparison to Ground Inference

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



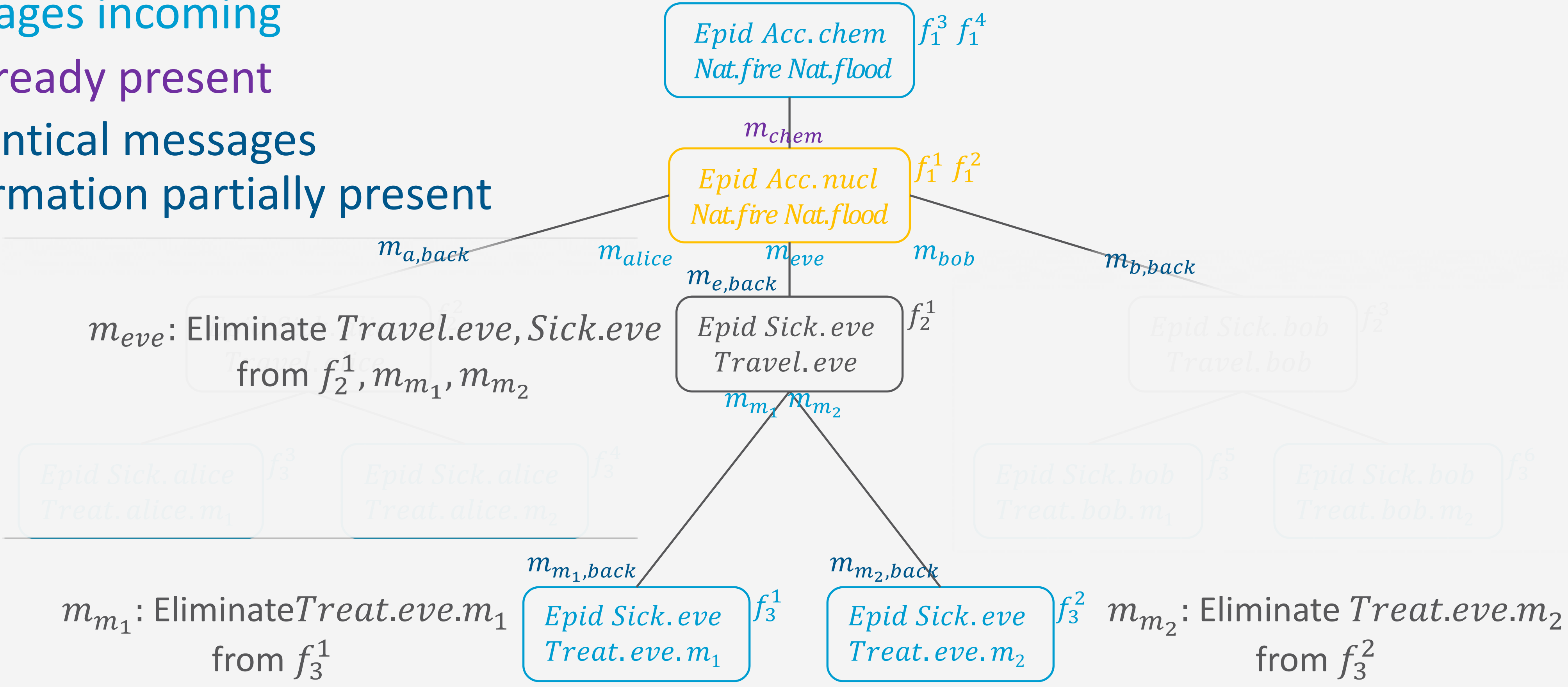
Junction Tree: Messages

- 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

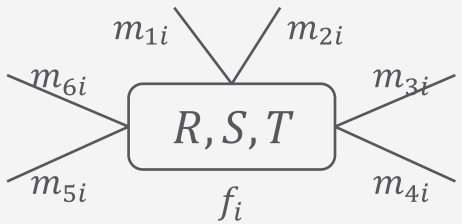


Message Calculation Strategies

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

- 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 parlcuster and message
      - Example right: for each outgoing message, only one incoming message changes
- Alternative: Hugin architecture [Jensen et al., 1989]
  - Hugin factor $h_i = \phi_i(\boldsymbol{C}_i)$ per parcluster $\boldsymbol{C}_i$ 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: $\mathrm{VE{-}JT}\big(f_i \,/\, m_{ji}, S_{ij}, \emptyset, .\big)$
    - Divide h_i by message m_{ji}, then sum out non-separators
      - One multiplication and one division instead of multiple multiplications

$m_{i1} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{2i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$
$m_{i2} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{1i}, m_{3i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$
$m_{i3} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{1i}, m_{2i}, m_{4i}, m_{5i}, m_{6i}\}, ., ., .)$
$m_{i4} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{5i}, m_{6i}\}, ., ., .)$
$m_{i5} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{6i}\}, ., ., .)$
$m_{i6} \leftarrow \mathrm{VE{-}JT}(\{f_i, m_{1i}, m_{2i}, m_{3i}, m_{4i}, m_{5i}\}, ., ., .)$



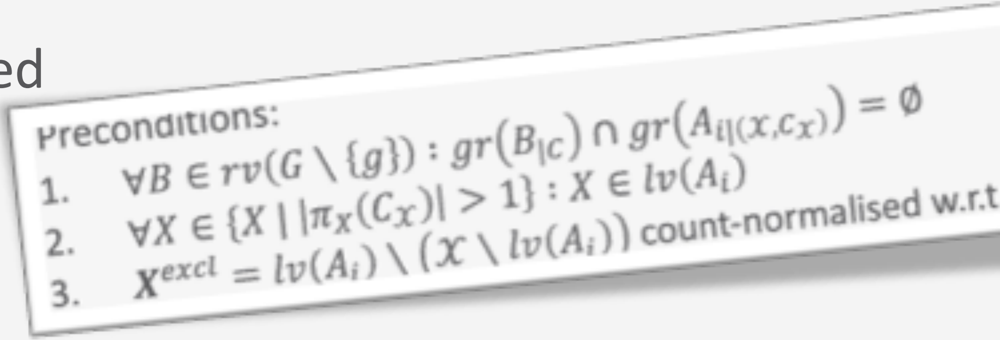
$h_i \leftarrow f_i$

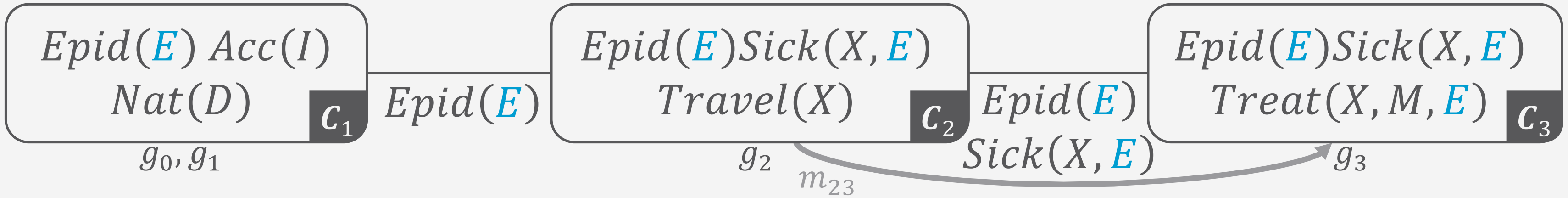
$h_i \leftarrow h_i \cdot m_{1i}$	$h_i \,/\, m_{1i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{2i}$	$h_i \,/\, m_{2i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{3i}$	$h_i \,/\, m_{3i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{4i}$	$h_i \,/\, m_{4i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{5i}$	$h_i \,/\, m_{5i} \rightarrow$ VE-JT
$h_i \leftarrow h_i \cdot m_{6i}$	$h_i \,/\, m_{6i} \rightarrow$ VE-JT

Glenn R. Shafer and Prakash P. Shenoy: Probability Propagation. In: Annals of Mathematics and Artificial Intelligence, 1990.
Finn V. Jensen and Steffen L. Lauritzen and Kristian G. Olesen: Bayesian Updating in Recursive Graphical Models by Local Computations. In: *Computational Statistics Quarterly*, 1990.
Moritz Hoffmann, Tanya B, and Ralf Möller: Lifted Division for Lifted Hugin Belief Propagation. In: *AISTATS-22 Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, 2022

In terms of Lifting: Is it that simple?

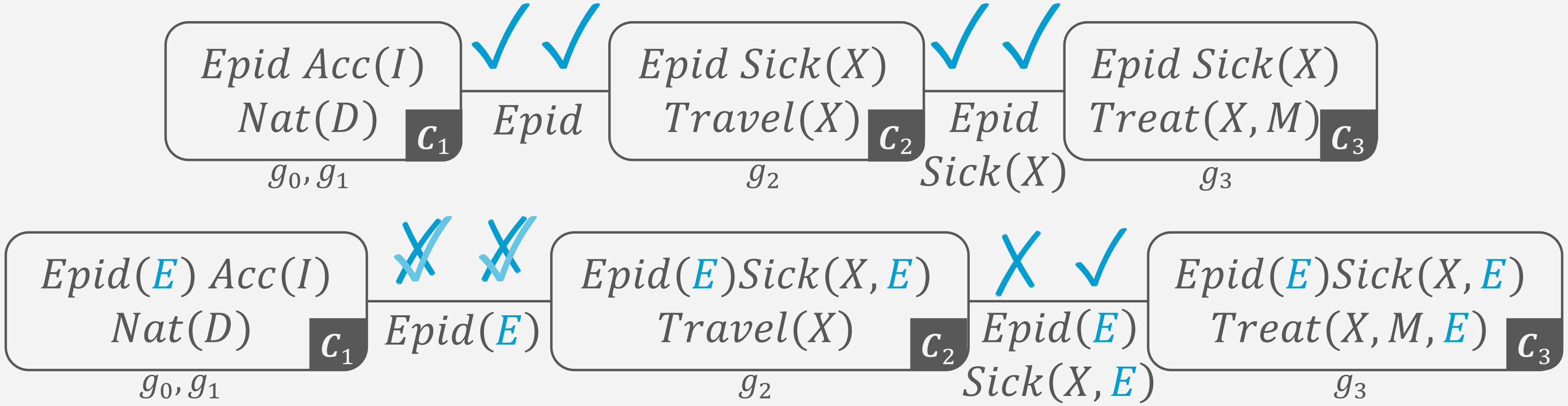
- Algorithm-induced groundings due to message passing
  - For message calculation, non-separator PRVs are eliminated
    - Separator as the query terms containing *logical variables*
  - Non-separator PRVs have to fulfil $\mathrm{sum-out}$ preconditions
    - Preconditions 1 + 3 fulfilled by construction
    - *May be 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:
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. $X^{excl} = lv(A_i) \setminus (\mathcal{X} \setminus lv(A_i))$ count-normalised w.r.t



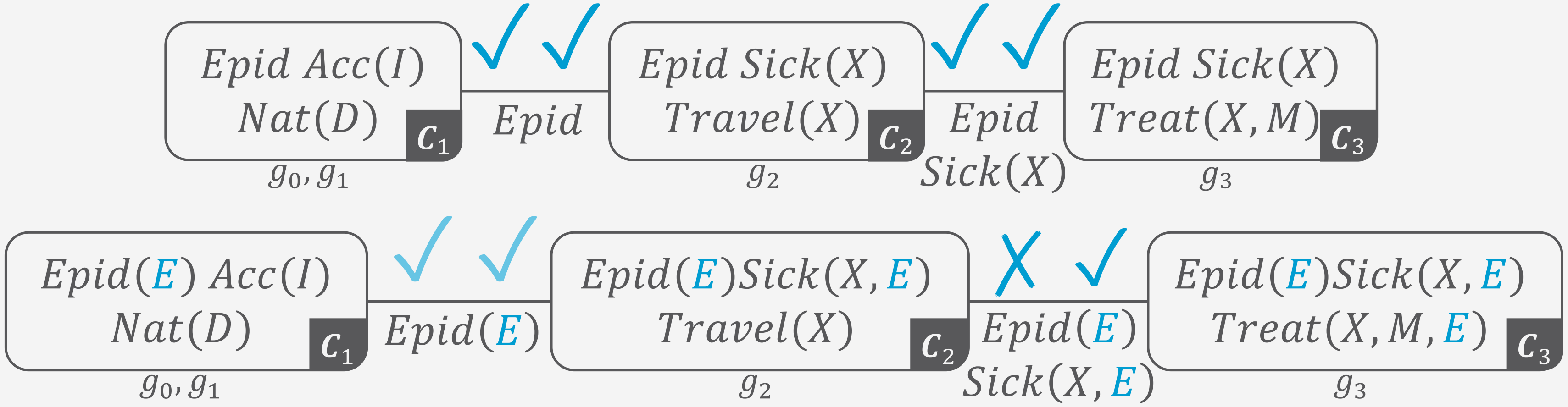
Conditions on Groundings

- For a lifted calculation of message m_{ij}, it necessarily has to hold that
  - for each PRV $A \in \left(\boldsymbol{C}_i \setminus \boldsymbol{S}_{ij}\right)$, i.e., A has to be eliminated:
    - for each separator PRV $S \in \boldsymbol{S}_{ij}$: $lv(S) \subseteq lv(A)$ (Cond. 1)
- If Cond. 1 does not hold, i.e., $lv(S) \nsubseteq lv(A)$, 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 $\boldsymbol{C}_k$, 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

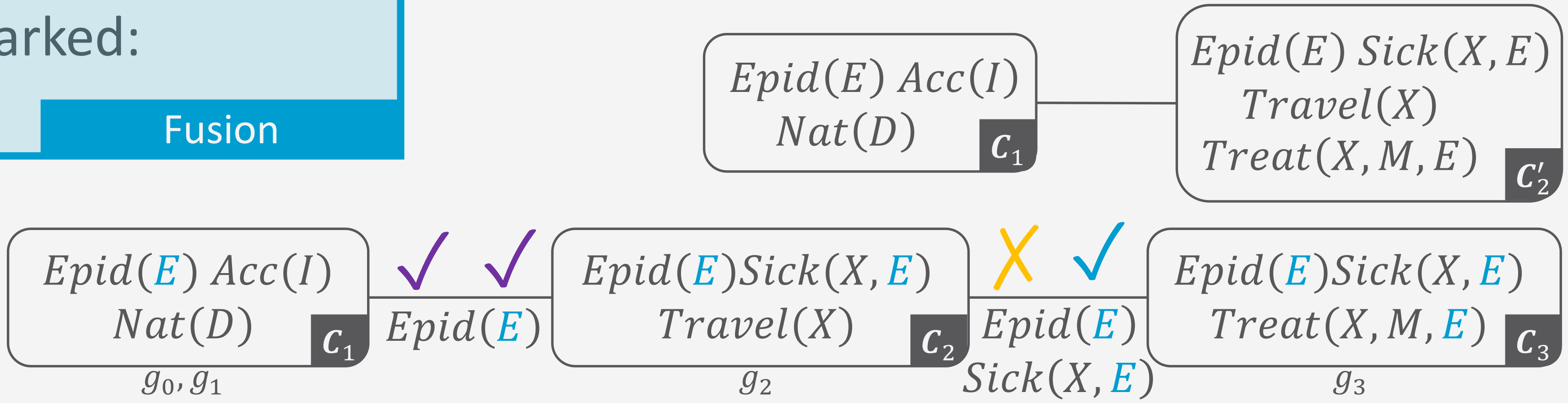


Fusion

- 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 $\boldsymbol{C}_i, \boldsymbol{C}_j$

Fusion

- Fusion an additional step after construction to guarantee lifted calculations for liftable models
- E.g., testing marks m_{23} → merge $\boldsymbol{C}_2, \boldsymbol{C}_3$ (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: $O\left(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\# w_\#}\right)$
    - Absorbing evidence complexity: $O\left(\log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$
      - Visits $\frac{1}{r} \cdot r^{w_g} \cdot n^{r_\# w_\#}$ lines, possibly exponentiates the potentials
    - At each node → $n_J \cdot O\left(\log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$
      - n_J number of nodes in FO jtree J
    - For each e evidence parfactors → $e \cdot O\left(n_J \cdot \log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$
      - Assuming $e \ll n_J$ → $O\left(n_J \cdot \log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$
  - First two steps accumulated: $O\left(n_J \cdot \log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$

LJT: Complexity

- Complexity of individual steps
  - First two steps accumulated: $O\left(n_J \cdot \log_2(n) \cdot r^{w_g - 1} \cdot n^{r_\# w_\#}\right)$
  - Message passing: $O\left(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\# w_\#}\right)$
    - Calculating one message = answering one query on a parcluster
      - Worst-case parfactor size at parcluster: $r^{w_g} \cdot n^{r_\# w_\#}$
      - Elimination of $\left|\boldsymbol{C}_i \setminus \boldsymbol{S}_{ij}\right|$ 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:

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

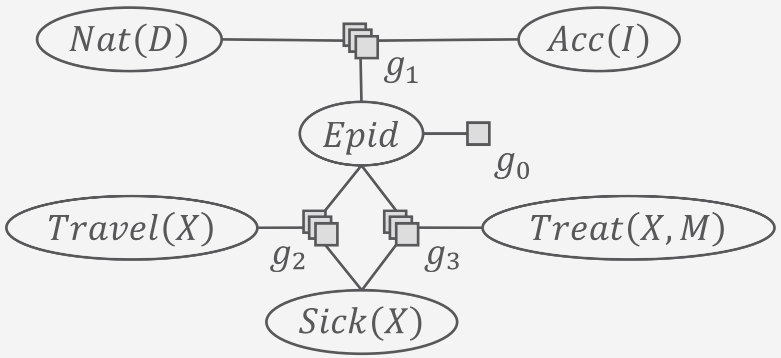
Comparison to LVE

- 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. $O\left(n_J \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\# w_\#}\right)$
  - 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: $O\left(\left(n_J + m\right) \cdot \log_2(n) \cdot r^{w_g} \cdot n^{r_\# w_\#}\right)$
  - Difference in $m \cdot n_T$ vs. $\left(n_J + m\right)$
    - 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)

LJT: Implementation

- Available at:
  - https://www.ifis.uni-luebeck.de/index.php?id=518&L=2
  - Based on the LVE implementation by Taghipour
    - Available at:
      - https://dtai.cs.kuleuven.be/software/gcfove
  - Includes an implementation of the propositional junction tree algorithm for comparison
- Input: BLOG files
  - Based on Bayesian Logic Programming Language
    - https://bayesianlogic.github.io



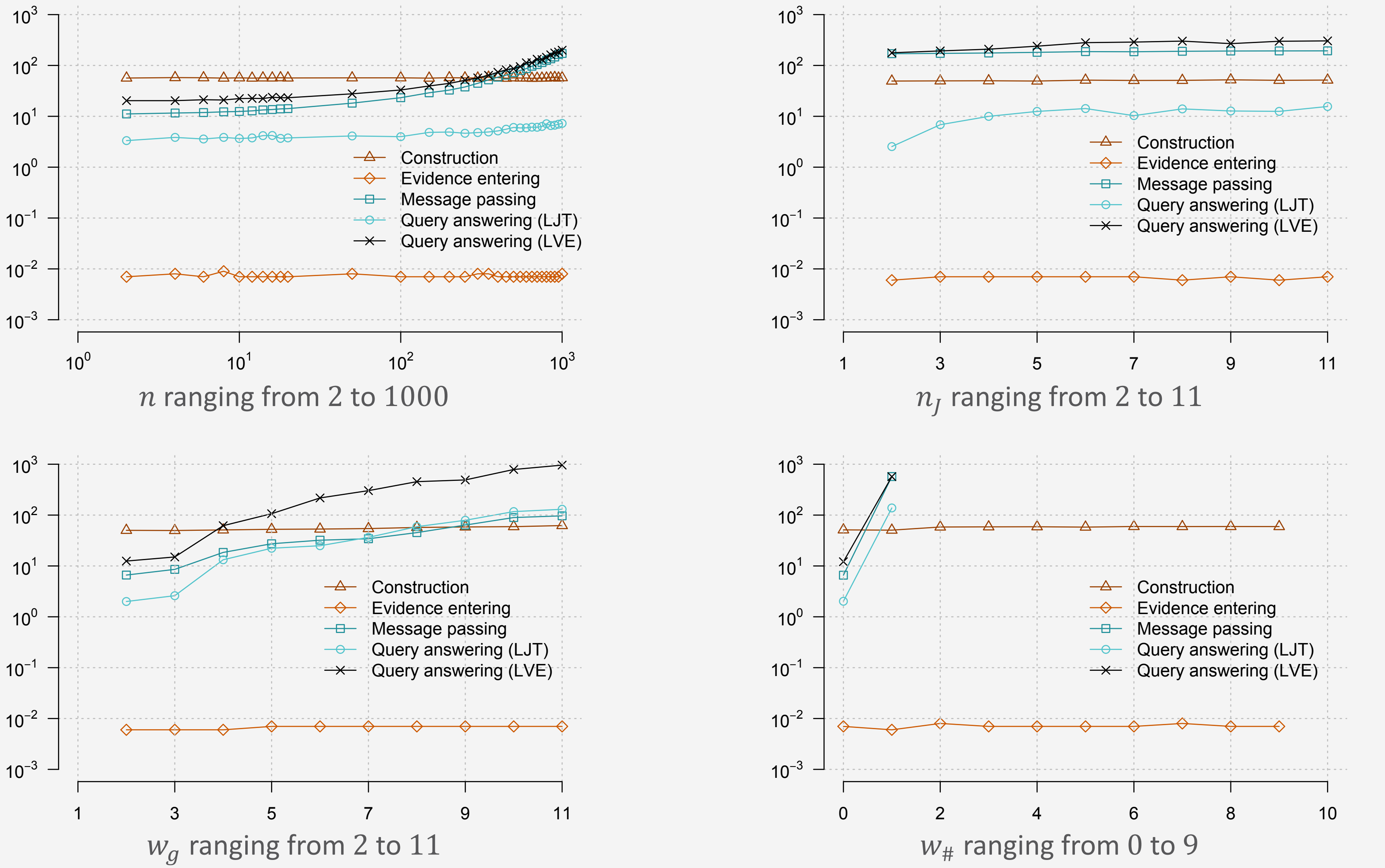
Runtimes: Increasing Domain Sizes

- Example model with all domain sizes $\in \{2,4,\dots,20,30,\dots,100,\ 200,\dots,1000\}$
- No evidence
- Queries:
  - $P(Travel(x_1))$
  - $P(Sick(x_1))$
  - $P(Treat(x_1,m_1))$
  - $P(Nat(d_1))$
  - $P(Man(w_1))$
  - $P(Epid)$

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

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

Step-wise



Runtimes in milliseconds

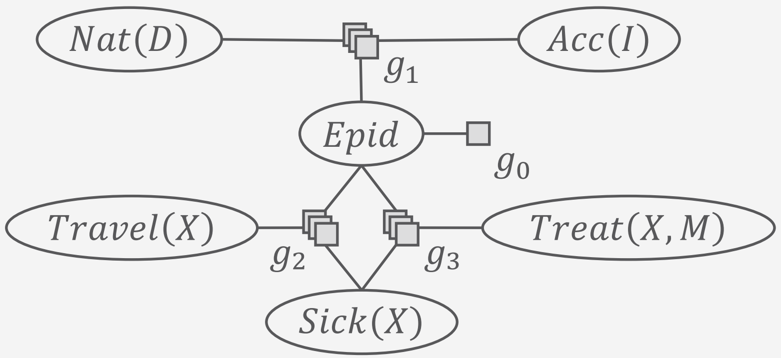
Default: $n = 1000, n_J = 3, w_g = 3, w_{\#} = 1$

Exact Inference: LJT

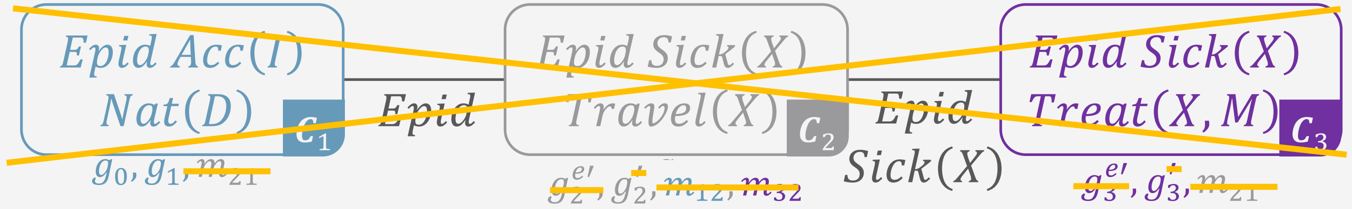
Changing Inputs

- Known as adaptive inference
  - Goal: do not start from scratch
- New queries $\{\boldsymbol{Q}'_i\}_{i=1}^m$
  - Restart query answering step: Answer queries in J
    - JLT supports *online* query answering
      - Queries not known beforehand → Stream of queries
- Changed evidence $\boldsymbol{e}'$
  - Restart with evidence handling: take original local models, handle $\boldsymbol{e}'$, pass messages, answer queries
- Changed model G'
  - Restart at beginning: Build new FO jtree, ...

If only local changes in $\boldsymbol{e}$ or G, proceed adaptively

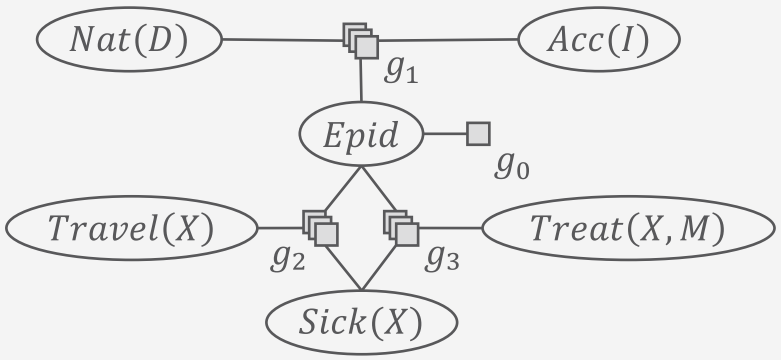


Evidence:
~~$sick(eve)$~~
$\neg travel(eve), \neg sick(eve)$
Queries:
~~$\{\{Epid\}, \{Travel(eve), Treat(eve, m_1)\}\}$~~
$\{\{sick(alice)\}, \{Treat(eve, m_2)\}\}$



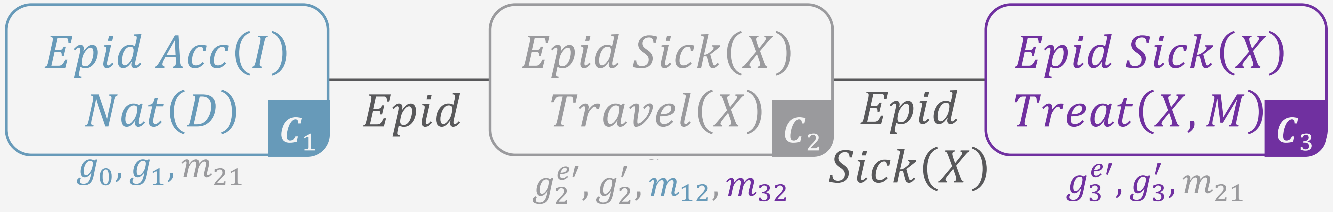
Changing Inputs and Adaptive Message Passing

- Local changes in G
  - Different potentials in parfactors
  - Changes in domain sizes (*special to relational modelling*)
  - Parfactors are removed or added
    - Maintain FO jtree properties!
    - Only worth it given local changes, otherwise build anew
- Local changes in $\boldsymbol{e}$
  - Only reset local models of parclusters covered by evidence
- Adaptive message passing
  - If changes in local model or incoming message, calculate new message
    - Otherwise: send empty message
  - Save up to half of the messages



Evidence:
$sick(eve)$

Queries:
$\{\{Epid\}, \{Travel(eve), Treat(eve, m_1)\}\}$



Interim Summary

- Motivation: Find clusters that are enough for query answering
- FO jtree: From FO dtree clusters to FO jtree parclusters
- LJT algorithm
  - Evidence handling before message passing
  - Propagation/message passing: Dynamic programming
  - Query answering: Find subgraph covering the query terms
- Runtime behaviour
  - Overhead for construction, message passing
  - Savings during query answering
  - Trade-off between those two
- Adaptive inference for local changes: adaptive message passing

Outline: 4. Lifted Inference

A. Exact Inference

i. Lifted Variable Elimination for Parfactor Models
  - Idea, operators, algorithm, complexity

ii. Lifted Junction Tree Algorithm
  - Idea, helper structure: junction tree, algorithm

iii. First-order Knowledge Compilation for MLNs
  - Idea, helper structure: circuit, algorithm

B. Approximate Inference: Sampling

- Rejection sampling
- (Lifted) likelihood sampling
- (Lifted) Markov Chain Monte Carlo sampling