

Master Thesis Report

Process Enhancement by Incorporating Negative Instances in Model Repair

Dfg Long-Term Dependency Discovery

Kefang Ding

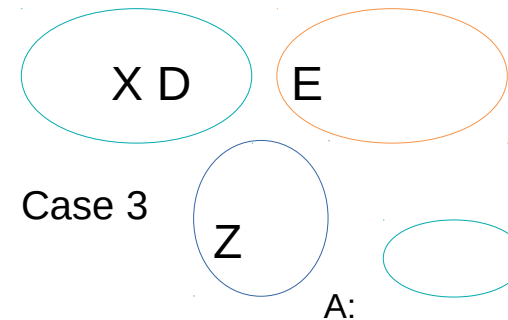
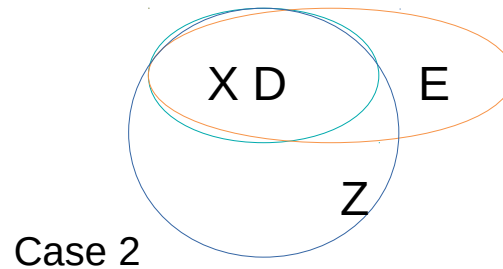
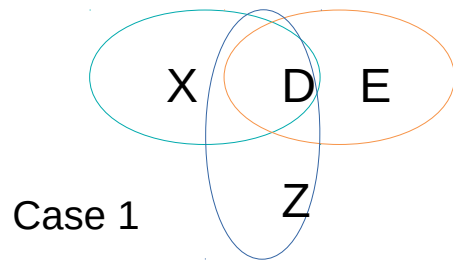
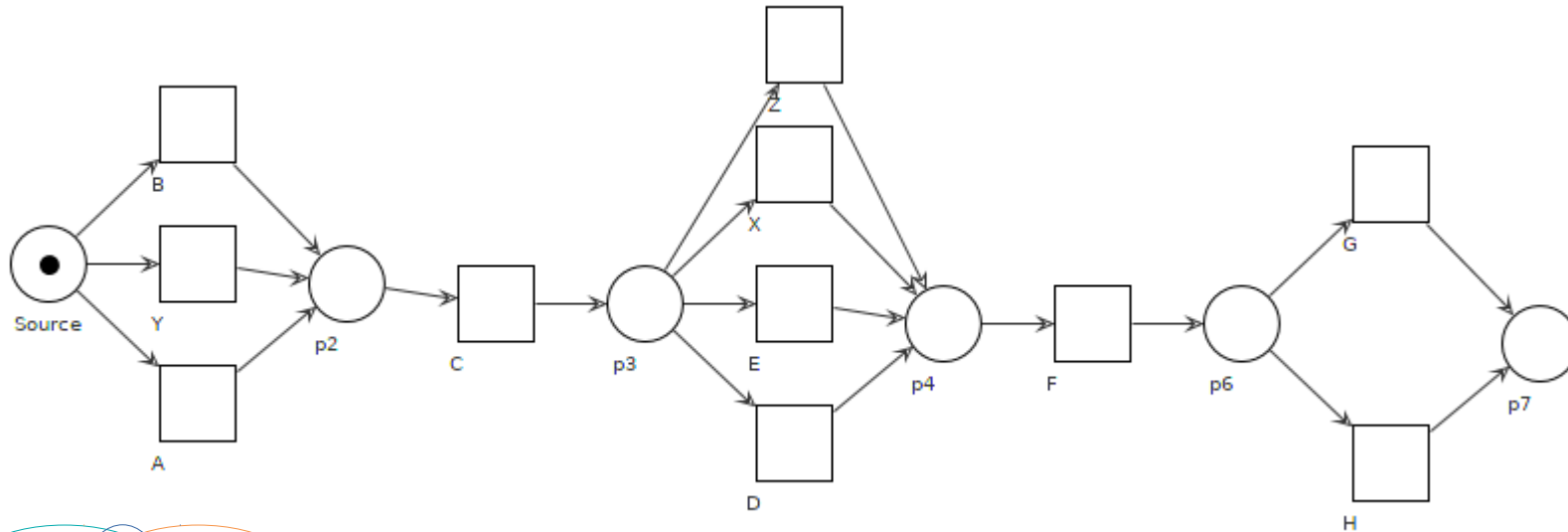
Outlines

- **Problem Introduction**
- **Preliminary**
- **Method Design**
- **Method Implementation**
- **Method Evaluation**
- **Appendix**
 - Plugin Development

Problem Introduction

- **Dfg method can not deal with long-term dependency.**
 - Can't discover it
 - Can't remove it
 - Can't find structure change
- Given Petrinet discovered by the dfg methods, event log, KPIs, we need to discover the long-term dependency, remove it or change its structure w.r.t. the event log.

Analysis - $\{A,B,Y\}$ to $\{D,E,X,Z\}$



Case 1: A set means A has long-term dependency with X and D.

B has long-term dependency with D and E.

Y has long-term dependency with D and Z.

The intersection of A and B means that they have both long-term dependency with D

A:

B:

Y:

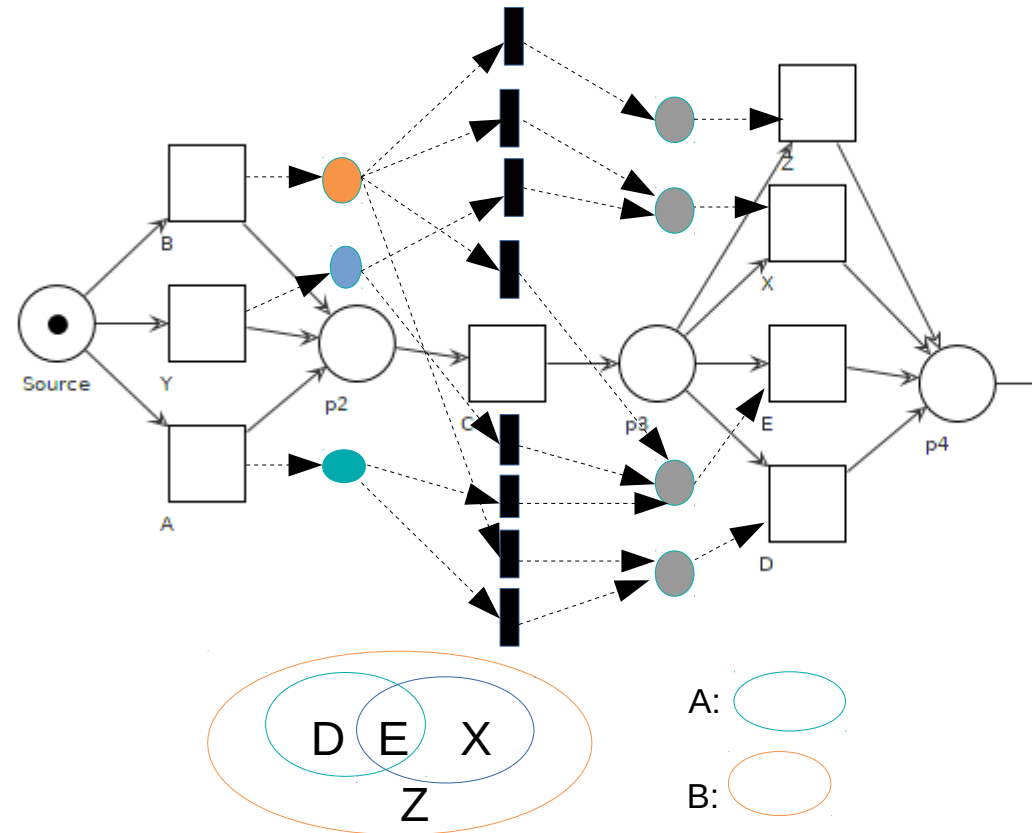
Method Design

- **Add long-term constraints**

- Add places after xor join structure
 - ✓ If any of {A,B,Y} has long-term dependency, we create post places for each branch,

$$\Rightarrow P[A], P[B], P[Y]$$
- Add places before xor split structure
 - ✓ If any of {X,D,E,Z} has long-term dependency with xor join before them, create a pre place for each of them
- Add silent transition to connect places
 - ✓ If there is connection of any combination from {A,B,Y} * {X,D,E,Z}, we create a silent transition
 - ✓ Connect the silent transition and places

$$[\text{postplace}] \rightarrow \text{tau} \rightarrow [\text{preplace}]$$



Case in graph:

A has long-term dependency with D,E;

Y has long-term dependency with E,X;

B actually has no long-term dependency with next xor, but because other nodes in the same structure has, so we also create post places for it

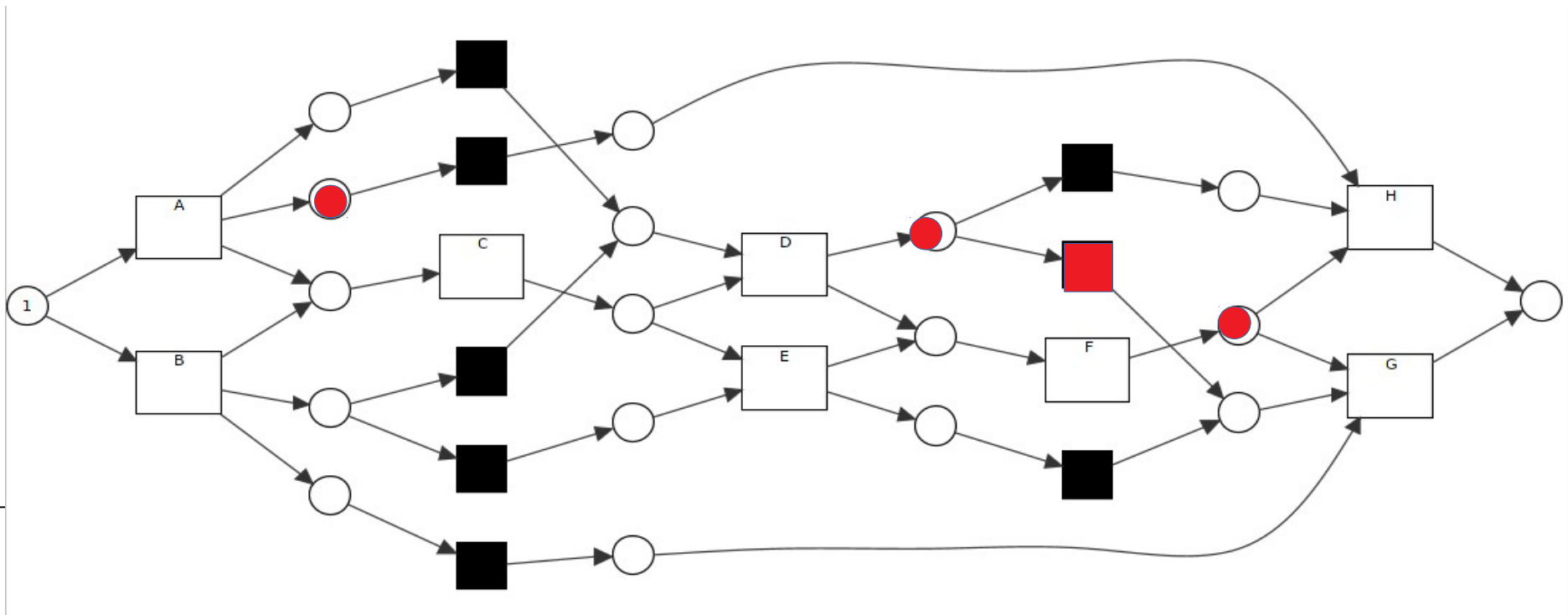
E shows in every cases, but because other nodes in the same structure has long-term dependency, so we create pre place for it

Method Design

- **Keep Model Sound**

Xor_1:{B,Y,A}, Xor_2: {Z,X,E,D}, Xor_3:{G,H}

- In the previous way to add the silent transition & places in Xor_1 and Xor_2 , the model is sound
- But not work for cross cases :: $\langle A,C,D,F,H \rangle, \langle B,C,D,F,G \rangle, \langle B,C,D,E,G \rangle$
 - ✓ Create cross long-term dependency pair $\{x01,x02\}, \{x02,x03\}, \{x01,x03\}$
 - ✓ $\langle A,C,D,F \dots ?? \rangle$ it can't complete, because there are another choices appeared.



Method Design

Soundness requirements

Option to complete

Whatever happens, an instance can always mark the sink place

Proper completion

On completion, only the sink place is marked, and it is marked only once

No dead tasks

No transition is dead

The short-circuited net is bounded, contains no dead transitions, but is not live. As a result, completion is not always possible.

The following transitions are not live in the short-circuited net

1. **G**
2. **A**
3. **D**
13. **DH**
14. **DG**
15. **BG**
16. **C**
17. **E**

The following diagnostic information assumes that there exists a part of the state space from which completion is still possible. Clearly, to avoid losing the option to complete, behavior should be restricted to this part. Thus, any transition leaving the part should be disabled.

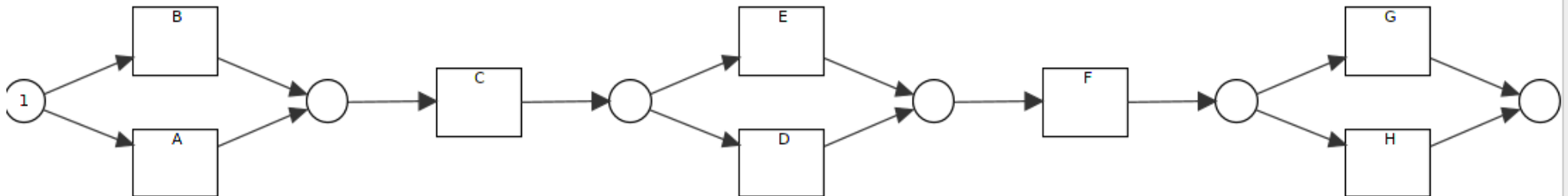
Disabling the following transitions at the following (reachable) markings effectively would restrict the behavior to the part from which completion is possible:

1. Transition **DG** at marking [sink 6, Place Before ABH, Place After DGH].
2. Transition **DG** at marking [Place After AGH, sink 5, Place After DGH].
3. Transition **DH** at marking [sink 6, Place After BGH, Place After DGH].
4. Transition **DG** at marking [Place Before ABH, sink 5, Place After DGH].
5. Transition **DG** at marking [sink 6, Place After AGH, Place After DGH].
6. Transition **DH** at marking [sink 5, Place After BGH, Place After DGH].
7. Transition **DH** at marking [sink 6, Place Before ABG, Place After DGH].
8. Transition **DH** at marking [Place Before ABG, sink 5, Place After DGH].

Method Design

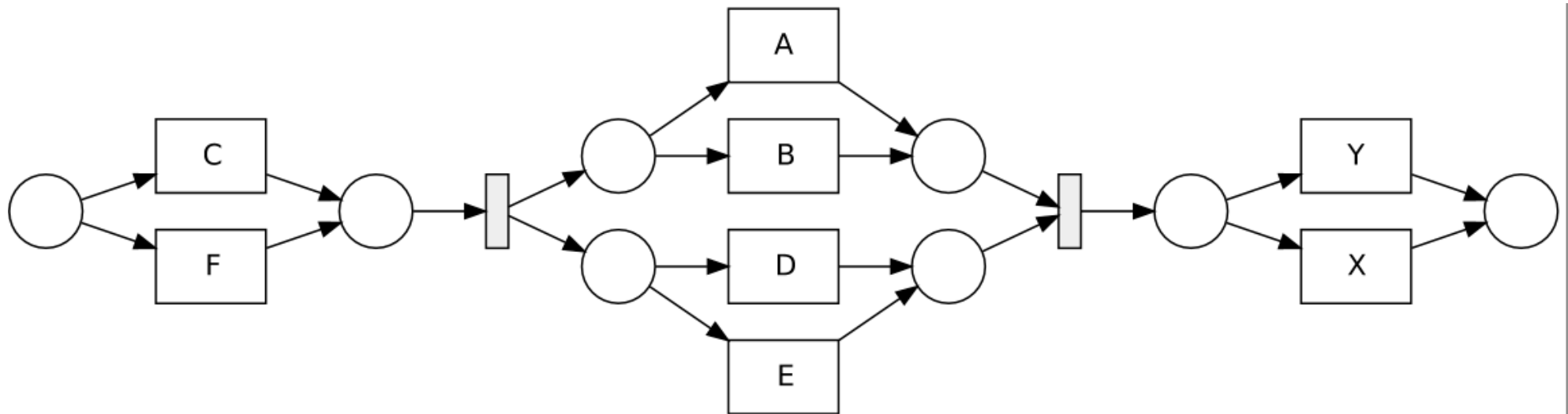
- **Use Binary Relation**

- Because the complexity of rules transitions, so we only create the connection of most nearest relation.
 - ✓ Sequence
 - $\{A,B\} \& \{E,D\} :: \{E,D\} \& \{G,H\}$
- Consequence:
 - ✓ not able to discover the cross-term dependency;
 - ✓ model less precise



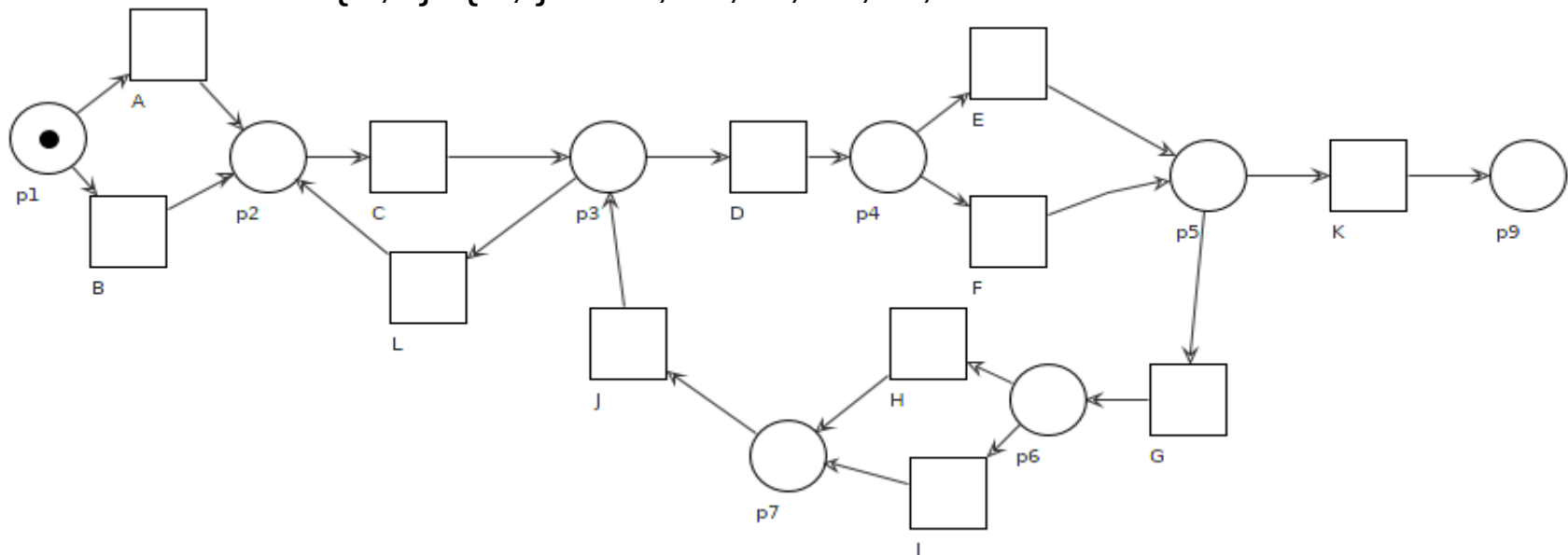
Method Design

- **Use Binary Relation – case variants:: Parallel**
 - $\{C,F\} \& \{A,B\} // \{C,F\} \& \{D,E\} // \{A,B\} \& \{X,Y\} // \{D,E\} \& \{X,Y\}$
 - $\{A,B\} \& \{D,E\}$, they have long-term dependency
 - ✓ In traces with $\langle A,D \rangle, \langle B,D \rangle, \langle B,E \rangle$



Method Design

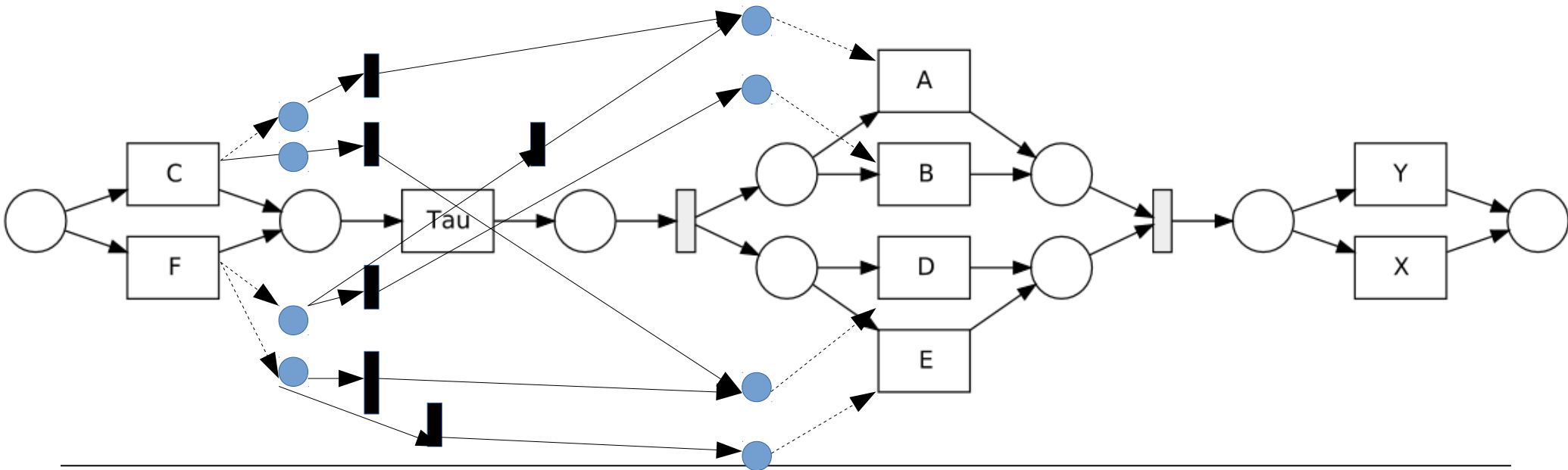
- **Use Binary Relation – Case Variants:: loop**
 - Every loop is an xor structure, to choose go back or forward
 - ✓ $\text{loop}\{C, \{L, D\}\}$ create long-term candidate: $\{A, B\} \& \{C, L\}$
 - Choose A, without loop: $\langle A, C, D \rangle$
 - Choose B, with loop: $\langle B, C, L, C, D \rangle$
 - ✓ Xor in the loop
 - Candidate from branch
 - $\{E, F\} \& \{G\}$ With E, no loop, with F, go loop $\langle E, G \rangle, \langle F, K \rangle$
 - $\{E, F\} \& \{H, I\} :: \langle E, H \rangle, \langle F, H \rangle, \langle F, I \rangle$



Method Design

- **Treat Parallel & Loop Specially**

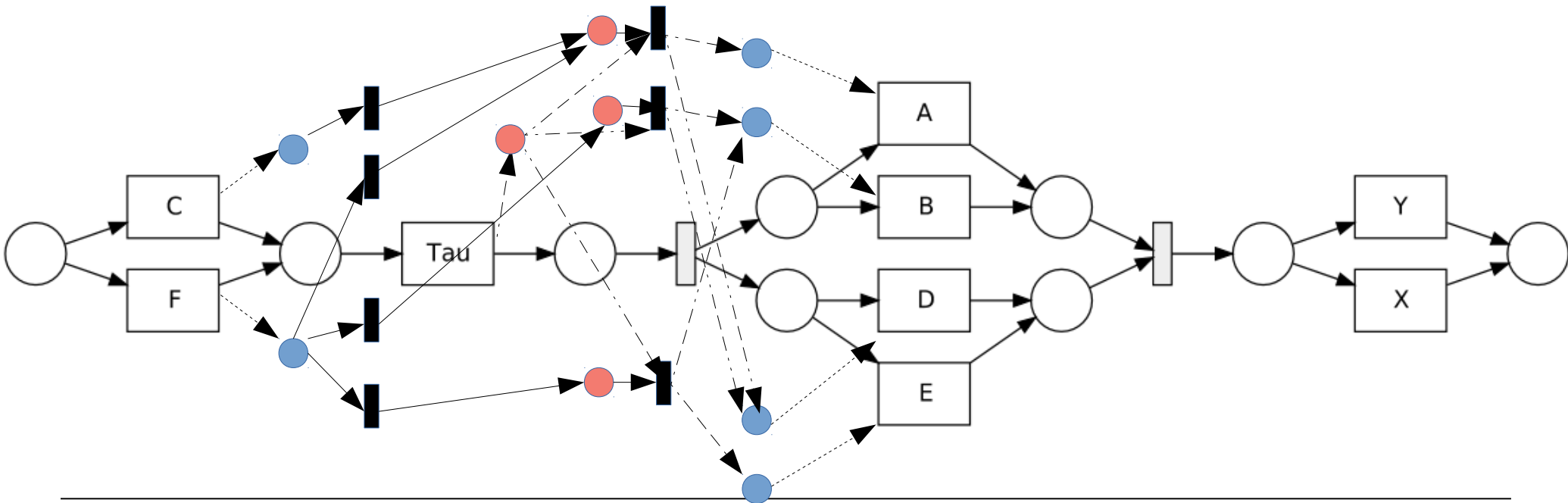
- Add control transitions for the xor in parallel & loop
- Add silent transitions before the main structure
 - ✓ $\langle C, A, D \rangle$, $\langle F, B, D \rangle$, $\langle F, B, E \rangle$, $\langle F, A, D \rangle$



Method Design

- **Treat Parallel & Loop Specially**

- Add control transitions for the xor in parallel & loop
- Add silent transitions before the main structure
 - ✓ $\langle C, A, D \rangle$, $\langle F, B, D \rangle$, $\langle F, B, E \rangle$, $\langle F, A, D \rangle$



Method Design

- **Begin From Process Tree**

- Since the long-term dependency depends also on the block that includes xor, so we use process tree to get block easier
- Long-term dependency needs the order of xor structures, visiting process tree by Depth-First-Search gives us the order of xor

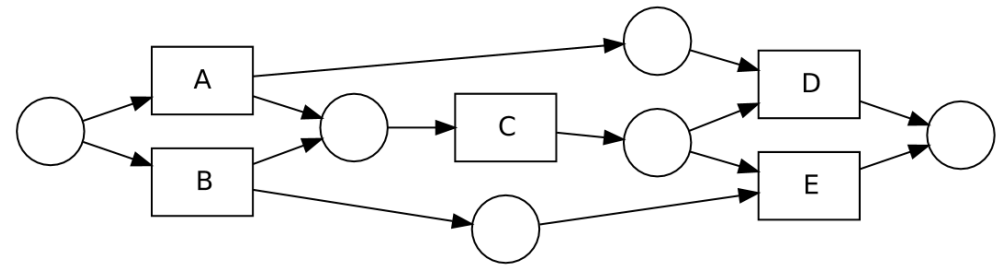
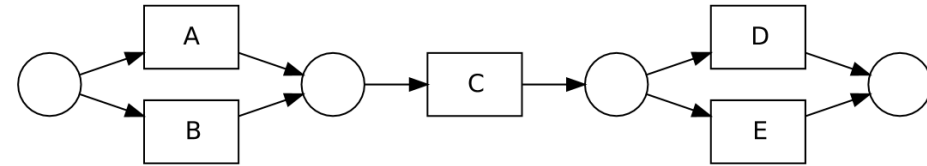
- **Constraints:**

- Petri net not corresponding to process tree!! Some petri net can't transfer to process tree
- Further work: discovery long-term dependency directly on petri net

Method Implementation

- **Methods Design**

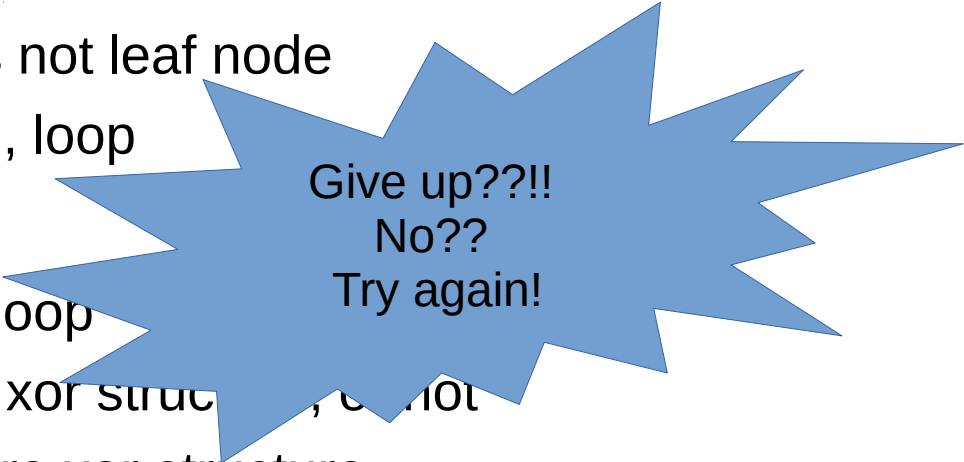
- Get all possible combinations of xor structure in Petri net, one combination includes:
 - ✓ one is xor join to create different precondition
 - ✓ one is xor split to decide which events to choose
 - ✓ Eg: {A,D}, {A,E}, {B,D},{B,E}
- Check event log, if they have specific paths through xor structure
 - ✓ <ACD, BCE>
 - $\#\{A,D\}=\text{true}, \#\{A,E\}=\text{false},$
 $\#\{B,D\}=\text{false}, \#\{B,E\}=\text{true}$
- Add places into Petri net to constrain the choices
 - ✓ $A \rightarrow o \rightarrow D, B \rightarrow o \rightarrow E$



Method Implementation

- **Obstacles**

- Diversity of xor structure
 - ✓ Xor pure :: all the children are leaf nodes
 - ✓ Xor impure :: at least one child is not leaf node
 - ✓ Xor position :: sequence, parallel, loop
- Complexity of impure xor
 - ✓ Branch is sequence, parallel, or loop
 - ✓ Substructure of branch has pure xor structure, or not
 - ✓ Substructure of branch has impure xor structure
- Difficulty to find the order of xor structure
 - ✓ sequence
 - ✓ parallel
 - ✓ loop



Give up??!!
No??
Try again!

Method Implementation – Simplify it

- **Decision**

- Exclude the nested xor structure
- Exclude the binary relation in parallel and loop
- Only consider the petri net which can be transferred from Process Tree

- **Current Progress**

- Managed to build the xor pair from process tree
- Managed to add places into the Petri net

- **Q&A**