Master Thesis Report

Process Enhancement by Incorporating Negative Instances in Model Repair

Group Report

Kefang Ding

19 Feb 2019

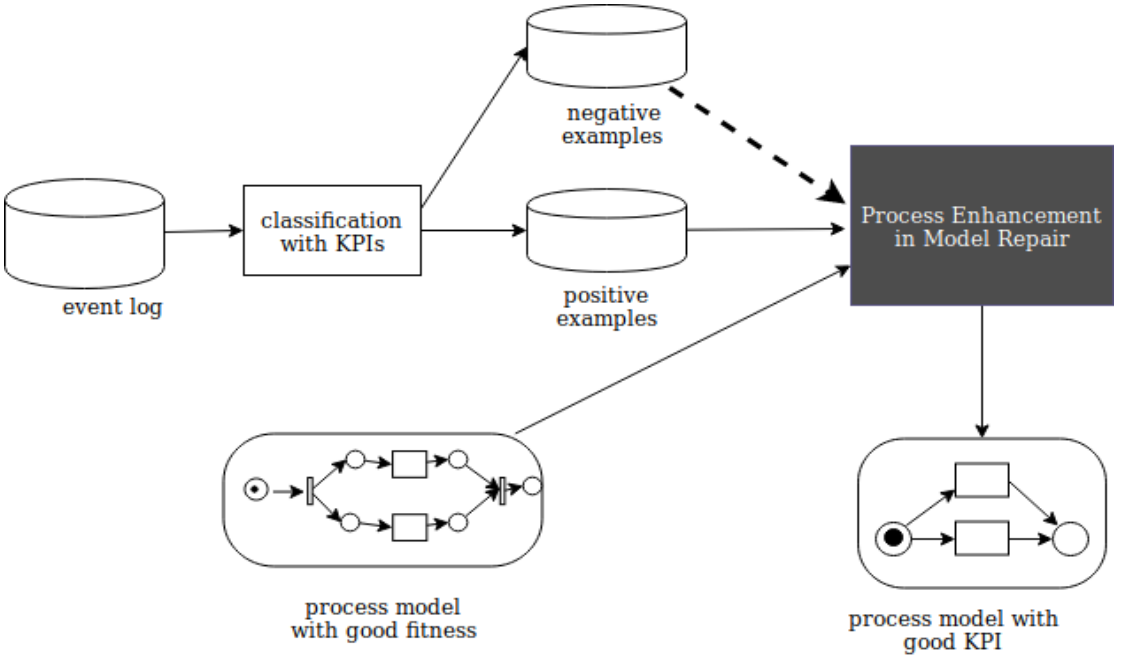
RWTH AACHEN UNIVERSITY

Outlines

- **Problem Introduction**
- **Algorithm & Implementation**
  - Add long-term dependency
  - Create dfg model
- **Demo Presentation**
- **Evaluation**

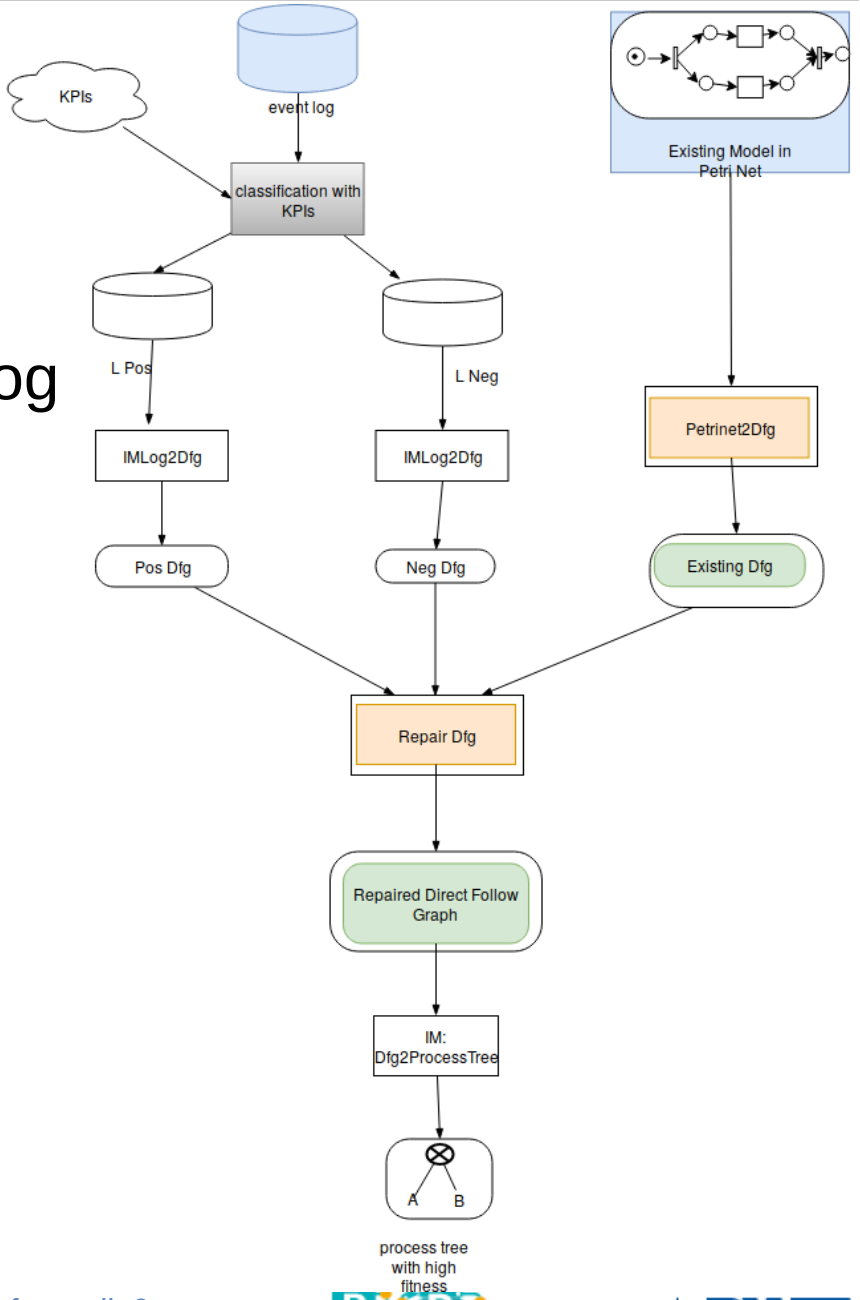
Problem Introduction

- **Description**
  - Given event log, process model and KPIs, how to incorporate negative KPIs outcomes to repair the process model for better performance?
- **Input**
  - Event log
  - Existing process model
  - KPIs
- **Output**
  - Repaired process model



Algorithm – generate dfg model

- **Items**
  - Weight from existing dfg
    - Transition system
  - Weight from positive event log
    - Inductiver Miner
  - Weight from negative enent log
    - Inductiver Miner
  - Control Parameters
    - Ext 0-1
    - Pos 0-1.0
    - Negative 0-1.0



Algorithm – generate dfg model

- **Directly-follows relation**
  - Existing model, positive and negative event log

$$W(A,B) := W(E_{G_{ext}}(A,B)) + W(E_{G_{pos}}(A,B)) - W(E_{G_{neg}}(A,B)), with$$

$$W(E_{G_{ext}}(A,B)) = C_{ext} \bullet \frac{1}{|*|}, *\text{the set of all possible activities}$$

after A, $|*|$ is the size of this set.

$C_{ext}$is the control weight on existing model from Plugin.

$$W(E_{G_{pos}}(A,B)) = C_{pos} \bullet \frac{Cardinality_{pos}(E(A,B))}{Cardinality_{pos}(E(A,*))},$$

$$W(E_{G_{neg}}(A,B)) = C_{neg} \bullet \frac{Cardinality_{neg}(E(A,B))}{Cardinality_{neg}(E(A,*))},$$

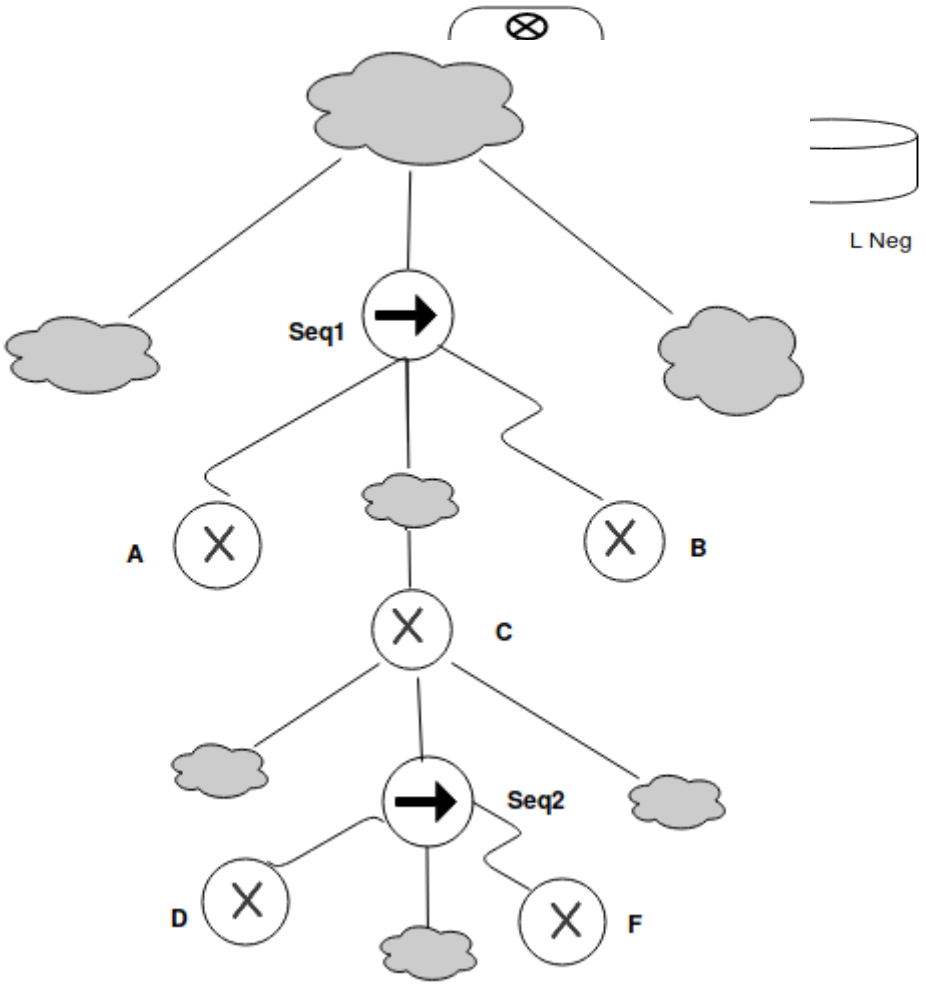
  - Keep this directly-follows relation if

$$W(A,B) > threshold, with -1 < W(A,B) < 2$$

  - Choose, threshold=0.5

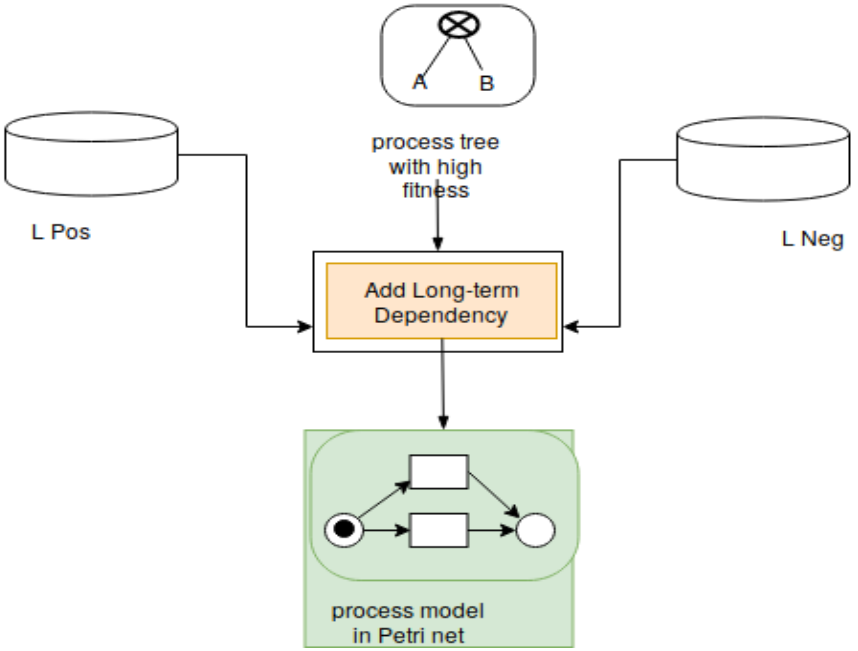
Algorithm – add long-term dependency

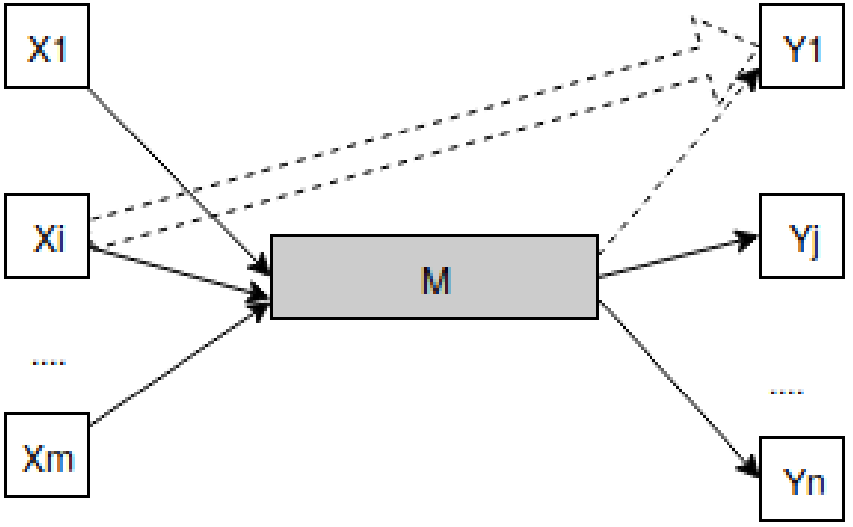
- **Long-term dependency**
  - Choices Dependency
    - exclusive blocks => xor bloc
    - not loop
  - Partial Order
    - Least Common Ancestor is
      - A <C < B, D<F
    - In same level
      - A,B,C pair
      - D,F pair



Algorithm – add long-term dependency

- **Long-term dependency**
  - Choices Dependency
    - ✔ exclusive blocks => xor block
    - ✔ not loop
  - Relation xor branches
    - ✔ Significant correlation





Algorithm – add long-term dependency

- **Correlation Defintion**
  - New generated model, positive and negative event log

$$Wlt(XORB_X, XORB_Y) = Wltext(XORB_X, XORB_Y) + Wltpos(XORB_X, XORB_Y) - Wltneg(XORB_X, XORB_Y), with$$

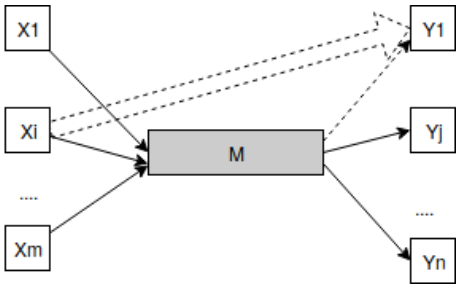
$$Wltext(XORB_X, XORB_Y) = C_{model} \bullet \frac{1}{|XORB_*|}, XORB_*$$

$*$ is the set of all xor branches from $XORB_X$

$$Wltpos(XORB_X, XORB_Y) = C_{pos} \bullet \frac{F_{pos}(XORB_X, XORB_Y)}{F_{pos}(XORB_X, *)}$$

$$Wltneg(XORB_X, XORB_Y) = C_{neg} \bullet \frac{F_{neg}(XORB_X, XORB_Y)}{F_{neg}(XORB_X, *)}$$

$F_{pos}(XORB_X, XORB_Y), F_{neg}(XORB_X, XORB_Y)$ are the frequency of coexistence of $XORB_X, XORB_Y$

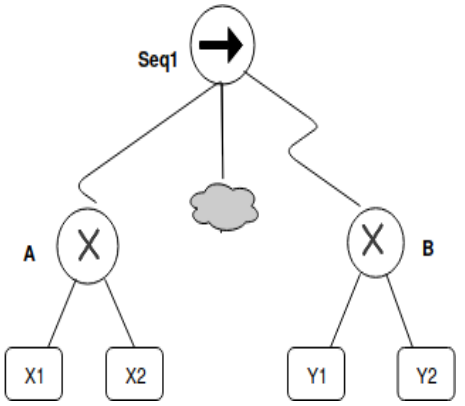


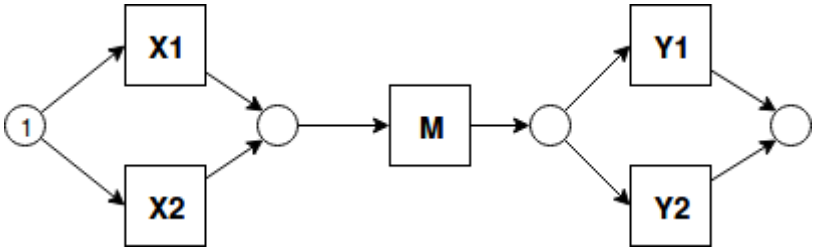
  - Significant Correlation

$$Wlt(XORB_X, XORB_Y) > \text{lt-threshold}, \quad with -1 < Wlt(XORB_X, XORB_Y) < 1.5$$

Algorithm – add long-term dependency

- **Long-term dependency Situations**
  1. LT= {X1-Y1,X1-Y2,X2-Y1,X2-Y2}
  2. LT= {X1-Y1,X1-Y2,X2-Y2}
  3. LT= {X1-Y1,X2-Y2}
  4. LT= {X1-Y1,X2-Y1}
  5. LT= {X1-Y1,X1-Y2}
  6. LT= {X1-Y1}
  7. empty set

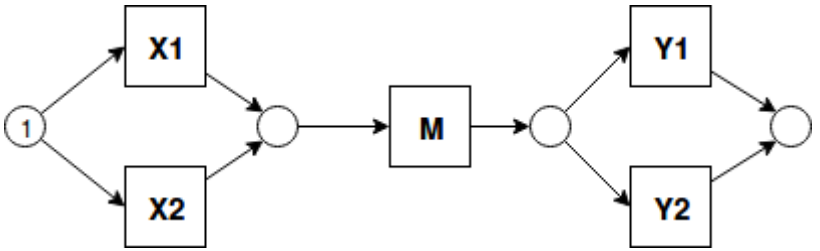




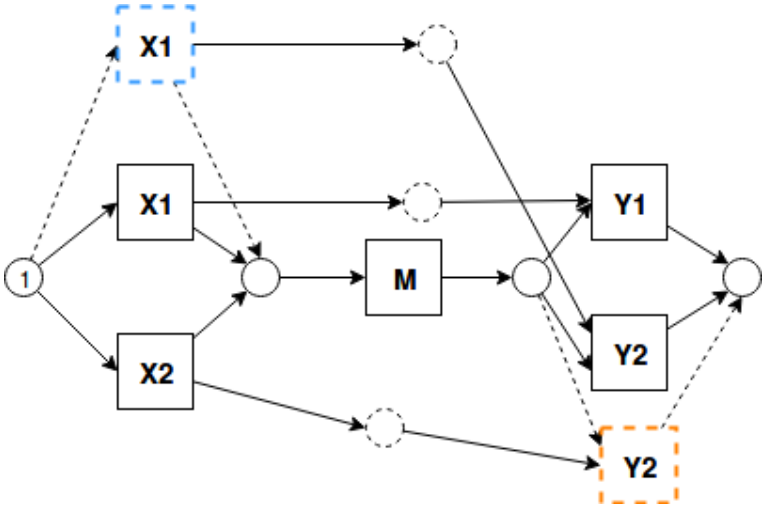
  - Situation 1 is full dependency, it keeps the original model

Algorithm – add long-term dependency

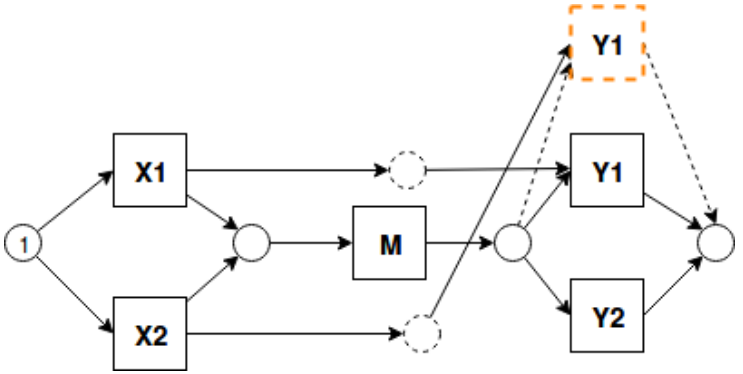
- Expressed On Petri net
  - Duplicate Transition
    - **Keep track of added xor branches for source and target xor branches S and T**



Given LT= {X1-Y1,X1-Y2,X2-Y2}

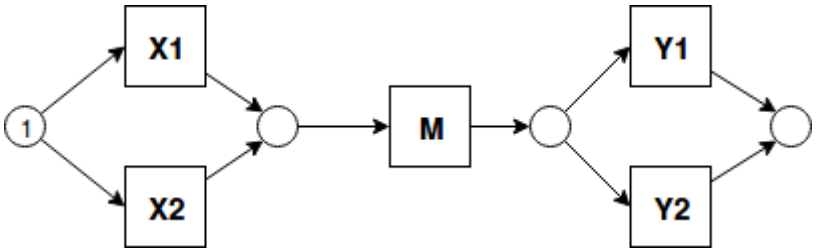


Given LT= {X1-Y1,X2-Y1}

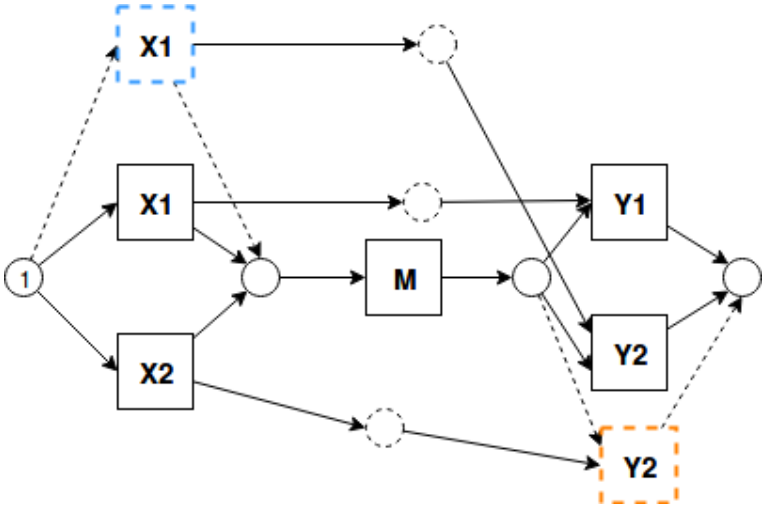


Algorithm – add long-term dependency

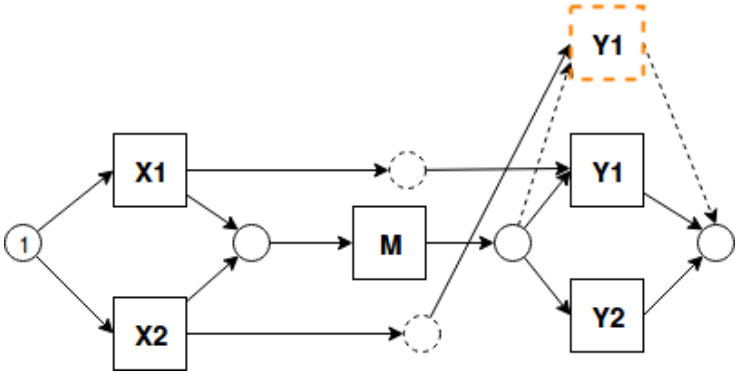
- Expressed On Petri net
  - Duplicate Transition
    - Keep track of added xor branches for source and target xor branches S and T
    - **For every item lt in LT, If source and target are already exist in S and T, then duplicate source and target in xor block**



Given LT= {X1-Y1,X1-Y2,X2-Y2}

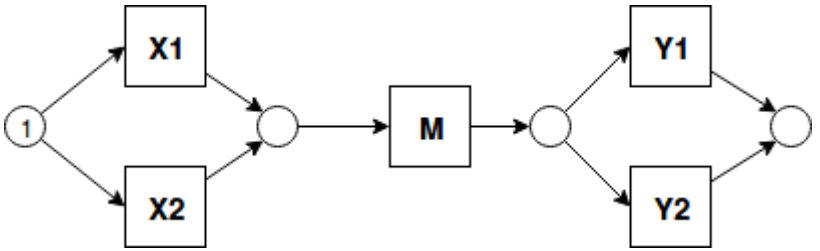


Given LT= {X1-Y1,X2-Y1}

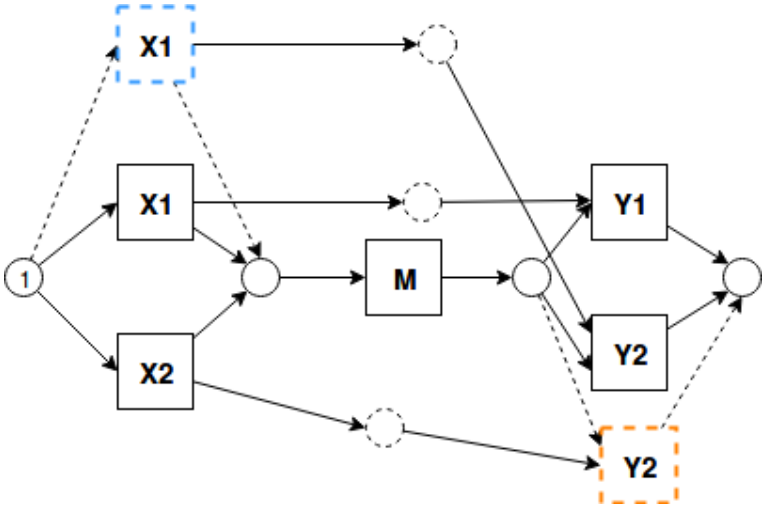


Algorithm – add long-term dependency

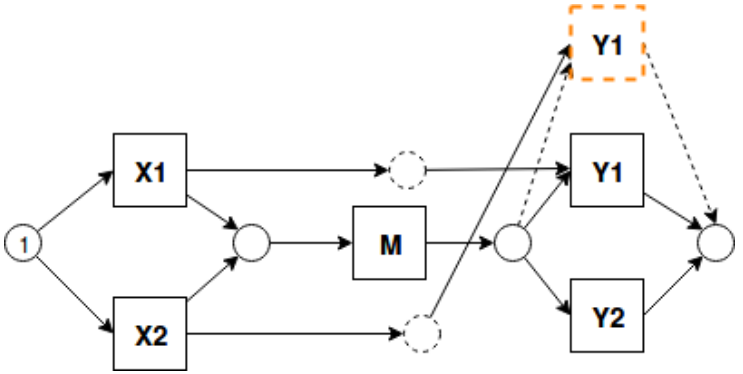
- Expressed On Petri net
  - Duplicate Transition
    - Keep track of added xor branches for source and target xor branches S and T
    - For every item lt in LT, If source and target are already exist in S and T, then duplicate source and target in xor block
    - **connect lt source and target by adding one place and arcs**



Given LT= {X1-Y1,X1-Y2,X2-Y2}

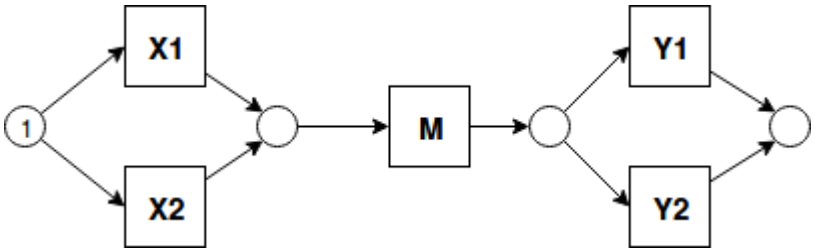


Given LT= {X1-Y1,X2-Y1}

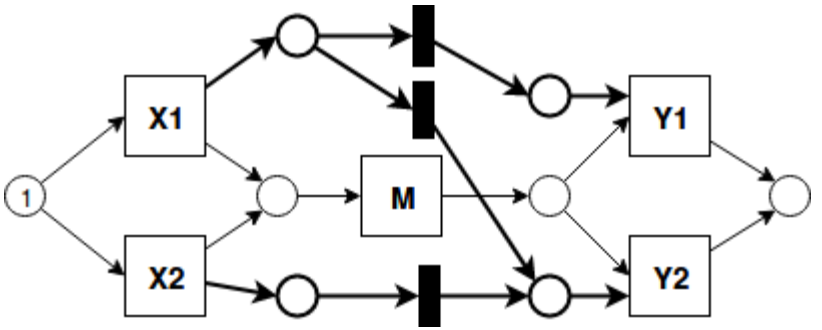


Algorithm – add long-term dependency

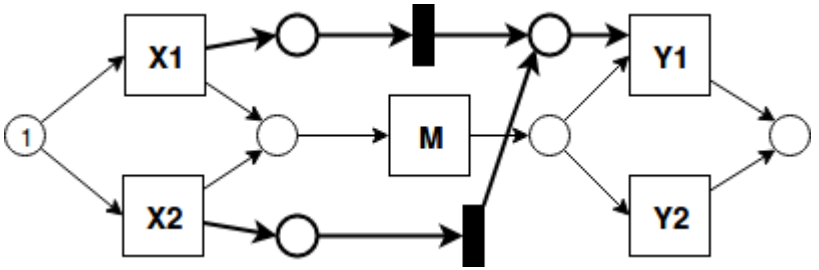
- Expressed On Petri net
  - Add silent transition
    - **Get the source set LT-S and target set LT-T of LT**
    - Create one control place as post-place post for every element in LT-S
    - Create one control place as pre-place before every element in LT-T



Given LT= {X1-Y1,X1-Y2,X2-Y2}

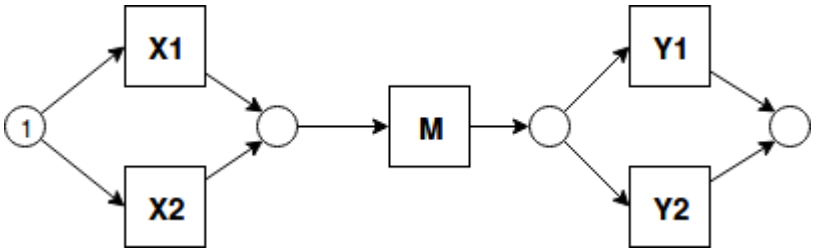


Given LT= {X1-Y1,X2-Y1}

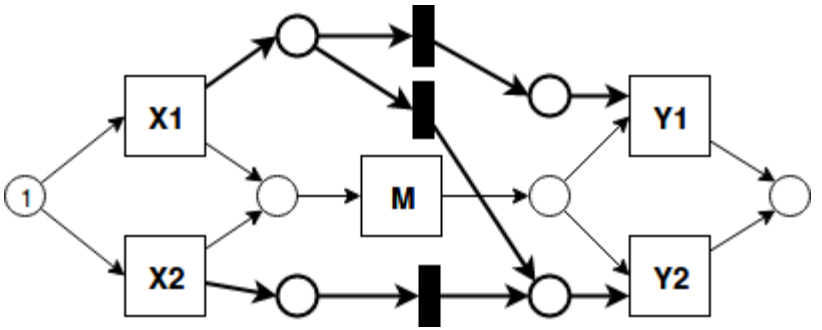


Algorithm – add long-term dependency

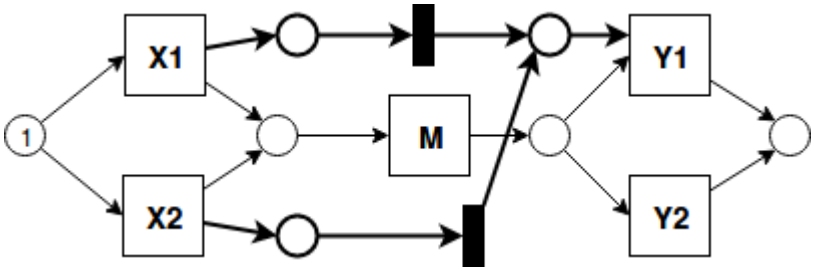
- Expressed On Petri net
  - Add silent transition
    - Get the source set LT-S and target set LT-T of LT
    - **Create one control place as post-place post for every element in LT-S,**
    - **Create one control place as pre-place before every element in LT-T**



Given LT= {X1-Y1,X1-Y2,X2-Y2}

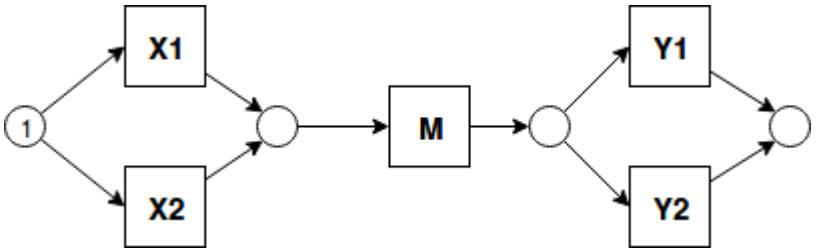


Given LT= {X1-Y1,X2-Y1}

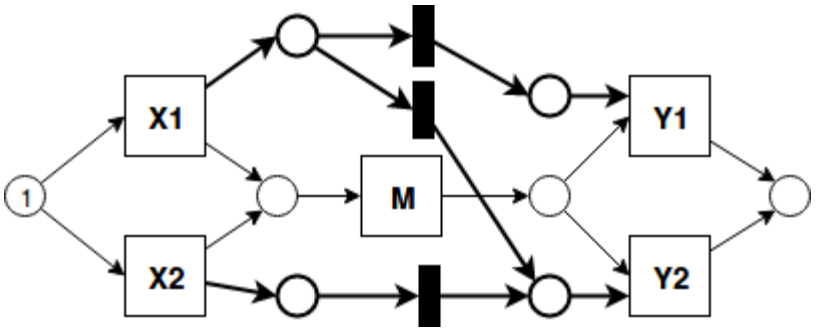


Algorithm – add long-term dependency

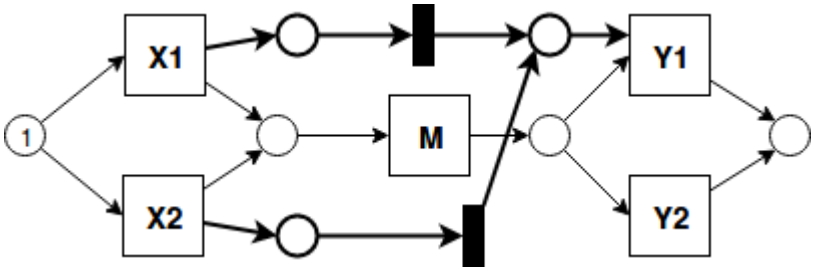
- Expressed On Petri net
  - Add silent transition
    - Get the source set LT-S and target set LT-T of LT
    - Create one control place as post-place post for every element in LT-S,
    - Create one control place as pre-place before every element in LT-T
    - **For every item in LT, create one silent transition to connect the corresponding post-place for lt source and pre-place for lt target**



Given LT= {X1-Y1,X1-Y2,X2-Y2}



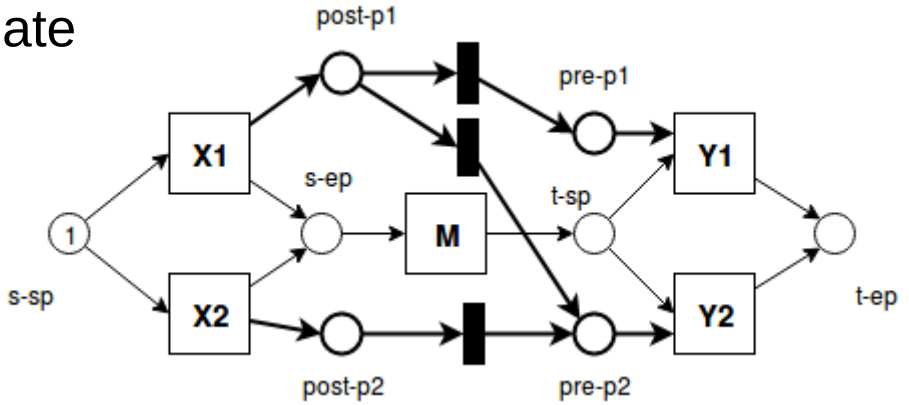
Given LT= {X1-Y1,X2-Y1}

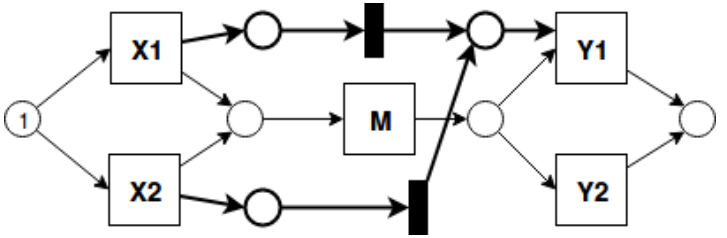


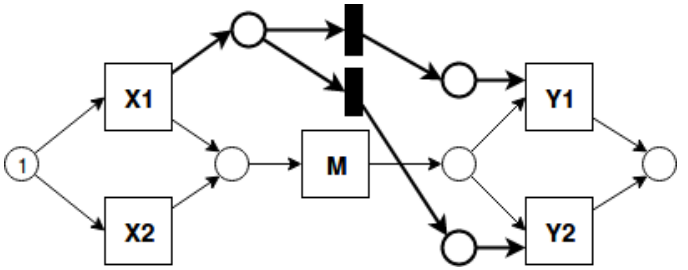
Algorithm – add long-term dependency

- **Soundness**
  - Situation 2 and 3 are sufficient to create sound model

    Proof:
    - **For every xor block pair, M(s-sp)=1, other places p, M(p)=0**



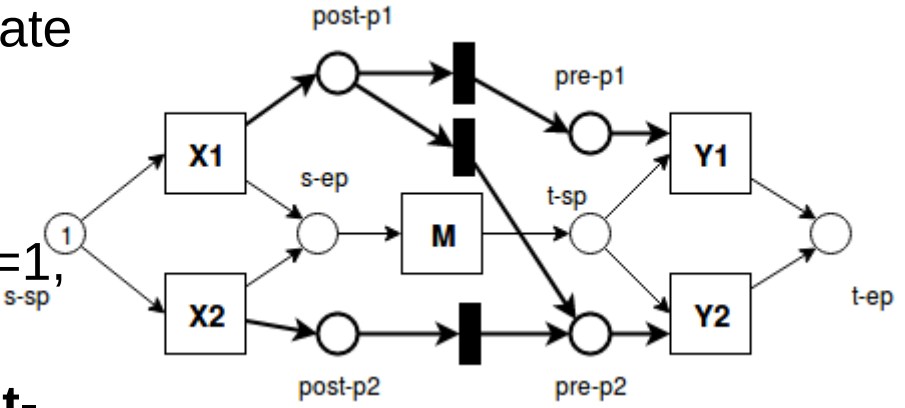


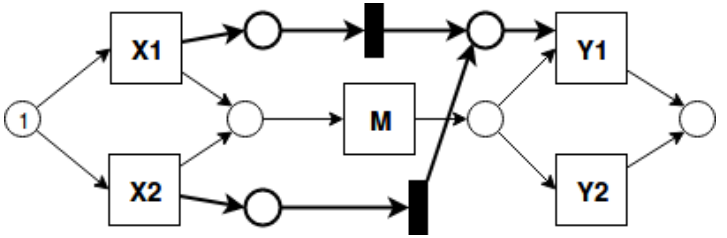


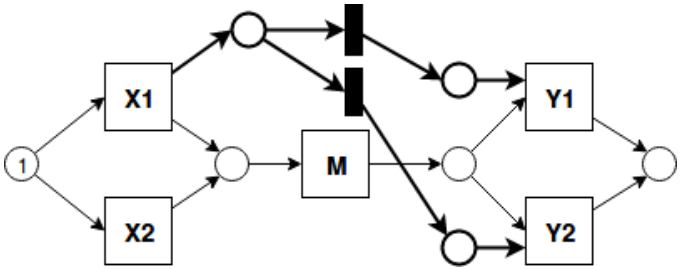
Algorithm – add long-term dependency

- **Soundness**
  - Situation 2 and 3 are sufficient to create sound model

    Proof:
    - For every xor block pair, M(s-sp)=1, other places p, M(p)=0
    - **Xi is chosen, M(s-ep)=1, M(post-pi)=1, other places p, M(p)=0**



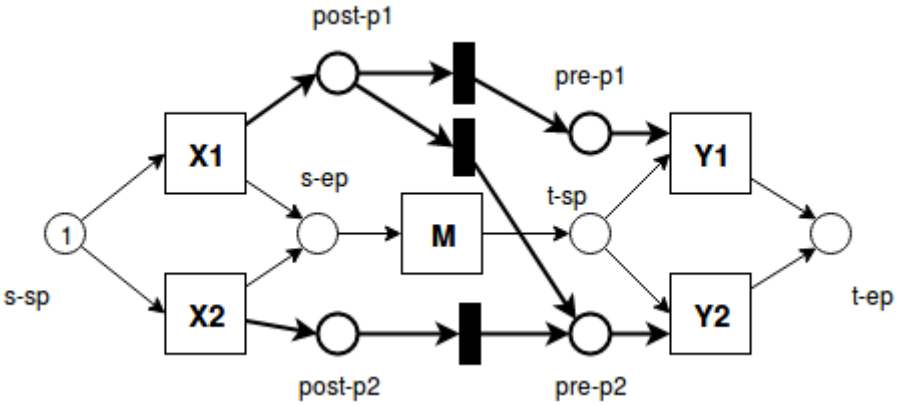


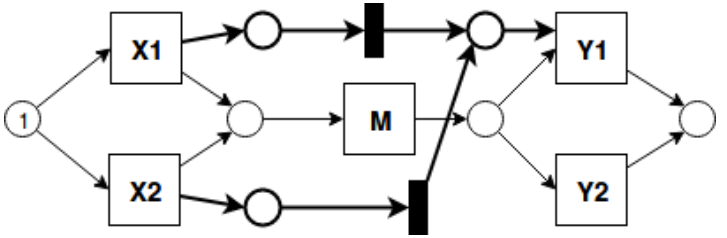


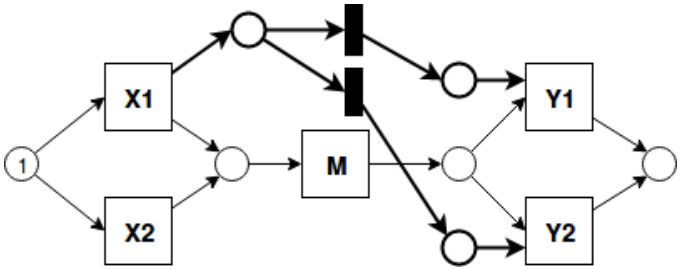
Algorithm – add long-term dependency

- **Soundness**
  - Situation 2 and 3 are sufficient to create sound model

    Proof:
    - For every xor block pair, M(s-sp)=1, other places p, M(p)=0
    - Xi is chosen, M(s-ep)=1, M(post-pi)=1, other places p, M(p)=0
    - **After M, M(t-sp)=1, for any silent transiton st, |InEdge(ts)|=|OutEdge(ts)|=1, token number stays the same until pre places, Sum(M(pre-pi))=Sum(M(post-pi))=1**



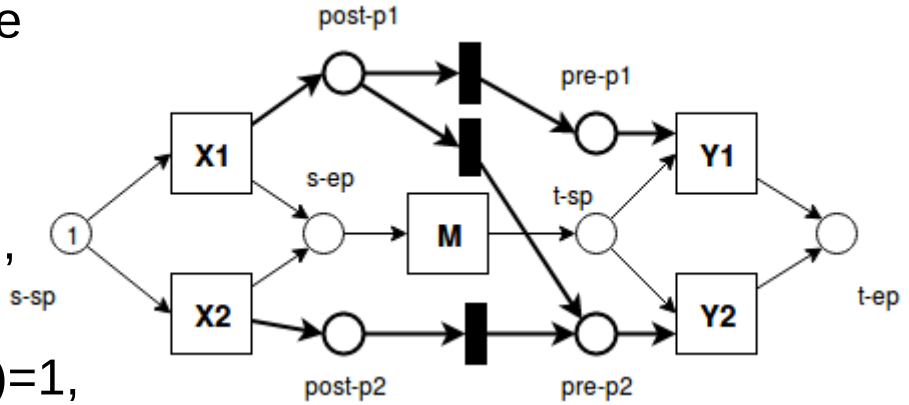


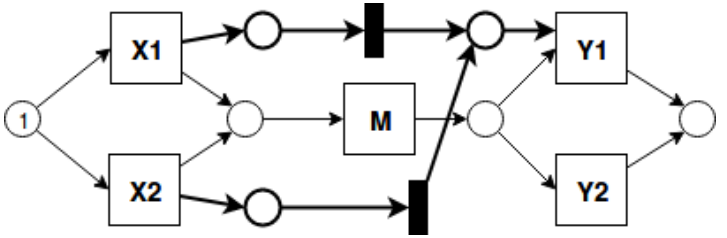


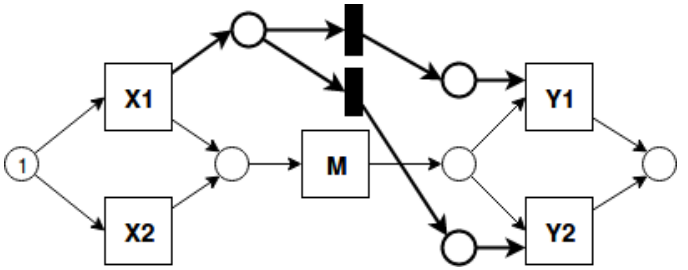
Algorithm – add long-term dependency

- **Soundness**
  - Situation 2 and 3 are sufficient to create sound model

    Proof:
    - For every xor block pair, M(s-sp)=1, other places p, M(p)=0
    - Xi is chosen, M(s-ep)=1, M(post-pi)=1, other places p, M(p)=0
    - After M, M(t-sp)=1, for any silent transiton st, |InEdge(ts)|=|OutEdge(ts)| =1, token number stays the same until pre places, Sum(M(pre-pi))=Sum(M(post-pi))=1
    - **For any Yj, *Yj=2, able to fire, consume two tokens, and produce one token to Place t-ep**

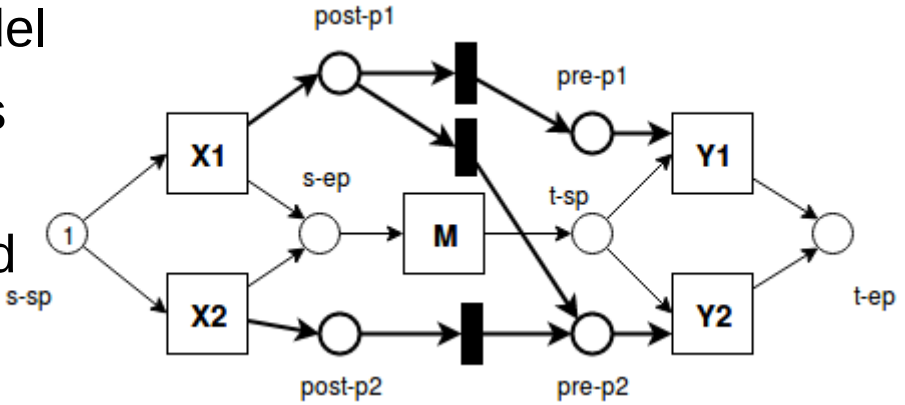


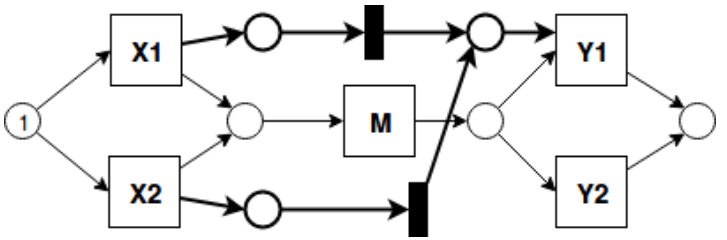


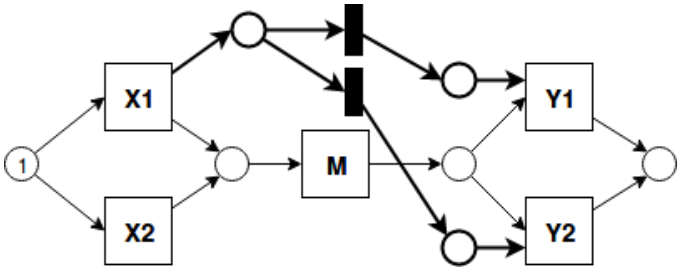


Algorithm – add long-term dependency

- **Soundness**
  - Situation 4-7 can't create sound model
    - Not proper to end: token remains before Y1
    - Dead part: Y1,Y2 can not be fired
  - Ignore those situations







Demo Representation

- **Repair Model**
  - Sequence

    Log={pos:<S1,A,B1,C,T1>,<S2,A,B1,C,T2>,<S2,A,B2,C,T2>

    neg:<S1,A,B2,C,T1>}

  - LT(Xor(S1,S2),Xor(B1,B2))= {S1-B1, S2-B1,S2-B2}

    LT(Xor(B1,B2), Xor(T1,T2))={B1-T1,B1-T2,B2-T2}

    LT(Xor(S1,S2),Xor(T1,T2))= {S1-T1(??), S2-T2}

Evaluation

- **Confusion Matrix**

	Allowed behavior	Not allowed behavior
Positive Traces	AP(High)	NP(Low)
Negative Traces	AN(Low)	NN(High)

Precision = Σ AP / Σ AP+ NP

Accuracy = Σ AP + NN / Σ AP + AN + NP + NN

Recall = Σ AP / Σ AP + AN