The WHY in Business Processes: Unification of Causal Process Models

– Supplementary Materials –

Anonymous Author(s)

No Institute Given

1 Causal Gateways Specification

Definition 1. *A Causal "And" Gateway, denoted as AND_C, is a node in a causal execution graph $G = (V, E)$, where $AND_C \in V$, that links a source edge $(s, AND_C) \in E$ and a set of two or more target edges $\{(AND_C, t) \mid t \in T_e, T_e \subseteq V, |T_e| \geq 2\} \subseteq E$. The gateway AND_C denotes that the node $s \in V$ linked via the source edge (s, AND_C) causes the execution of all nodes in T_e linked via the set of target edges.*

Definition 2. *A Causal "Or" Gateway, denoted as OR_C, is a node in a causal execution graph $G = (V, E)$, where $OR_C \in V$, that links a source edge $(s, OR_C) \in E$ and a set of two or more target edges $\{(OR_C, t) \mid t \in T_e, T_e \subseteq V, |T_e| \geq 2\} \subseteq E$. The gateway OR_C denotes that the node $s \in V$ linked via the source edge (s, OR_C) causes the execution of some (at least one) of the nodes in T_e linked via the set of target edges.*

Concerning the general type of the "Or" gateways, we also define a specific type in which the source activity can be a cause for any possible combination among its target activities as follows:

Definition 3. *A Exhaustive Causal "Or" Gateway, denoted as OR_C^E, is a special type of causal "Or" gateway in a causal execution graph $G = (V, E)$, where $OR_C^E \in V$, that links a source edge $(s, OR_C^E) \in E$ and a set of two or more target edges $\{(OR_C^E, t) \mid t \in T_e, T_e \subseteq V, |T_e| \geq 2\} \subseteq E$. The gateway OR_C^E denotes that the node $s \in V$ linked via the source edge (s, OR_C) causes any combination of the nodes in T_e linked via the set of target edges, including one or multiple target nodes.*

Definition 4. *A Causal "Xor" Gateway, denoted as XOR_C, is a node in a causal execution graph $G = (V, E)$, where $XOR_C \in V$, that links a source edge $(s, XOR_C) \in E$ and a set of two or more target edges $\{(XOR_C, t) \mid t \in T_e, T_e \subseteq V, |T_e| \geq 2\} \subseteq E$. The gateway XOR_C denotes that the node $s \in V$ linked via the source edge (s, XOR_C) causes the execution of exactly one node in T_e linked via the set of target edges.*

2 Log Partitioning Algorithm

Algorithm 1 Log Partitioning

Log Partitioning Function:
Input:

- Event log L (a set of traces)
- *Optional:* Selection of variants V_s, each identifying a subset of traces in L; default: $V_s = \emptyset$
- *Optional:* Boolean flag `split_by_variants`, default: `false`

Output: A set of causal execution graphs G_i; initialized as $G_i = \emptyset$

if $V_s = \emptyset$ **then**
  $V_s \leftarrow$ PM.getAllVariants(L) ▷ Retrieve all variants using a process mining algorithm
end if
$T_s \leftarrow \bigcup V_s$ ▷ Assign T_s as the union of all traces in the variants V_s
if `split_by_variants` = `true` **then**
  $P_s \leftarrow V_s$ ▷ Directly assign V_s to P_s if splitting by variants
else
  $P_s \leftarrow$ SplitFunction(T_s) ▷ Invoke the `SplitFunction` on T_s to get partitions
end if
for each partition $p_i \in P_s$ **do**
  $g_i \leftarrow$ CBP.discover(p_i) ▷ Derive the causal execution graph for p
  $G_i \leftarrow G_i \cup \{g_i\}$ ▷ Add the element g_i to the set G_i
end for
return G_i

Split Function:
Input: A set of traces T
Output: A set of partitions P, where each partition contains traces from T that share the same set of activities (regardless of order).

Initialize $P \leftarrow \emptyset$ ▷ The set of partitions
Initialize a mapping $M \leftarrow \emptyset$ ▷ Maps sets of activities to partitions
for each trace $t \in T$ **do**
  $A_t \leftarrow$ Set(t) ▷ Compute the set of activities in t
  if $A_t \notin M$ **then**
    $M[A_t] \leftarrow \emptyset$ ▷ Create a new partition for this set of activities
  end if
  $M[A_t] \leftarrow M[A_t] \cup \{t\}$ ▷ Add the trace t to the appropriate partition
end for
$P \leftarrow \{M[k] \mid k \in M\}$ ▷ Collect all partitions from the mapping M
return P

3 Input Graphs to Matrix Representation Algorithm

Algorithm 2 Processing Input Graphs into a Matrix Representation

Require: Set of graphs $\mathcal{G}_\rangle = \{g_1, g_2, \ldots, g_n\}$, where $g_i = (N_i, E_i)$
Ensure: Matrix $M[node][graph]$ representing child sets
Compute the union of nodes: $N \leftarrow \bigcup_{i=1}^{n} N_i$
Initialize the matrix M:
for all node $u \in N$ and graph $g_i \in \mathcal{G}_\rangle$ **do**
    $M[u][g_i] \leftarrow \emptyset$
end for
Populate the matrix:
for all graph $g_i = (N_i, E_i) \in \mathcal{G}_\rangle$ **do**
    for all edge $(u, v) \in E_i$ **do**
        $M[u][g_i] \leftarrow M[u][g_i] \cup \{v\}$
    end for
end for
return M

4 Unification Algorithm

Algorithm 3 Processing the Matrix with the Unification Algorithm

Require: Matrix $M[node][graph]$, where columns correspond to graphs $g_1, g_2, \ldots, g_n$
Ensure: Revised matrix M' with gateway markings and viable alternatives for all OR_C gateways in M_{OR}

$M' \leftarrow M$
$M_{OR} \leftarrow \emptyset$ ▷ Initialize map sets to capture alternatives executions for OR_C
for each node $u \in M$ **do**
  Family $\leftarrow \{M[u][g] \mid g \in \{g_1, g_2, \ldots, g_n\}, M[u][g] \neq \emptyset\}$
  ▷ Step 1: AND_C Identification
  for each child set $S_i \in$ Family **do**
    IntersectsPartially $\leftarrow$ False
    for each child set $S_j \in$ Family, $j \neq i$ **do**
      if $S_i \cap S_j \neq \emptyset$ **and** $S_i - S_j \neq \emptyset$ **then** ▷ Checking no partial intersection
        IntersectsPartially $\leftarrow$ True
        break
      end if
    end for
    if not IntersectsPartially **then**
      for each occurrence of S_i in Family **do**
        $S_i \leftarrow$ "(" + S_i + ")" ▷ Mark child set as AND_C
      end for
    end if
  end for
  ▷ Step 2: XOR_C Check (only if row has 2+ sets)
  if |Family| ≥ 2 **and** $\forall S_i, S_j \in$ Family, $i \neq j, S_i \cap S_j = \emptyset$ **then** ▷ Checking family is an exclusive set
    $u \leftarrow$ "}" + u + "{" ▷ Mark node as XOR_C
  else ▷ Step 3: OR_C^E Check (only if row has 2+ sets)
    if |Family| ≥ 2 **then**
      $U \leftarrow \bigcup_{S \in \text{Family}} S$ ▷ Compute the union of all sets in Family
      if |Family| $= 2^{|U|}$ **then** ▷ Check if Family is a powerset of U
        $u \leftarrow$ "[" + u + "]" ▷ Mark node as OR_C^E
      else
        $u \leftarrow$ "$*$" + u + "$*$" ▷ Default to OR_C
        $M_{OR}[u] \leftarrow \emptyset$ ▷ Create a new map entry for this family set
        $M_{OR}[u] \leftarrow M_{OR}[u] \cup$ Family ▷ Add the Family set to the entry
      end if
    end if
  end if
end for
return M', M_{OR}

5 Unified Graph Construction Algorithm

Algorithm 4 Construct Unified Graph from Revised Matrix

Require: Revised matrix M', where each row represents a node and columns represent its child sets
Ensure: Unified graph $G_u = (N, E)$, with N as nodes and E as edges

Initialize $N \leftarrow \emptyset$ and $E \leftarrow \emptyset$ ▷ Create an empty graph
for each row $u \in M'$ with row number r **do** ▷ Iterate through nodes in the revised matrix
  $N \leftarrow N \cup \{u\}$ ▷ Add node u to N
  for each child set $S \in M'[u]$ **do**
    if $S = \emptyset$ **then** ▷ Skip empty sets
      continue
    end if
    if $|S| = 1$ **then** ▷ S contains a single element
      $N \leftarrow N \cup \{v\}$ ▷ Add node v to N
      $E \leftarrow E \cup \{(u, v)\}$ ▷ Add edge from u to v
    else if $S = \{(v_1, v_2, \ldots, v_k)\}$ **and** $label(S) =$ "(...)" **then**
      ▷ S contains multiple elements and is surrounded by round brackets
      $N \leftarrow N \cup \{\text{AND}_{Cr}\}$ ▷ Add AND_{Cr} to N
      $E \leftarrow E \cup \{(u, \text{AND}_{Cr})\}$ ▷ Add edge from u to AND_{Cr}
      for each $v_j \in \{v_1, v_2, \ldots, v_k\}$ **do**
        $E \leftarrow E \cup \{(\text{AND}_{Cr}, v_j)\}$ ▷ Add edge from AND_{Cr} to v_j
      end for
    end if
  end for
  if $label(u) =$ "}...{" **then** ▷ Handle XOR_C
    $N \leftarrow N \cup \{\text{XOR}_{Cr}\}$ ▷ Add XOR_{Cr} to N
    for each edge $(u, v) \in E$ where v is a child of u **do**
      $E \leftarrow E \setminus \{(u, v)\}$ ▷ Remove edge (u, v)
      $E \leftarrow E \cup \{(u, \text{XOR}_{Cr}), (\text{XOR}_{Cr}, v)\}$ ▷ Redirect edges via XOR_{Cr}
    end for
  else if $label(u) =$ "[...]" **then** ▷ Handle OR_C^E
    $N \leftarrow N \cup \{\text{OR}_{Cr}^E\}$ ▷ Add OR_{Cr}^E to N
    for each edge $(u, v) \in E$ where v is a child of u **do**
      $E \leftarrow E \setminus \{(u, v)\}$ ▷ Remove edge (u, v)
      $E \leftarrow E \cup \{(u, \text{OR}_{Cr}^E), (\text{OR}_{Cr}^E, v)\}$ ▷ Redirect edges via OR_{Cr}^E
    end for
  else if $label(u) =$ "$*$...$*$" **then** ▷ Handle OR_C
    $N \leftarrow N \cup \{\text{OR}_{Cr}\}$ ▷ Add OR_{Cr} to N
    for each edge $(u, v) \in E$ where v is a child of u **do**
      $E \leftarrow E \setminus \{(u, v)\}$ ▷ Remove edge (u, v)
      $E \leftarrow E \cup \{(u, \text{OR}_{Cr}), (\text{OR}_{Cr}, v)\}$ ▷ Redirect edges via OR_{Cr}
    end for
  end if
end for
return $G_u = (N, E)$

5.1 Properties of the Algorithms

The processing of the log to derive a U-CX graph consists of several steps. The first step, **variant identification**, involves splitting the log into variants by grouping traces that follow the same sequence of activities. This is accomplished using a hashing approach that stores string representations of activity sequences as hash keys, resulting in an overall computational complexity of $O(TA)$, where T is the number of traces in the log and A is the number of activities per trace.

Following the identification of variants, the process proceeds to **variant ordering**, which ensures that variants with the same set of activities, irrespective of order, are grouped together. This requires sorting the activity sequences of the variants, leading to a complexity of $O(VA \log A)$, where V represents the number of unique variants.

Once ordered, the next step is **partition identification**, where variants with identical sets of activities are combined into partitions. With pre-sorted variants, this operation can be completed in linear time with a complexity of $O(V)$.

After partitioning, the process moves to **causal discovery**, where the LiNGAM algorithm is applied to the traces within each partition to infer causal dependencies among activities. Given the cubic complexity of LiNGAM with respect to the number of activities, the overall complexity for this step is $O(TA^3)$.

The resulting causal graphs are then transformed into a **matrix representation** in the graph transformation step. Each row in this matrix corresponds to a parent activity node in the causal graphs, and columns contain sets of child activities that are causally related to the parent. The complexity of this step is $O(PA^2)$, where P is the number of partitions.

Next, the matrix undergoes processing in the **matrix unification** step to identify the causal gateways. This step involves scanning the matrix to detect partial intersections and exclusivity conditions. Checking for partial intersections and exclusivity, which dominate the computational cost, results in a complexity of $O(A^2P^2)$, while the complexity of checking for the exhaustive OR involves a simple count of the child sets and size comparison $O(1)$ using the formula:

$$|S| = 2^{|\cup S|}$$

where S is the union set of activities in all child sets. Finally, the marked matrix is transformed back into a U-CX graph in the **graph reconstruction** step. This involves processing each row of the matrix to reconstruct nodes and edges, yielding a complexity of $O(A^2)$.

Combining the complexities of all steps, the total computational complexity of the entire process is:

$$O(TA + VA \log A + V + TA^3 + PA^2 + A^2P^2 + A^2),$$

where each term corresponds to a step in the pipeline, mostly dominated by a cubic complexity in the maximal number of activities (i.e., $O(TA^3)$) in the employment of LiNGAM over the partitions.

6 Proof

In this section, we present a proof for the soundness and completeness properties of our unification algorithm. Given a set of input causal execution graphs $G_i = [G_1, \ldots, G_n]$ and a unified, extended, causal execution output graph G_U, we define soundness and completeness as follows:

Soundness: A unified causal execution graph G_U is **sound** if every causal dependency that can be derived in G_U must have originated from at least one of the input graphs in G_i.

In other words, soundness ensures that the unification process does **not introduce any new causal dependencies** that were not already present in one of the input graphs. This guarantees that G_U remains a **valid representation** of the input graphs and does not create incorrect causal dependencies.

Completeness: A unified causal execution graph G_U is **complete** if every causal dependency that appears in at least one input graph must also be derivable in G_U.

This means that no edges from the original input graphs are **lost** during the unification process. Completeness ensures that G_U **fully preserves** the structural relationships found in the input graphs and does not omit any existing information.

Lemma 1. *The models G_i corresponding to each partition are each both sound and complete w.r.t to the partition traces.*

Theorem 1. *(Soundness of the unified model) For each causal execution dependency that the unified model expresses, the same causal execution dependency is expressed by one of the underlying partition models.*

Proof. We define a path $p(n_s, n_t)$ between a source node n_s and a target node n_t as either a single direct edge (n_s, n_t) or a gateway-mediated path, consisting of a pair of edges (e_s, e_t), where $e_s = (n_s, GATE)$ and $e_t = (GATE, n_t)$. The node $GATE$ represents one of the causal gateway types: AND_C, OR_C, OR_C^E, or XOR_C, whether split or join. Next, our formalism corresponds to a split gateway type, without the loss of generality, the proof is the same for all join gateway, with the difference of having everything mirrored in its direction.

To prove the theorem, we must show the following.

For every node n_j in G_U, we consider a set of target nodes $N_t = \{n_{t1}, \ldots, n_{tk}\} \subseteq G_U.nodes$, which represents the potential target nodes reachable from n_j.

The type of $GATE$ determines how these target nodes are grouped into subsets. As a preceding step, if the type is an AND_C gateway, we replace the group of all corresponding target nodes in N_t with a single literal as a 'syntactic sugar' for that group. Subsequently, if the type is a XOR_C, then the partitioning is an exclusive split of N_t. If the type is a OR_C^E, then the partitioning is the powerset of N_t. For a type of a non-exhaustive OR_C, the partitioning is some partial set of the powerset that can only be resolved if explicitly specified.

Let $N_{\hat{t}_k} \subseteq N_t$ be a subset of these target nodes forming a **family of sets** indexed by k, where each $N_{\hat{t}_k}$ represents a distinct path group.

The corresponding family of paths is given by:

$$P = \{P_k \mid P_k = \{p(n_j, n_x) \mid n_x \in N_{\hat{t}_k}\}, k \in K\}$$

and the associated family of edge sets is:

$$E = \{E_k \mid E_k = \{(n_j, n_x) \mid n_x \in N_{\hat{t}_k}\}, k \in K\}.$$

If each path group P_k is determined by the behavior of $GATE$ type, then there exists a graph $g_i \in G_i$ such that each edge set E_k is exactly the set of edges in g_i that originate from n_j, meaning:

$$E_k = \{e \in g_i.edges \mid e = (n_j, n_x) \text{ for some } n_x \in g_i.nodes\}.$$

In other words, the set E_k must contain *all and only* the edges originating from n_j in g_i. This ensures that the outgoing edges from n_j in G_U are fully determined by a single graph g_i.

For the case where the set of edges E is a single direct edge $\{(n_j, n_x)\}$, we need to show that there exists a graph $g_i \in G_i$ where $E = \{e \in g_i.edges \mid e = (n_j, n_x) \text{ for some } n_x \in g_i.nodes\}$. An edge (n_j, n_x) is inferred in G_U according to our algorithm when n_x is a single descendant of n_j in at least one or more $g_i \in G_i$, i.e., the family set corresponding to n_j in $M[n_j][*]$ contains either elements equal to $\{n_x\}$ or empty sets $\{\emptyset\}$. Thus, satisfying the condition.

For the case where the set of edges E includes more than a single edge, corresponding to a family of gateway-mediate paths, we prove the theorem by the case of each specific gateway type.

For the case of an XOR_C gateway, we establish how the original set of target nodes in N_t was formed in the algorithm. Any edge (n_j, n_x) in E_k was inferred as part of a path $p(n_j, n_x)$ that includes an XOR_C gateway. Specifically, our algorithm labeled the row $M[n_j][*]$ as XOR_C if and only if n_x was part of an exclusive family set in a specific cell $M[n_j][g_i]$, and n_x was included in that cell only if it was originally identified as a descendant of n_j in some $g_i \in G_i$.

Thus, the condition for an XOR_C gateway is satisfied.

For the case of an OR_C^E gateway, we establish how the original set of target nodes in N_t was formed in the algorithm. Any edge (n_j, n_x) in E_k was inferred as part of a path $p(n_j, n_x)$ that includes an OR_C^E gateway. Specifically, our algorithm labeled the row $M[n_j][*]$ as OR_C^E if and only if that row was equal to $\mathcal{P}\left(\bigcup_{g_i \in G_i} M[n_j][g_i]\right)$, where $\mathcal{P}(\cdot)$ denotes the **power set** operator.

Moreover, (n_j, n_x) appears in that row only if n_x was originally identified as a descendant of n_j in some $g_i \in G_i$, and since n_x is included in the union, it must also belong to the power set. Consequently, each E_k has a corresponding cell in the row $M[n_j][g_i]$, indicating that n_x was indeed part of the power set of all possible children of n_j.

Thus, the condition for an OR_C^E gateway is satisfied: we obtain every possible combination of descendants for n_j in the power set, ensuring (n_j, n_x) was drawn

from at least one graph g_i, thereby enforcing the exhaustive nature of the OR_C^E gateway.

For the case of a non-exhaustive OR_C gateway, we establish how the original set of target nodes in N_t was formed in the algorithm. Unlike the OR_C^E case, where the row in $M[n_j][*]$ corresponds to the power set of possible descendants, a non-exhaustive OR_C gateway represents a **subset** of the power set. That is, the annotated row in the matrix satisfies: $M[n_j][*] \subset \mathcal{P}(\cdot)$.

Without an explicit specification of the target node sets in N_t, it is **not possible** to determine whether each edge set E_k has a corresponding graph g_i. This is because the partial subset selection means that some combinations of descendants may not correspond directly to any single input graph.

However, if an explicit specification of the sets in N_t is disclosed, it resolves the association of each E_k to a corresponding g_i. That is, once the partial selection is defined, we can establish which elements of the power set correspond to edges derived from some subgraph g_i. Consequently, each E_k has a corresponding cell in the row $M[n_j][g_i]$, ensuring that each edge (n_j, n_x) is accounted for within some input graph.

Thus, the condition for a non-exhaustive OR_C gateway is satisfied: the inferred edges result from a partial selection of possible descendant combinations, whose explicit specification allows the correct association of each E_k with at least one graph g_i.

Theorem 2. *(Completeness of the unified model) For each causal execution dependency expressed by any of the partition models, the same causal execution dependency is expressed by the unified model.*

Proof. For every node $n_j \in g_i.nodes$, where $g_i \in G_i$, we consider a set of target nodes $N_t = \{n_{t1}, \ldots, n_{tk}\} \subseteq g_i.nodes$, where $(n_j, n_{tk}) \in g_i.edges$. We prove that G_U is complete by showing that for every target node $n \in N_t$, either $(n_j, n) \in G_U.edges$ or there exists a path $p(n_j, n)$ in G_U.

With respect to the set of targets N_t corresponding to some g_i, the following cases are considered.

The first case is when the set of targets includes only a single element n_t. This occurs either when n_t is a target only in a single $g_i \in G_i$ and does not appear as a target in any other $g_j \in G_i$ where $g_i \neq g_j$, or when n_t is a target in multiple graphs $G_k \subset [G_1, \ldots, G_n]$, but the target sets of all remaining graphs G_k are empty.

According to our algorithm, a single-element target in a single g_i would correspond to a row $M[n_j][*]$ with a column for g_i containing n_t and all remaining columns for all remaining graphs in $G_i \setminus g_i$ containing $\emptyset$. Such a row would not have been annotated. As a result, the direct edge (n_j, n_t) is included in $G_U.edges$.

If n_t appears as a target in multiple graphs $G_k \subset [G_1, \ldots, G_n]$, but no other target nodes exist in the remaining graphs, then the corresponding row $M[n_j][*]$ will have multiple cells containing n_t and all remaining cells corresponding to $G_i \setminus G_k$ containing $\emptyset$. Since such a row would not have been annotated in the algorithm, this also entails that the direct edge (n_j, n_t) is included in $G_U.edges$.

We now extend the analysis to cases where the set of target nodes N_t contains multiple elements. This entails two subsequent steps.

First, if the set of elements in N_t corresponding to g_i has no partial intersection with any of the other target sets, this would correspond to a row $M[n_j][*]$ with a cell containing N_t that has no intersection with the sets in the other cells. Such a row would be annotated as 'AND_C', thus including each node $n \in N_t$ in edges in G_U of the form (AND_C, n). As in the previous case, for the next step, we consider such sets to be replaced by a single literal n as a syntactic sugar.

Thus, as a second step, a target set of elements in N_t corresponding to g_i may be an element of an exclusive set of sets (i.e., the targets of all graphs in G_i). In this case, according to the algorithm, the cells in the row $M[n_j][*]$ would have been annotated as XOR_C, and as a result all nodes $n \in N_t$ would have been added in edges in G_U of the form (XOR_C, n), with the edge (n_j, XOR_C) also added to G_U. This would thus entail the inclusion of the path $p(n_j, n) = \{(n_j, XOR_C), (XOR_C, n)\}$ in G_U. If the node n denotes a syntactic sugar for the case of an AND_C above, the path extends further to the set $\{(n_j, XOR_C)$, (XOR_C, AND_c), $(AND_c, n)\}$ in G_U.

If the set of elements in N_t corresponding to g_i is not an element of an exclusive set of sets (i.e., the targets of all graphs in G_i), it may be an element in the powerset of the union of all target sets of all graphs in G_i. In this case, according to our algorithm, the cells in the row $M[n_j][*]$ would have been annotated as OR_C^E, and all nodes $n \in N_t$ would have been added in edges in G_U of the form (OR_C^E, n), with the edge (n_j, OR_C^E) also added to G_U, thus entailing the inclusion of the path $p(n_j, n)$ in G_U.

Eventually, considering the above two alternatives are not met and the set of elements in N_t corresponding to g_i is neither part of an exclusive set, nor a part of a powerset, concerning the set of targets of all graphs in G_i. In this case, according to our algorithm, the cells in the row $M[n_j][*]$ would have been annotated as OR_C, and all nodes $n \in N_t$ would have been added in edges in G_U of the form (OR_C, n), with the edge (n_j, OR_C) also added to G_U, thus entailing the inclusion of the path $p(n_j, n)$ in G_U. Consequently, in all the cases where the set of target nodes N_t contains multiple elements, for every target node $n \in N_t$, there exists a gateway-mediated path $p(n_j, n$ in G_U that contains either an XOR_C gateway, an OR_C^E gateway, or an OR_C gateway, thus satisfying our condition for completeness.