Handling Dynamic Changes in Petri Net Models of Workflow Processes

Third Annual Progress Seminar
By

Ahana Pradhan
(113050039)

Working under the guidance of
Prof. Rushikesh K. Joshi

Department of Computer Science & Engineering
Indian Institute of Technology Bombay
Powai, Mumbai-400076, India

Dynamic Migration of Workflows

Dynamic instance migration needs to be facilitated for workflows in order to reflect real-world changes in automated processes.

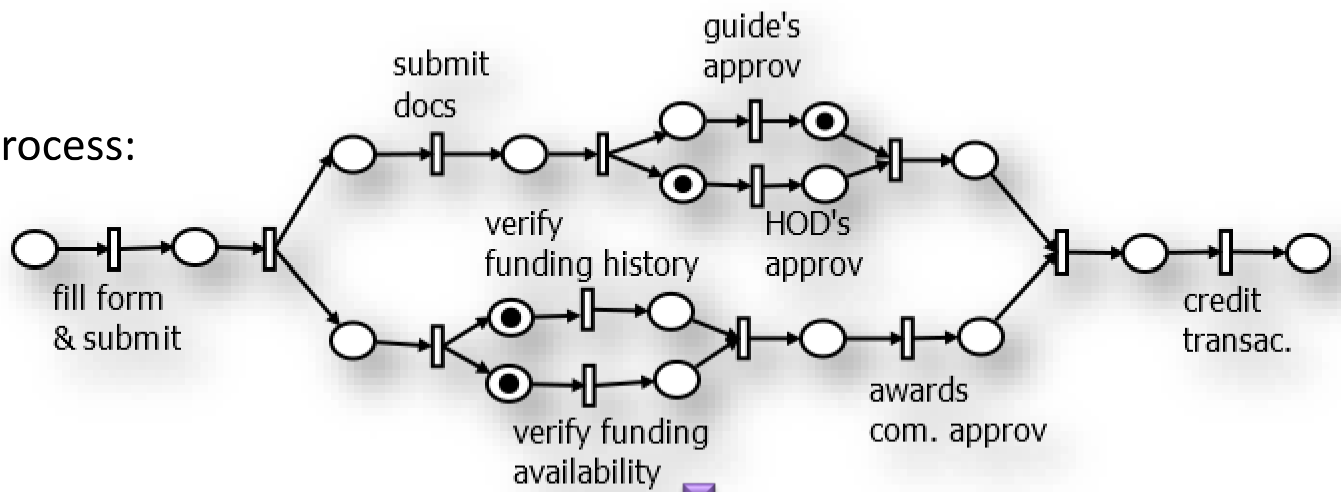
Consistency model:

Equivalence mapping from current state of old workflow to the migrating state of the new workflow.

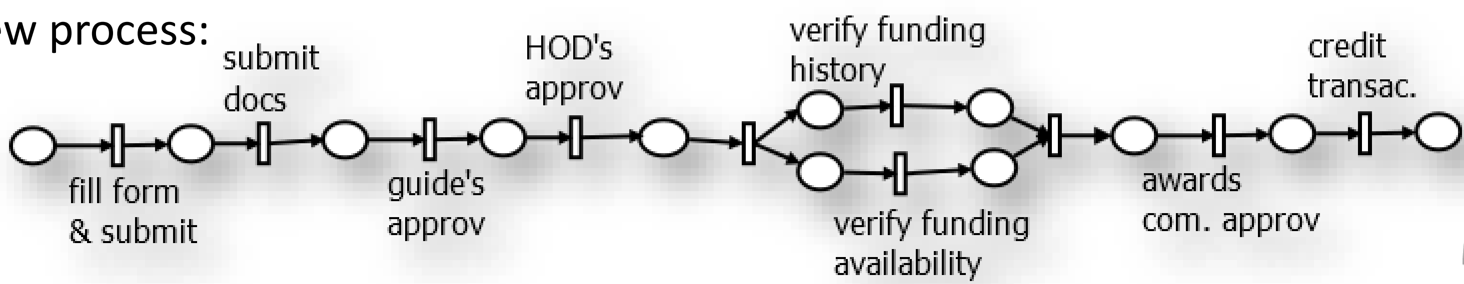
Dynamic Evolution of Workflows

Reimbursement Workflow in an Academic Institute

Old process:



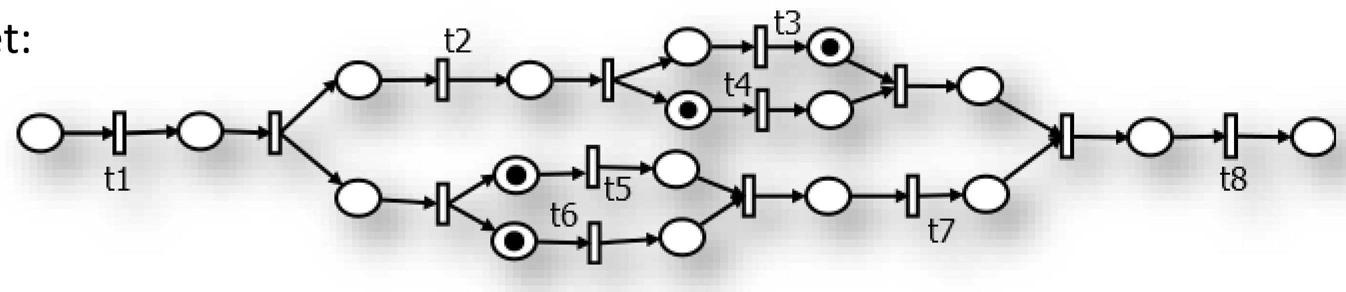
New process:



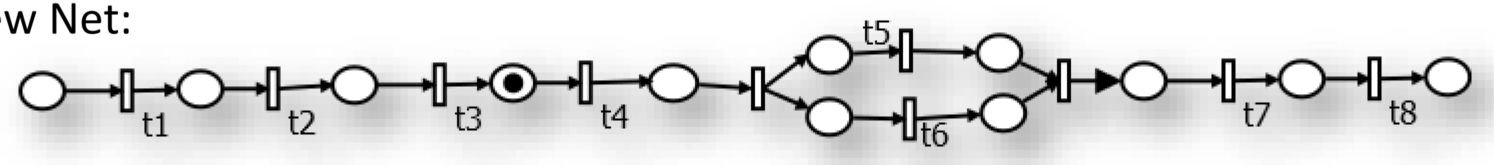
Token Transportation

Given a marking in the old net (running instance), goal is to obtain a marking in the new net (migrated instance)

Old Net:

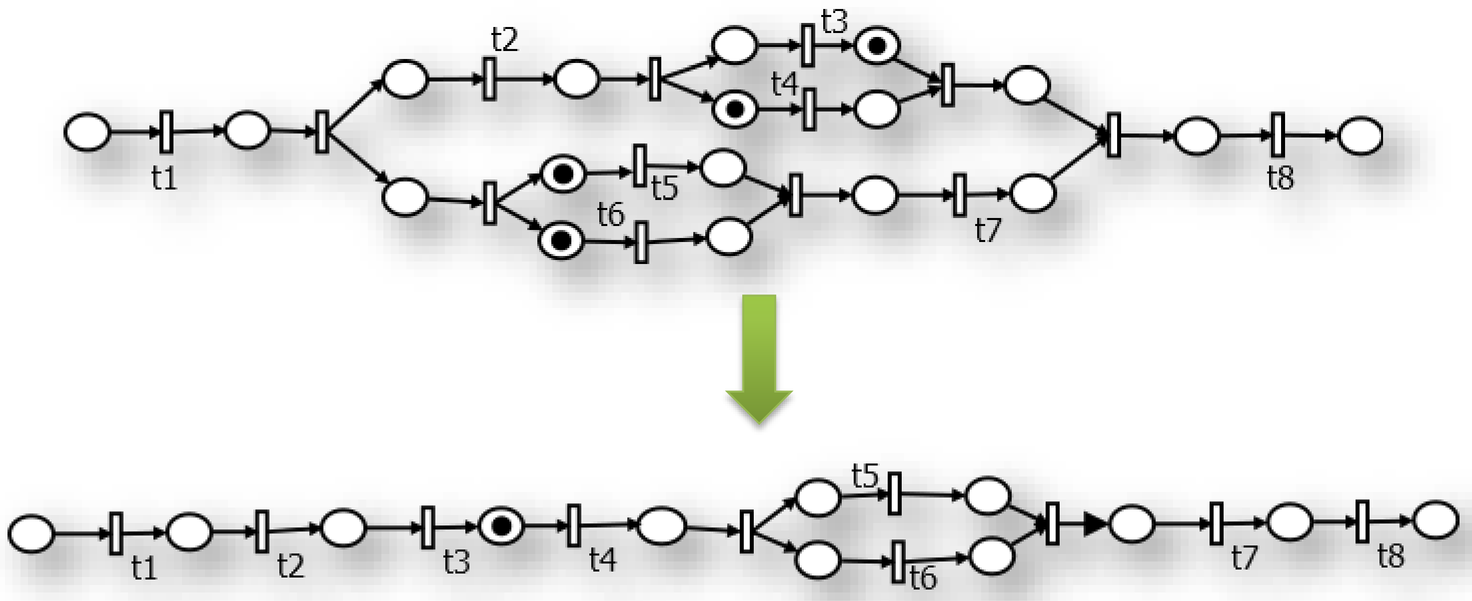


New Net:



History-based Consistency

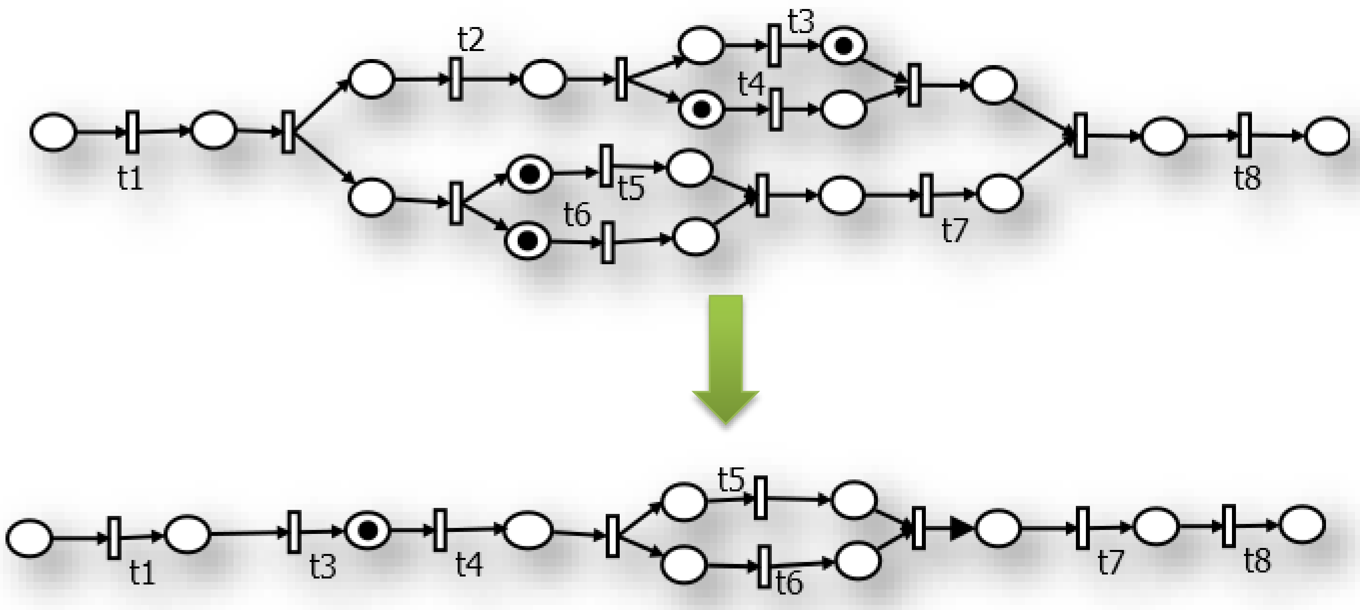
History equivalence (Compliance) [Ellis et al. COCS'95, Rinderle et al. ER'08]



History: t1, t2, t3

History-based Consistency

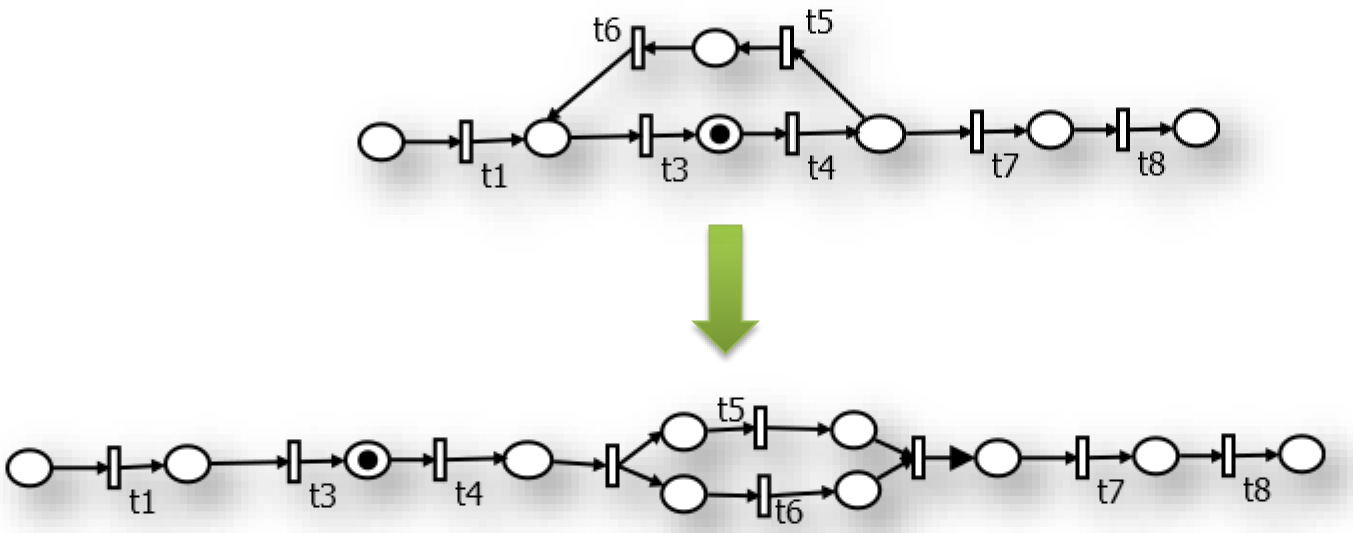
Delete-purged Compliance [Rinderle et al. ER'08]



Delete-purged History: t1, t3

History-based Consistency

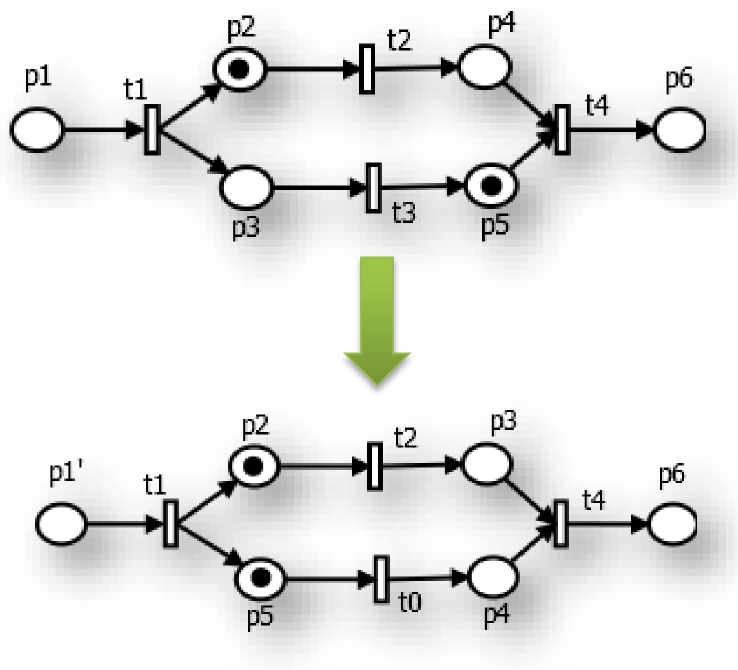
Loop-purged Compliance [Rinderle et al. ER'08, Sun et al., IST'09]



Common Reduced History: t1, t3

Marking-based Consistency

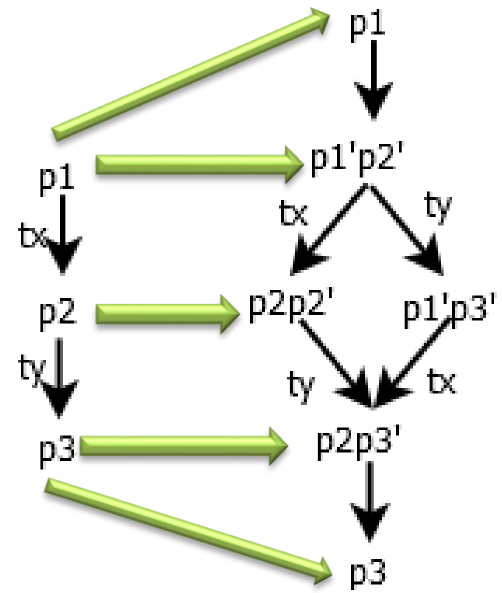
Valid transfer [Van der Aalst, ISF 01]



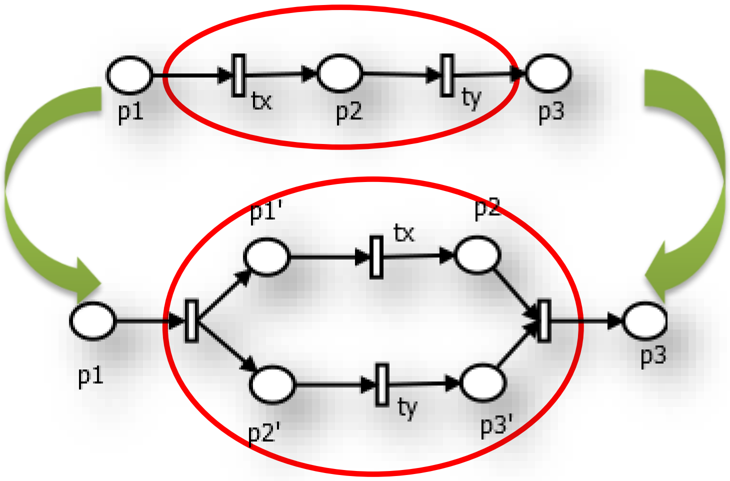
Marking { p2, p5 }

Notable Existing Solution Approaches

State Space [Agostini et al., CSCW'00]



Change region [Van der Aalst, ISF'01; Sun et al., IST'09]



Outline

1. Algorithm for Trace equivalence token transportation
2. Lookahead Trace based consistency models
3. Conclusion
4. Future works

Yo-Yo Algorithm

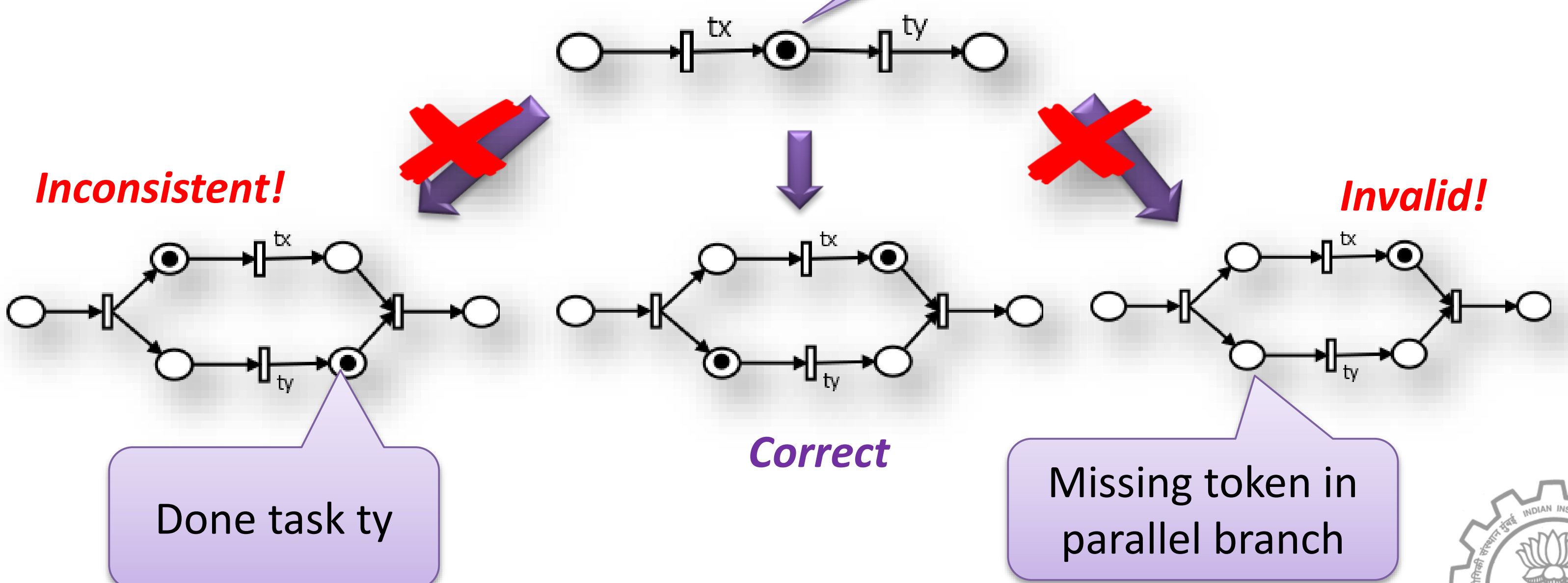
Token Transportation: Correctness

Consistency

preservation of history (done tasks in old ↔ done tasks in new)

Validity

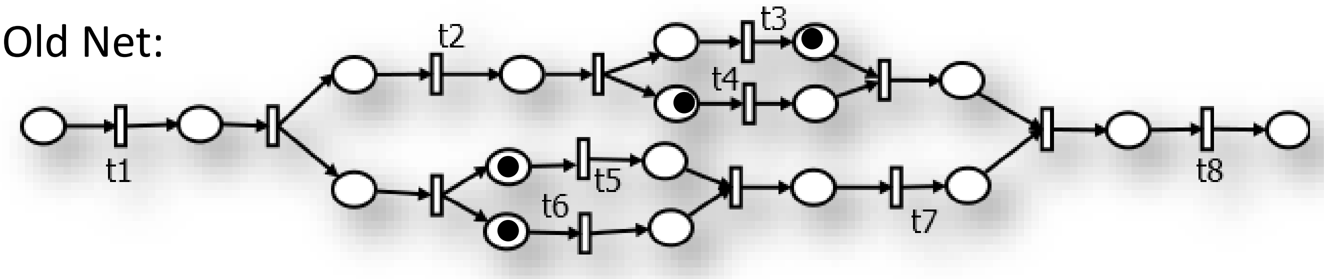
reachability of marking in the new net



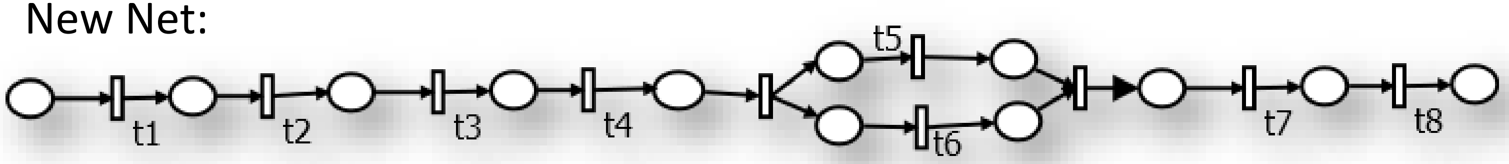
Yo-Yo Approach

Token transportation by: Folding, transport, Unfolding

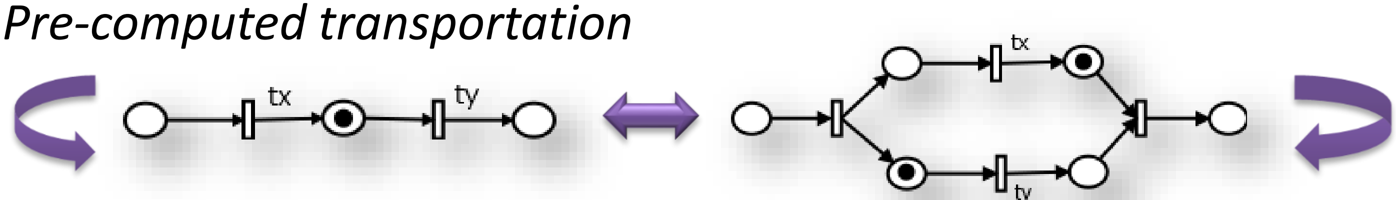
Old Net:



New Net:

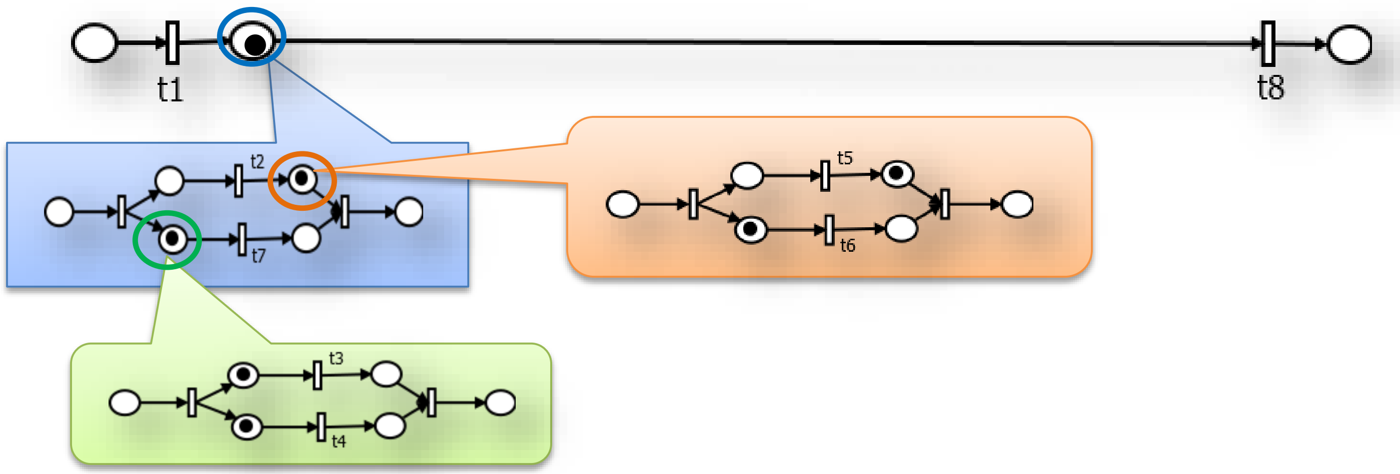


Pre-computed transportation

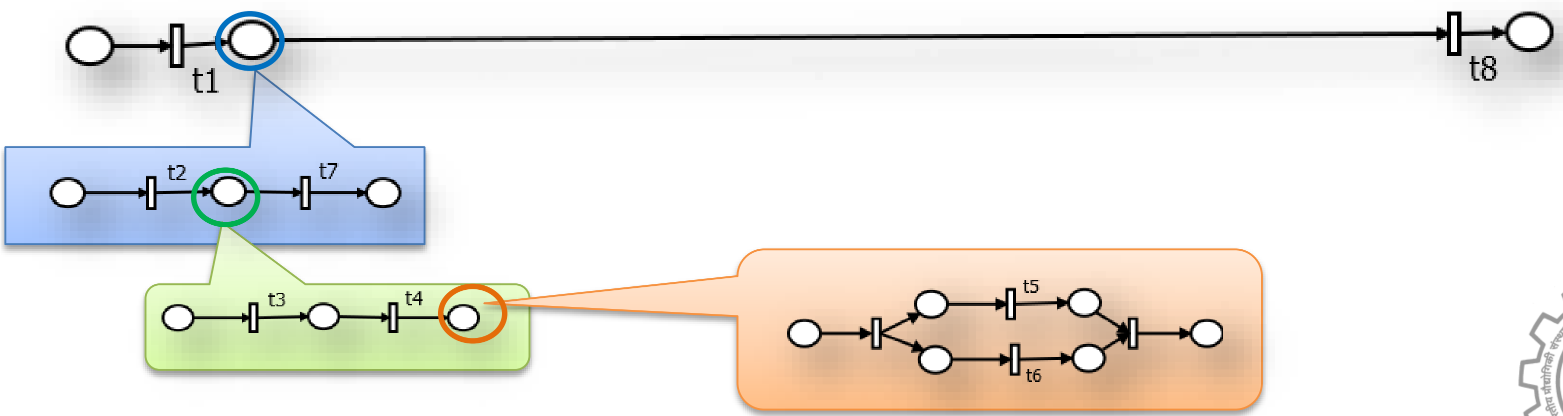


Yo-Yo Approach: Folding

Old Net:

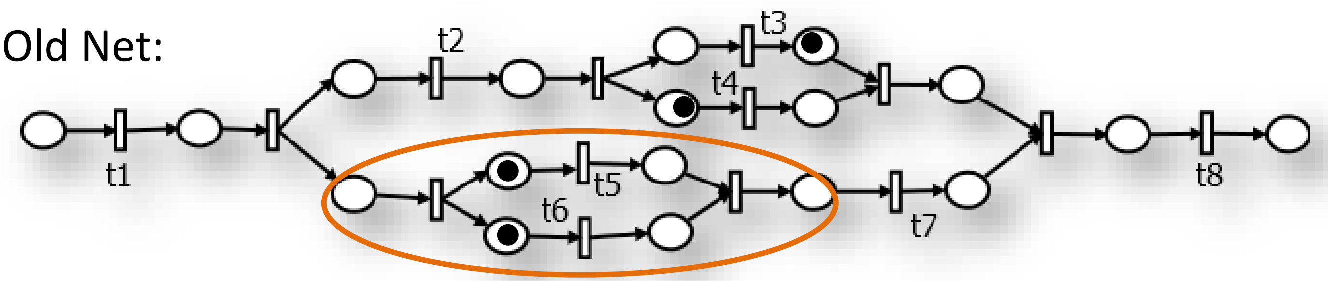


New Net:

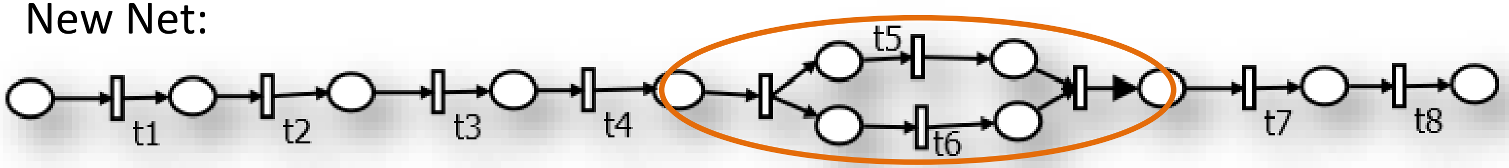


Folding: Original Nets

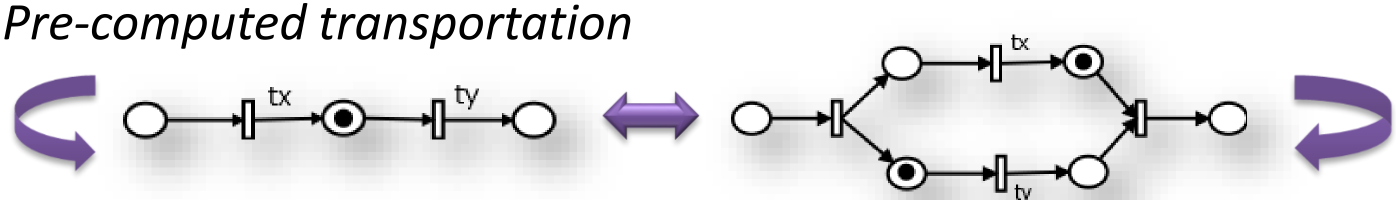
Old Net:



New Net:

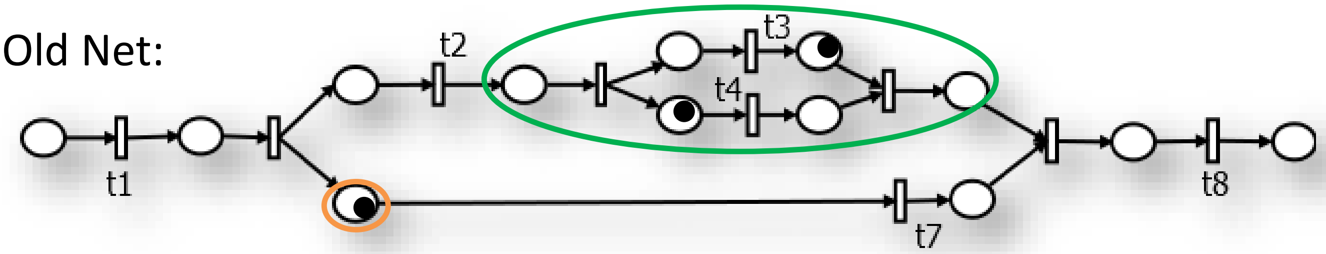


Pre-computed transportation

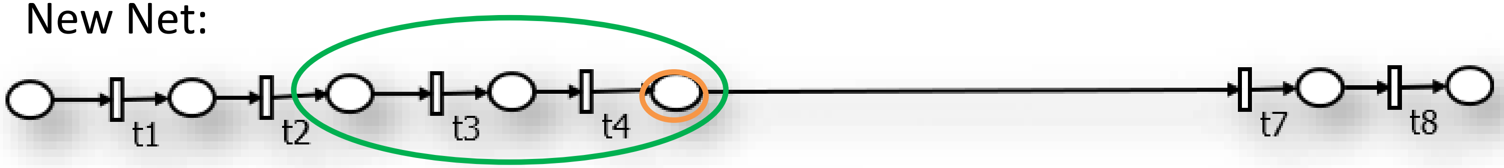


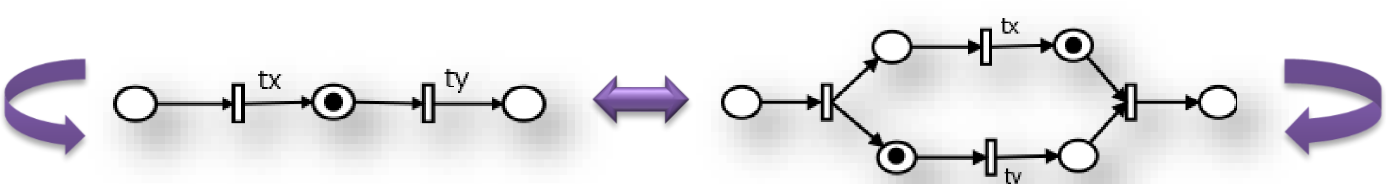
Folding: Step 1

Old Net:



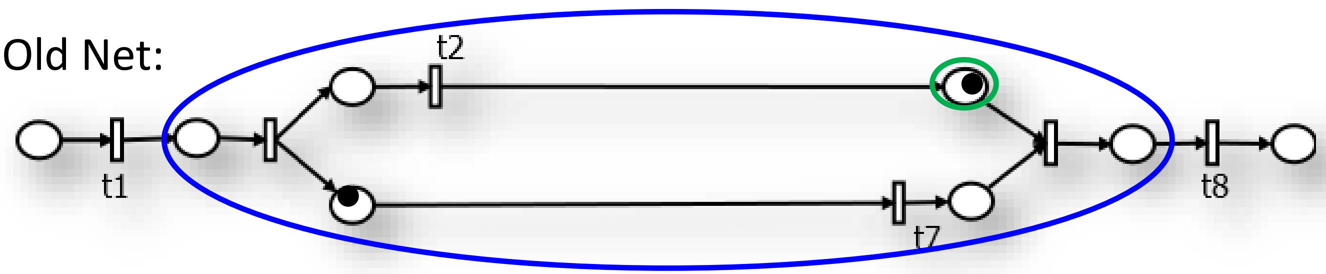
New Net:



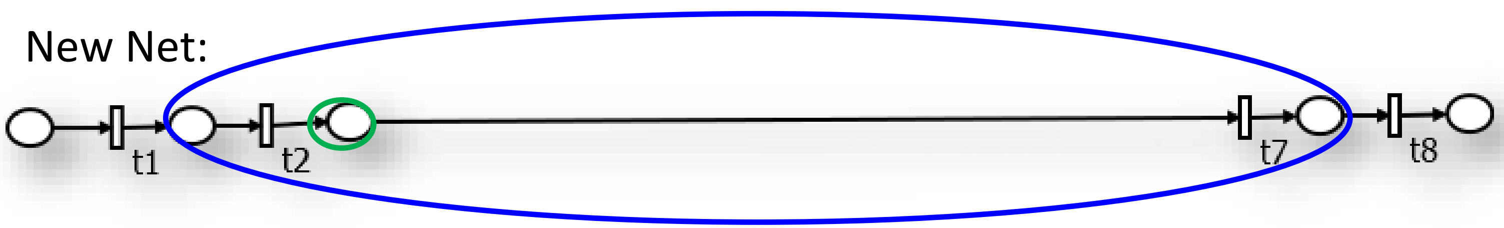


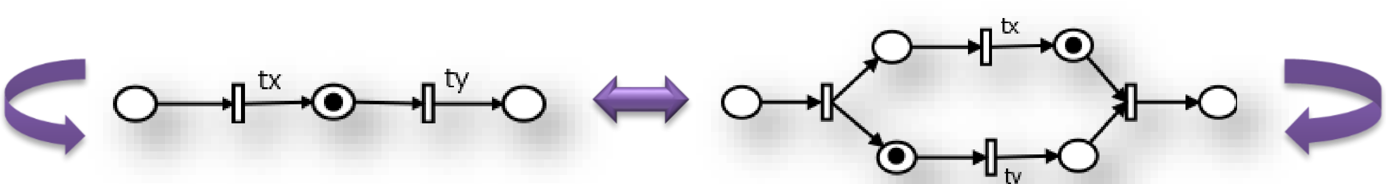
Folding: Step 2

Old Net:



New Net:

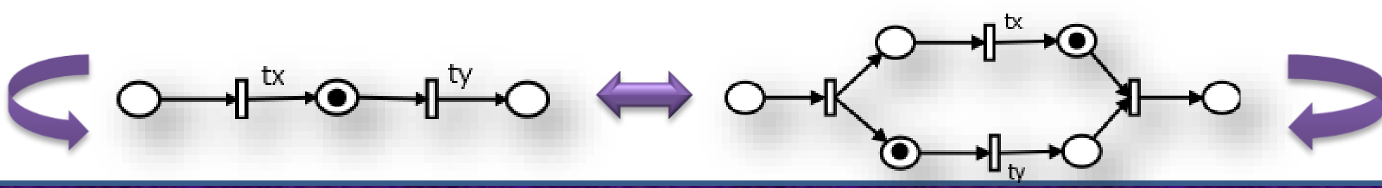




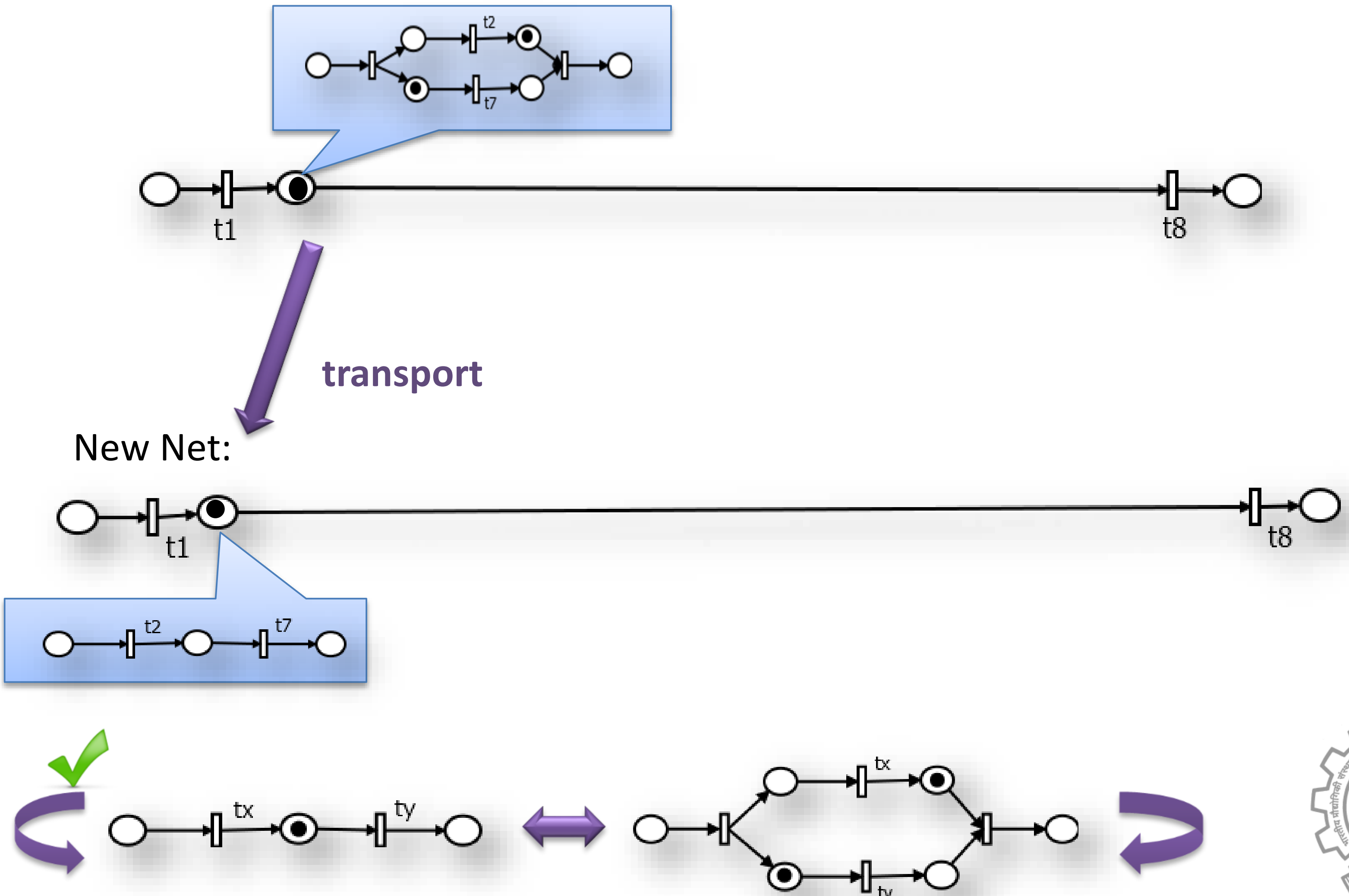
Folding: Step 3

Old Net:

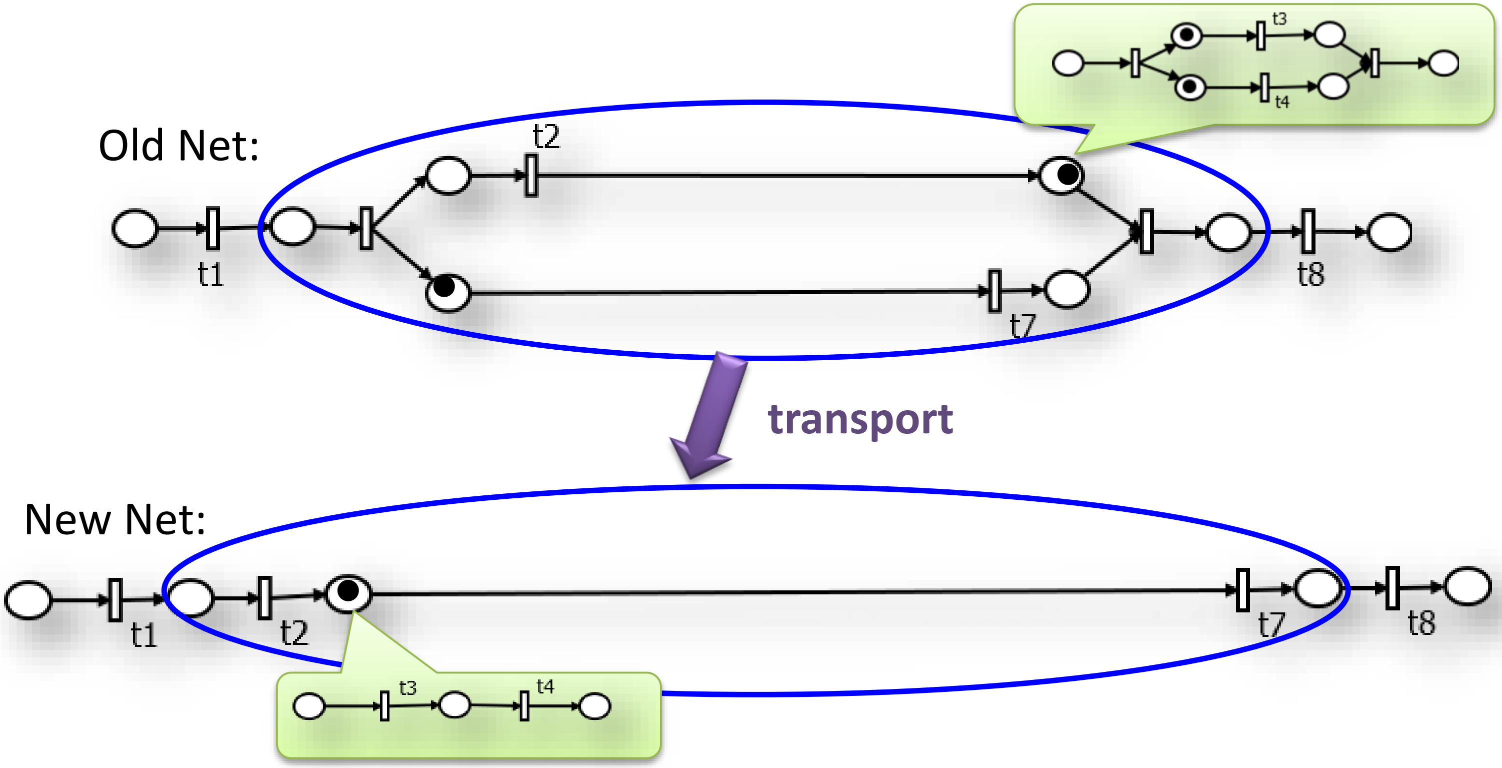
New Net:

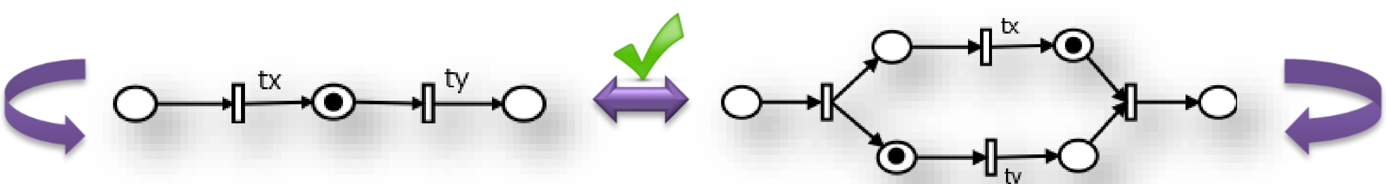


Transport: Step 1

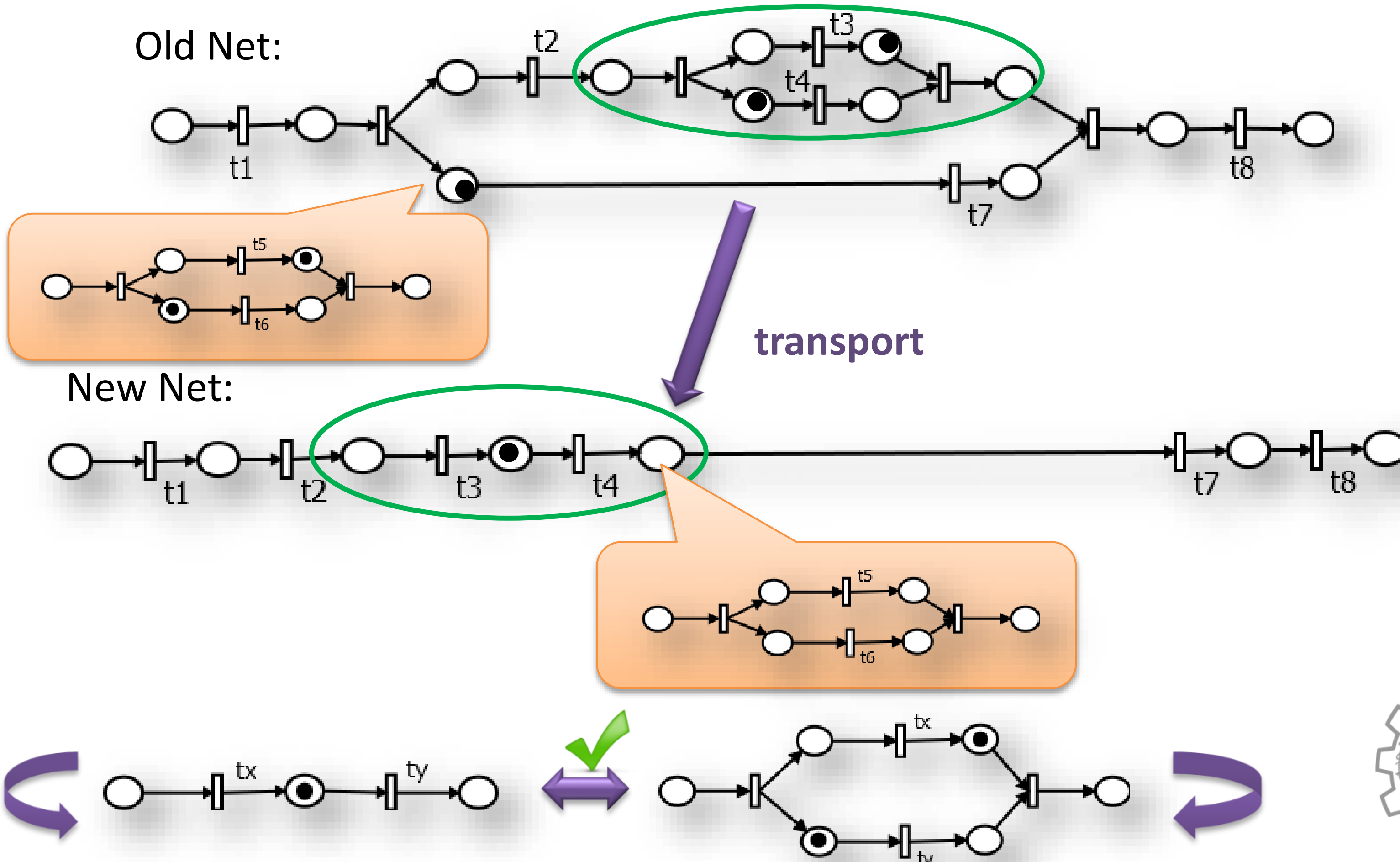


Unfolding and Transport: Step 2



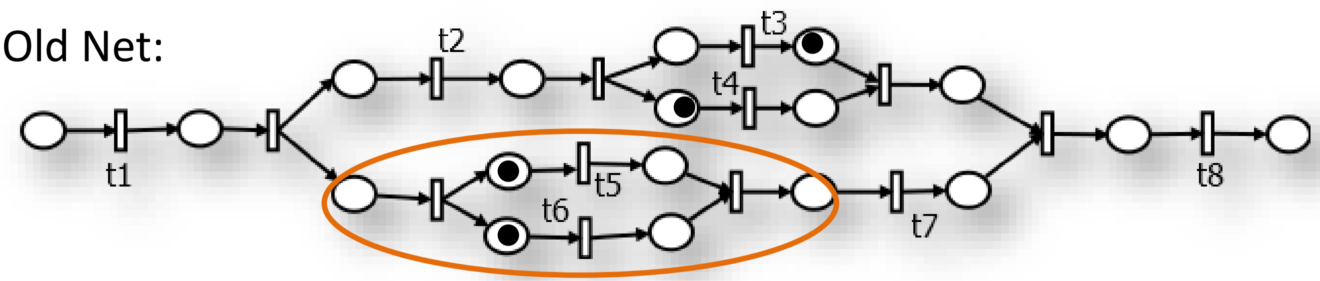


Unfolding and Transport: Step 3



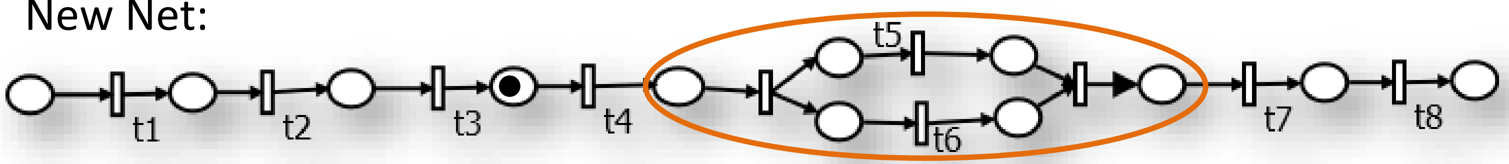
Unfolding: Step 4

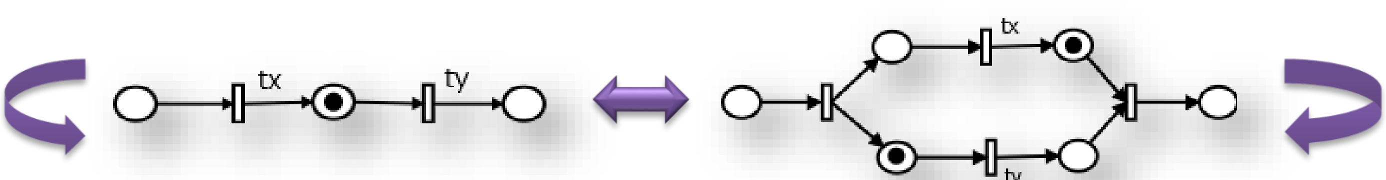
Old Net:



New Net:

No transport required

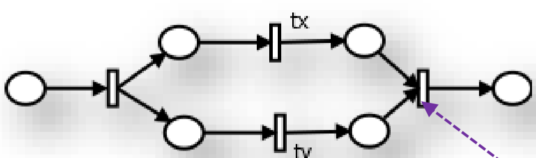
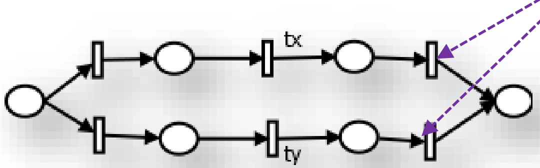




Yo-Yo Approach: ingredients

Transportation between which two patterns	*Peer patterns*
When such hand-in-hand folding of nets are possible	*Yo-Yo compatibility*
Which pattern to fold when	*Folding order*, obtained from **Derivation Trees**
What all pre-computed transportations cover the scope	**Token transportation Catalog**

Input nets

Pattern	Specification	Net Model
SEQ:	tx ty	
AND:	(tx) (ty)	
XOR:	[tx] [ty]	

silent transitions model gateway logic

Input nets

Composition of primitive patterns: sequence or nesting

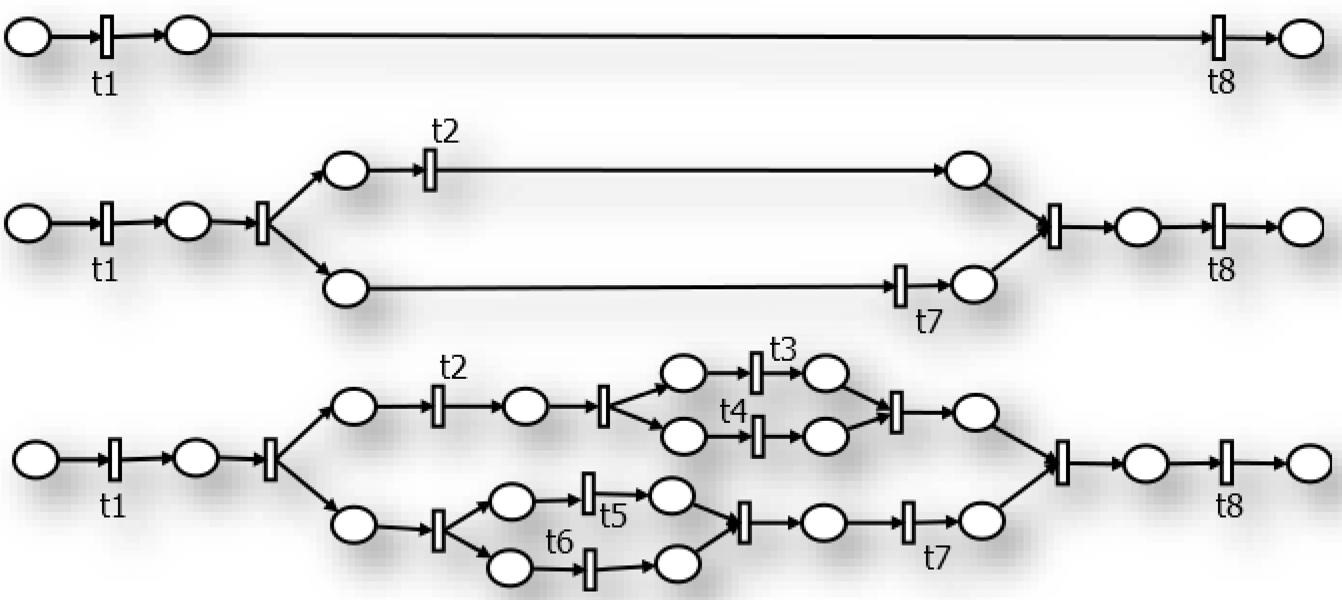
Start → SEQ
SEQ → SEQ **t** SEQ **t** SEQ | SEQ AND SEQ | SEQ XOR SEQ | e
AND → (SEQ **t** SEQ) (SEQ **t** SEQ)
XOR → [SEQ **t** SEQ] [SEQ **t** SEQ]

Example derivation

Start → SEQ → SEQ t1 SEQ t8 SEQ → t1 AND t8
→t1 (SEQ t2 SEQ) (SEQ t7 SEQ) t8
→t1 (t2 SEQ AND SEQ) (SEQ AND SEQ t7) t8
→t1 (t2 (SEQ t3 SEQ) (SEQ t4 SEQ)) ((SEQ t5 SEQ) (SEQ t6 SEQ) t7) t8
→ t1 (t2 (t3) (t4)) ((t5) (t6) t7) t8

Input nets

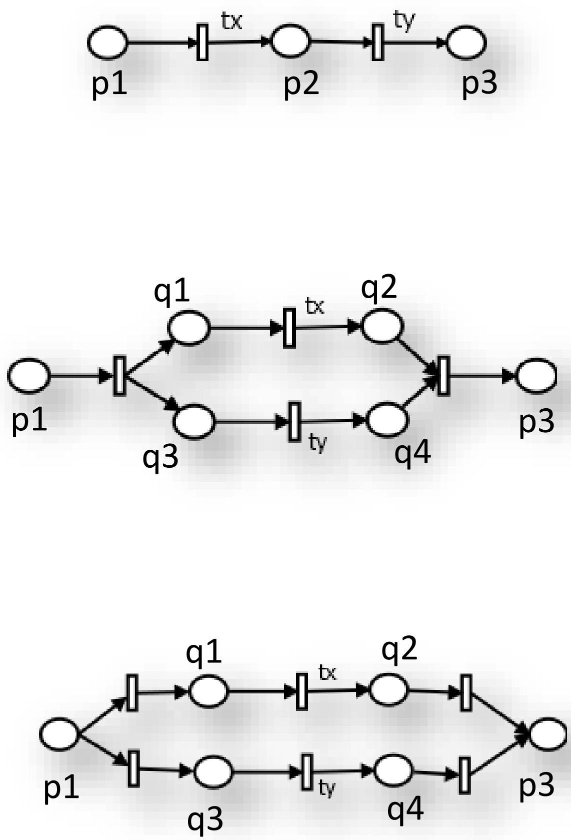
Start → SEQ → SEQ t1 SEQ t8 SEQ → t1 AND t8
→t1 (SEQ t2 SEQ) (SEQ t7 SEQ) t8
→t1 (t2 SEQ AND SEQ) (SEQ AND SEQ t7) t8
→t1 (t2 (SEQ t3 SEQ) (SEQ t4 SEQ)) ((SEQ t5 SEQ) (SEQ t6 SEQ) t7) t8
→ t1 (t2 (t3) (t4)) ((t5) (t6) t7) t8



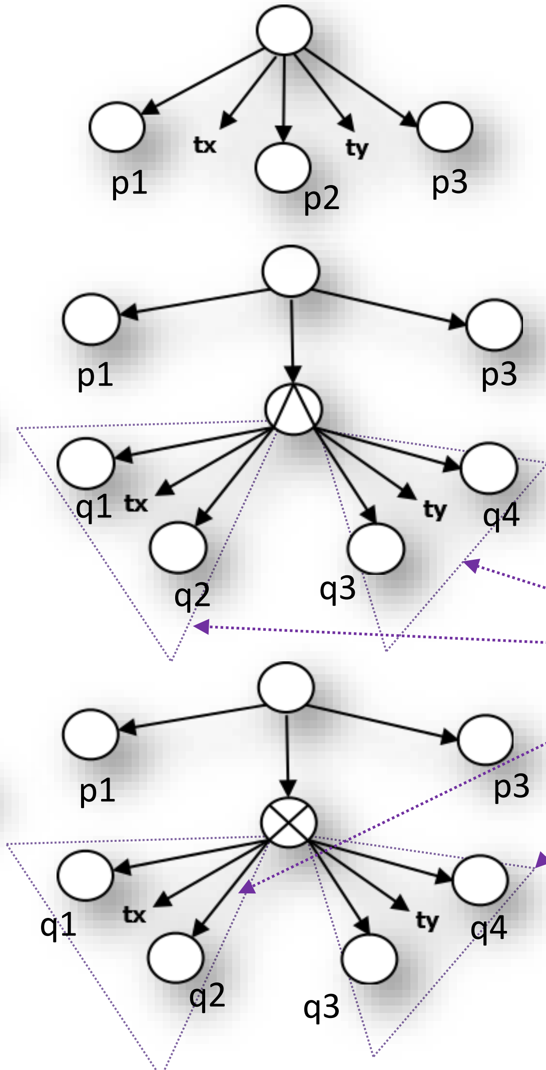
Folding steps follow such order of derivation..

Derivation Trees

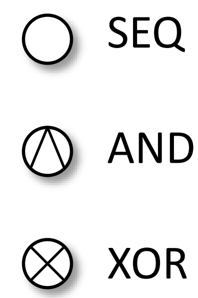
Primitive Block



Derivation Tree

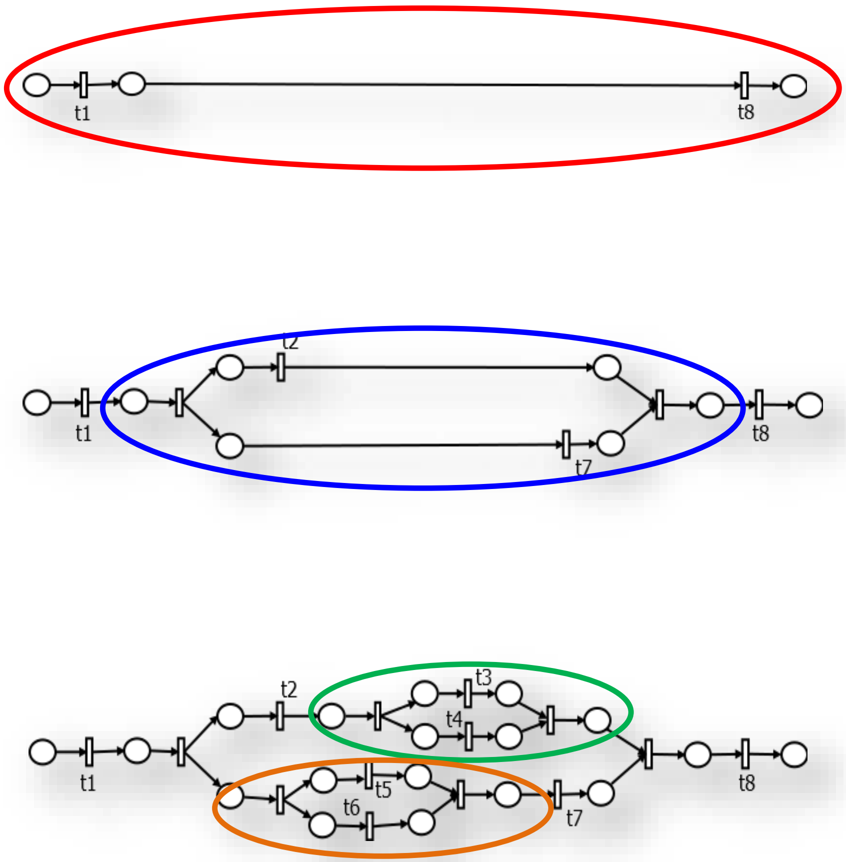


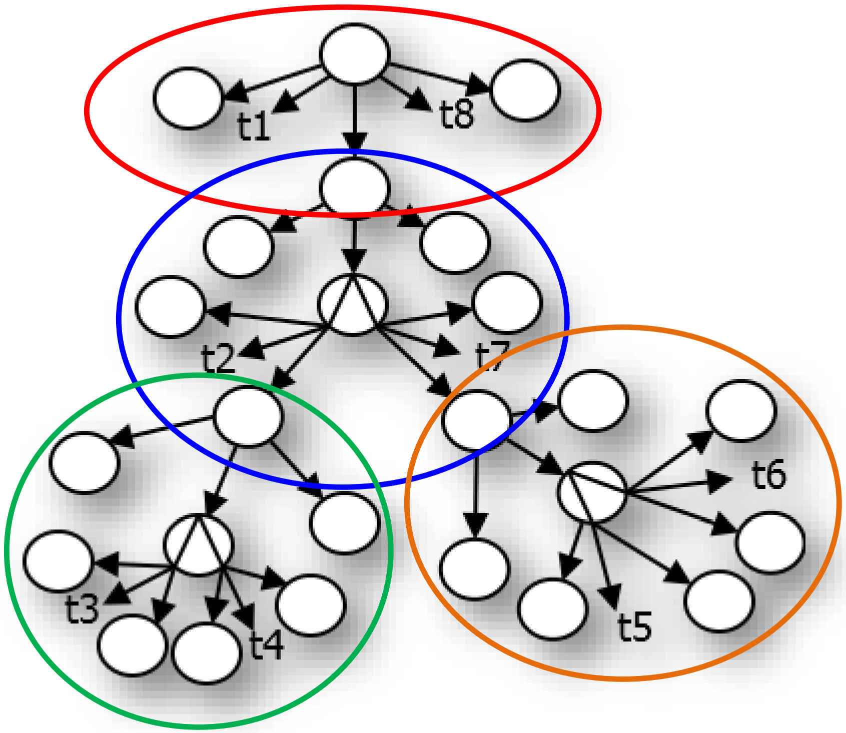
Grammar Non-terminals



Triplets:
Left-right positioning w.r.t. parent does not matter

Derivation Trees



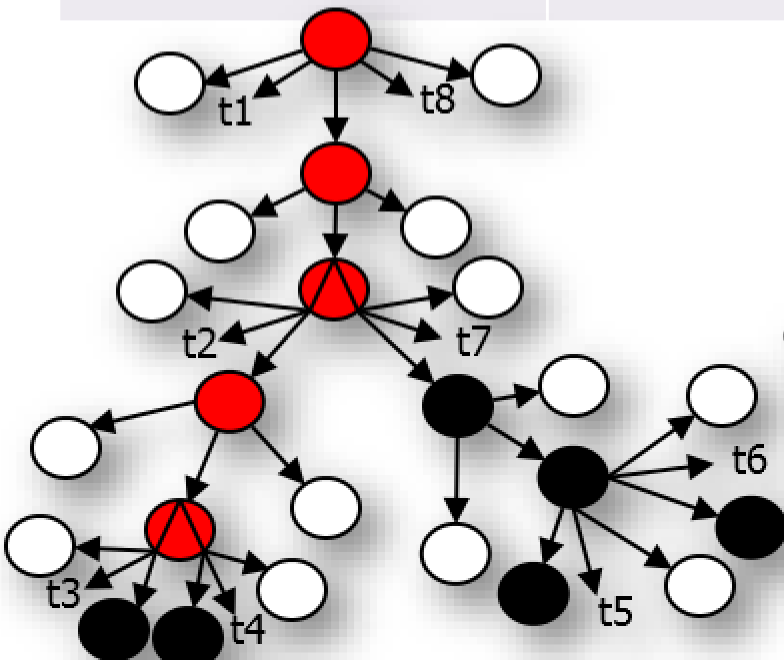


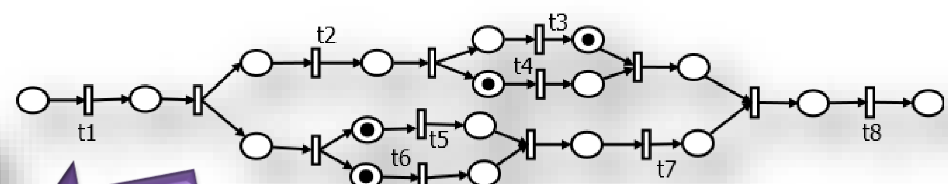
Colored Derivation Trees

Node Type		Description
Leaf/Non-leaf	○	Unmarked folded/unfolded place
Leaf	●	marked place in net
Non-leaf	●	abstraction of null-executed subnet
Non-leaf	🔴	abstraction of subnets where at least one labeled transition has been fired

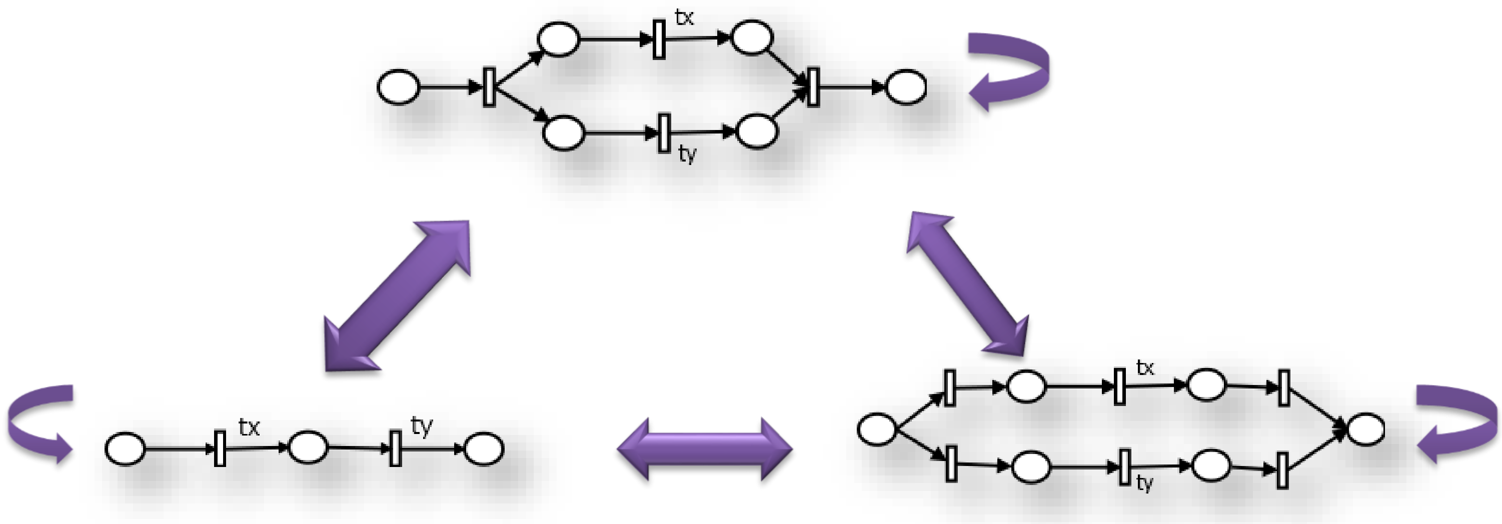
Red node:
Color parent red

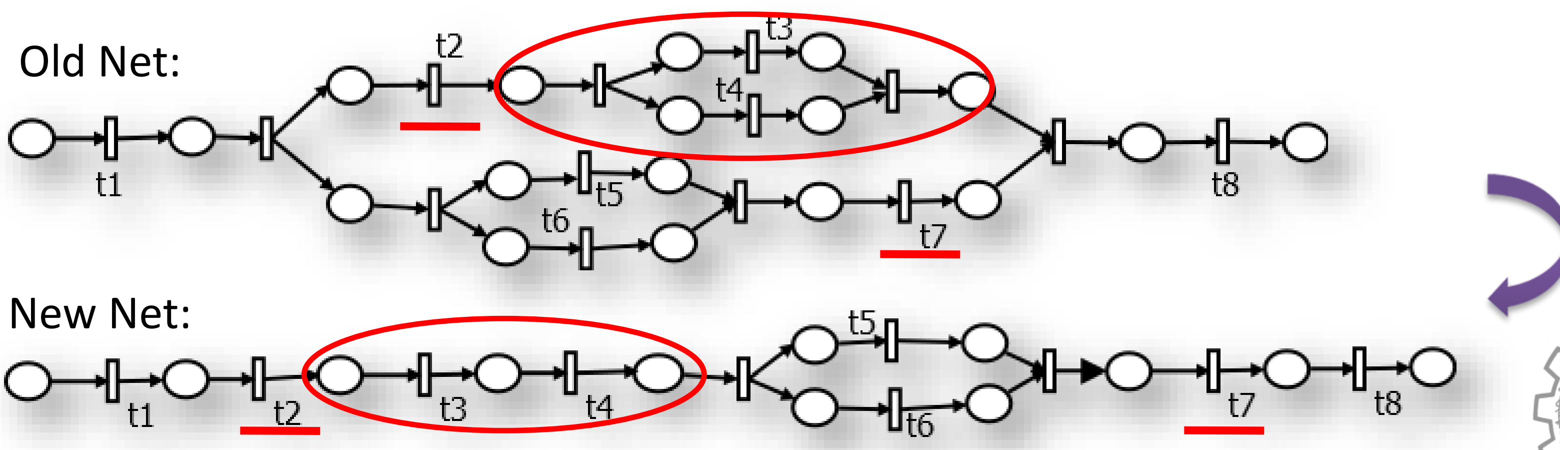
Black node:
Check if any transition Sibling has color at right, If yes, color parent red; Else color parent black



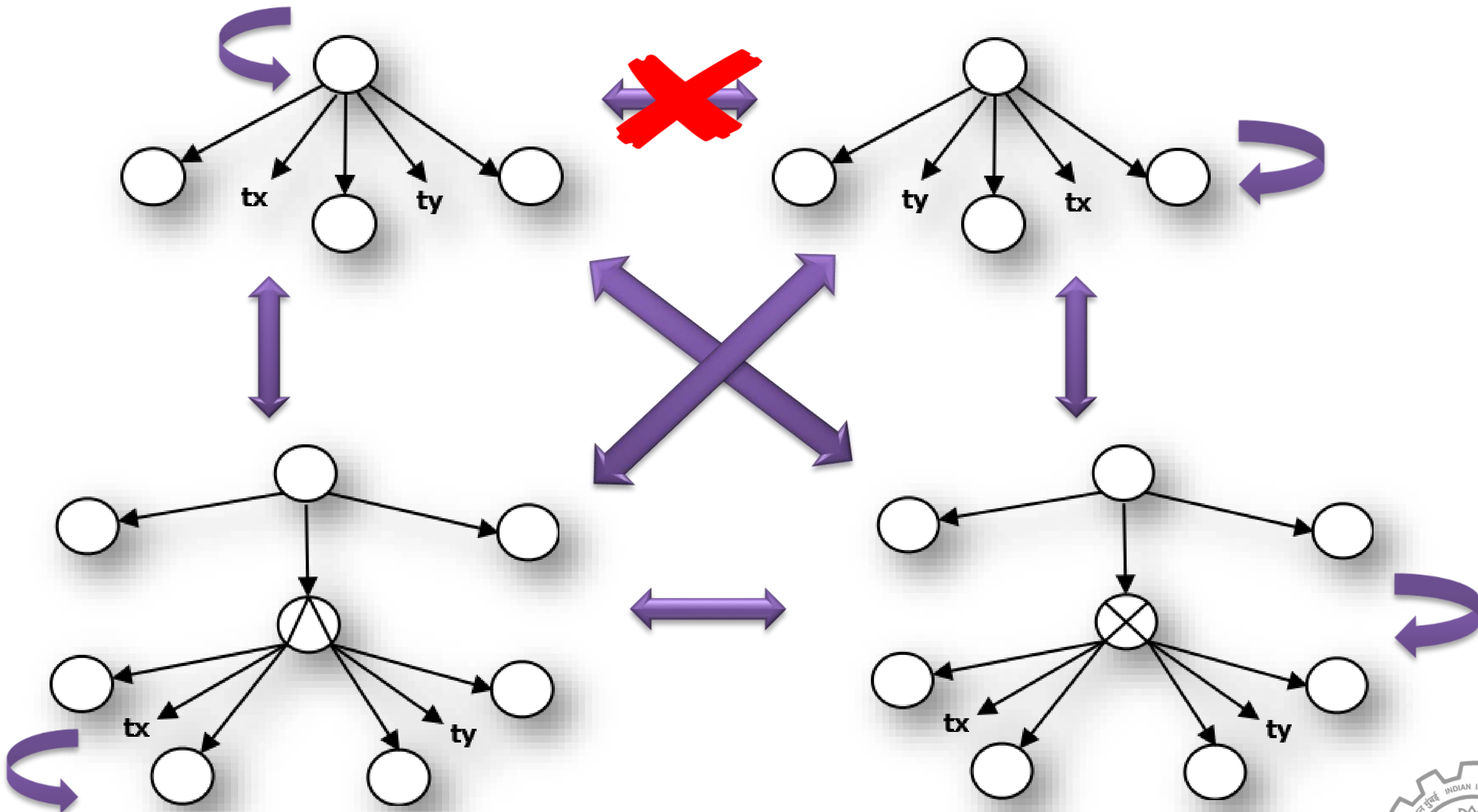


Pattern Alterations

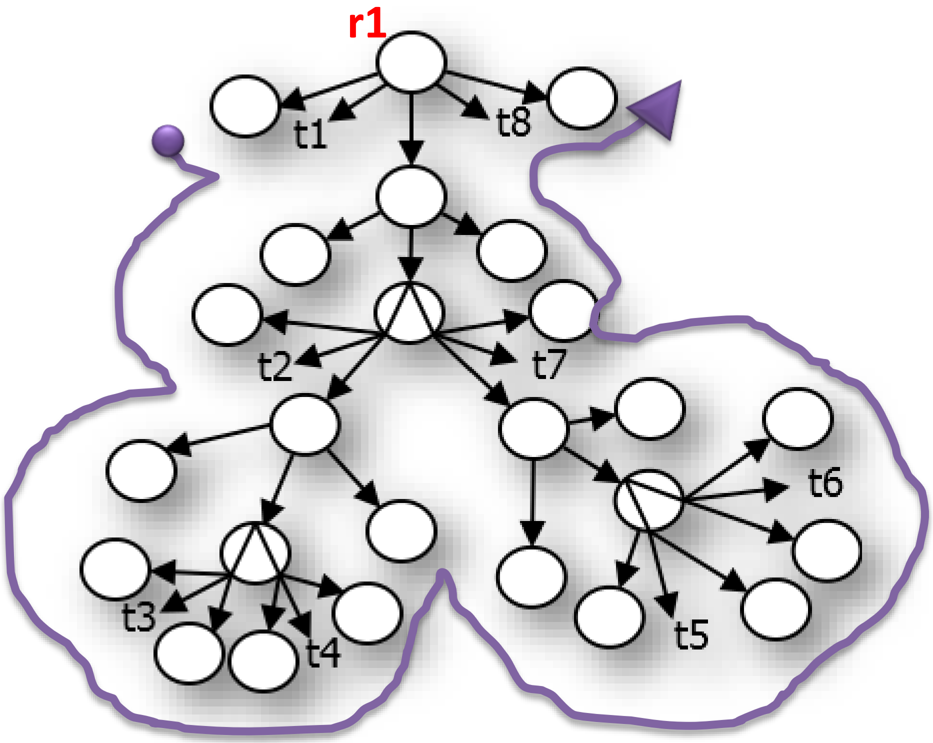


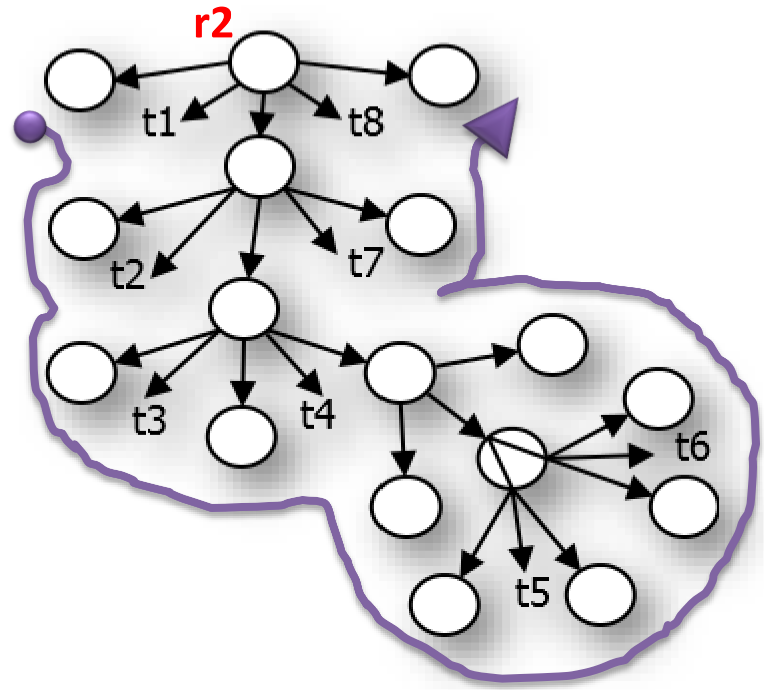


Peer Patterns



Yo-Yo compatibility





Yield of r1 =

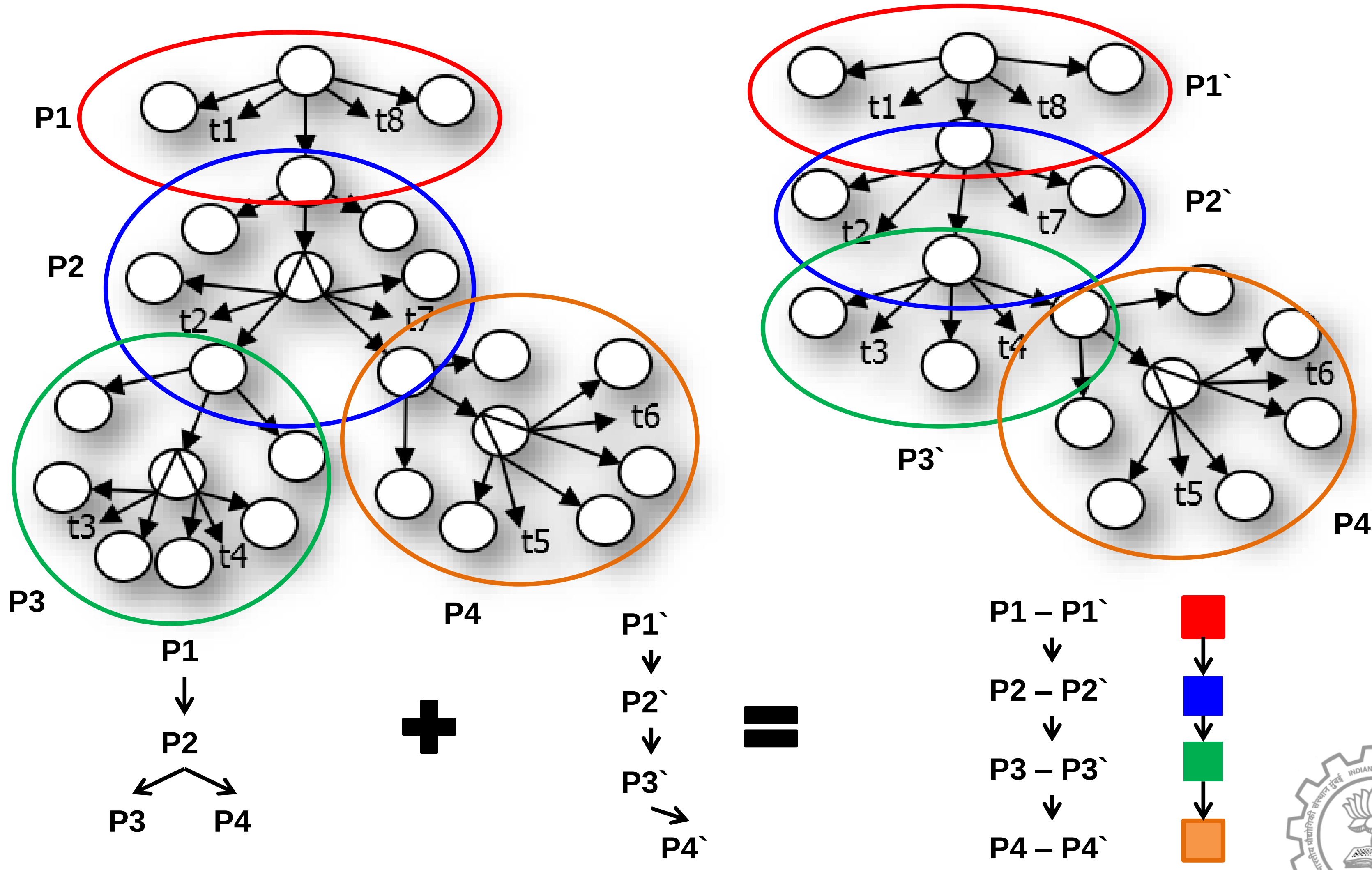
t1 { t2 { t3, t4 } , { t5, t6 } t7 } t8

Yield of r2 =

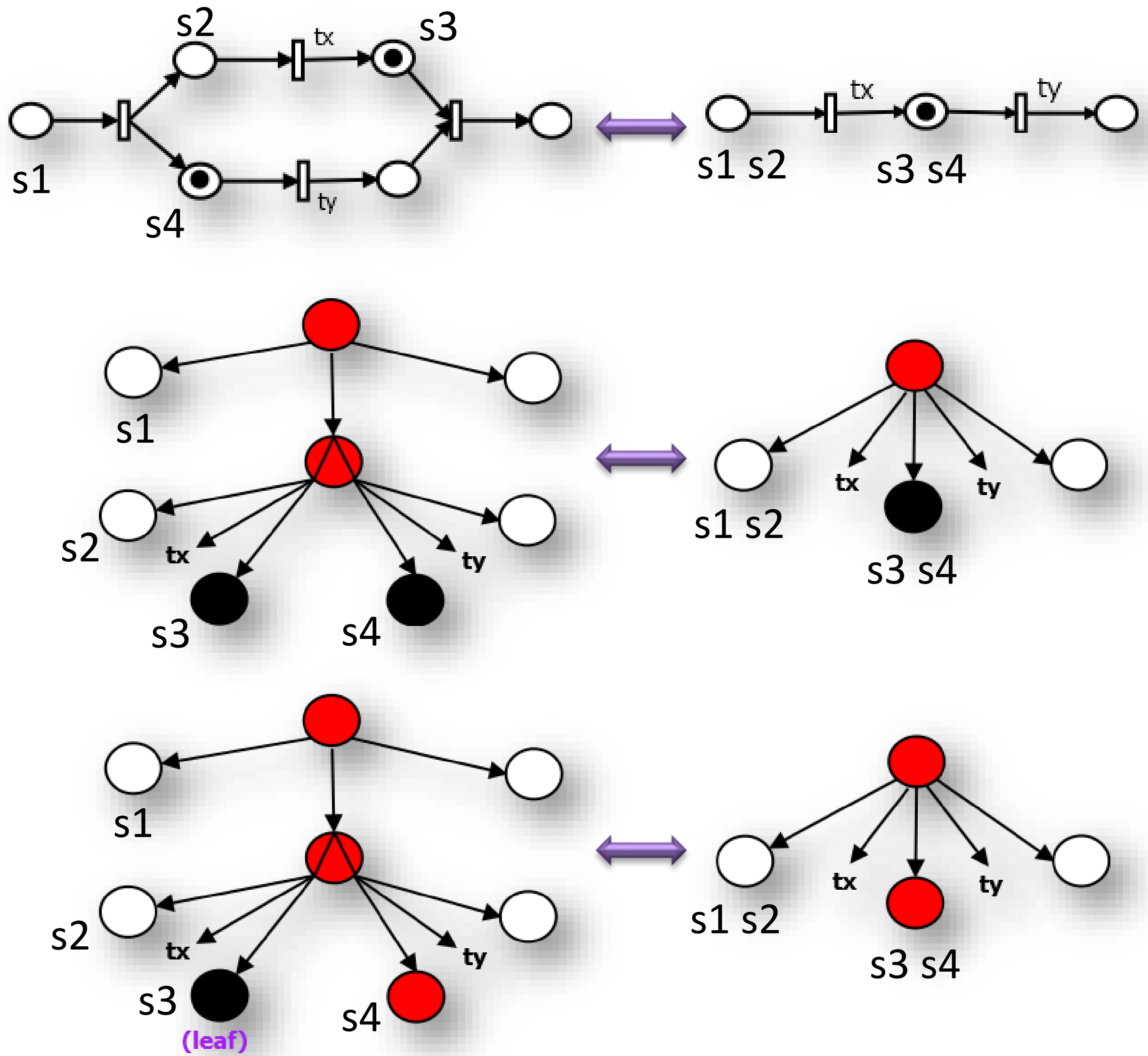
t1 t2 t3 t4 { t5, t6 } t7 t8

Both can generate the same sequence **t1 t2 t3 t4 t5 t6 t7 t8** → ***Folding order* exists**

Folding order



Pre-computed Token Transportation

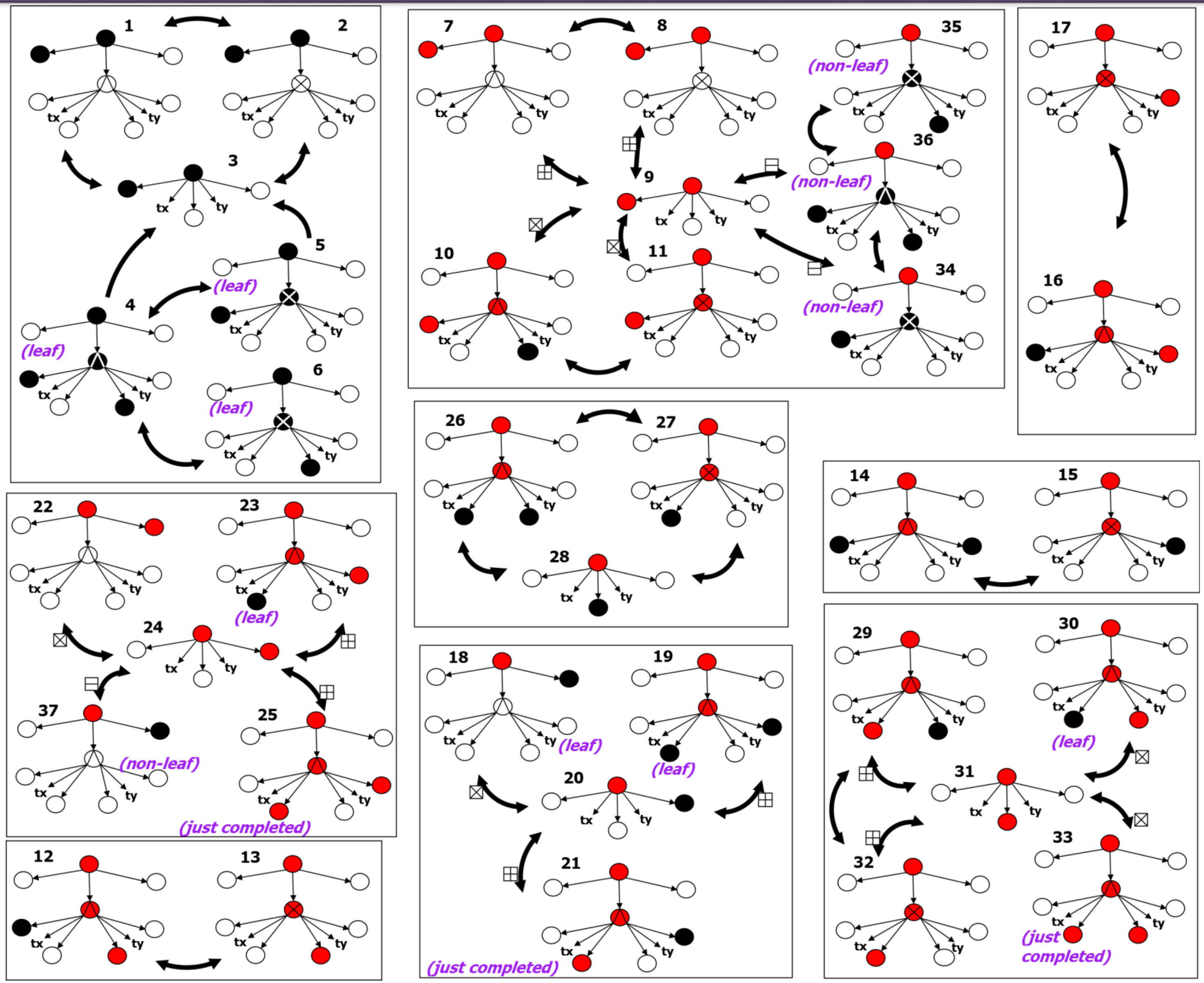


Compatible yields
s1 s2 tx s3 s4 ty …

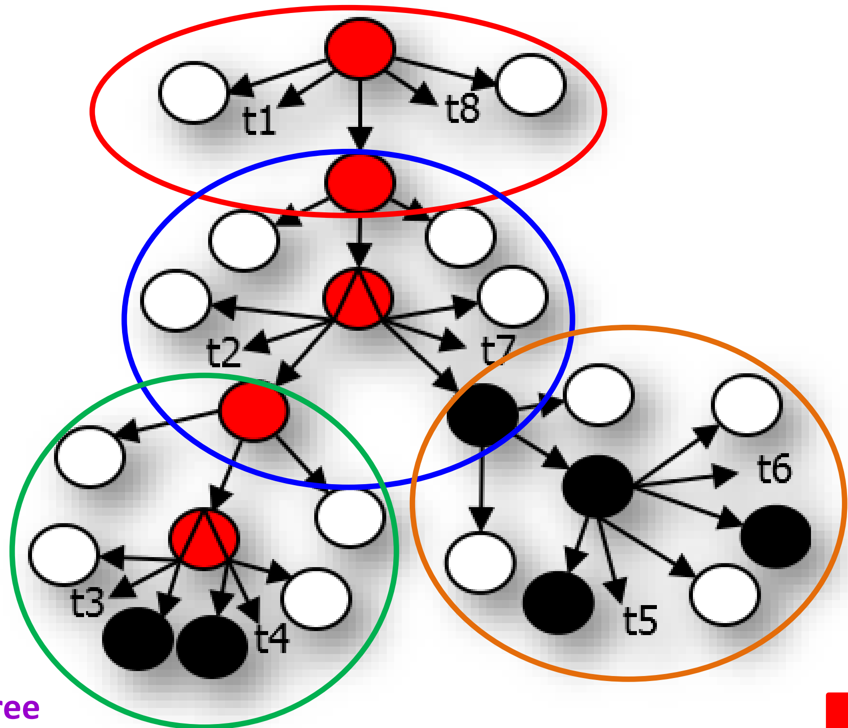
s3 = s4 = ε

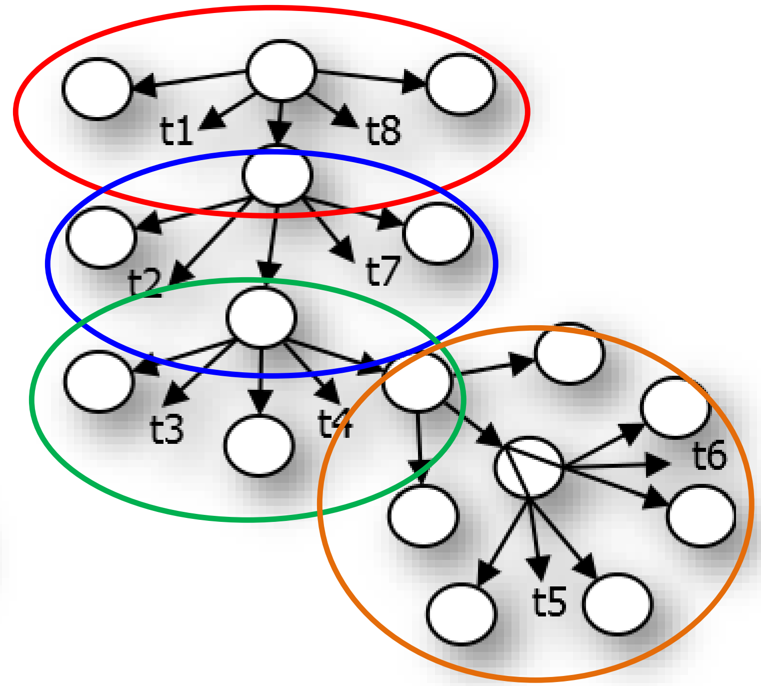
s3 = ε

Token Transportation Catalog

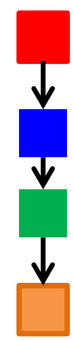


Yo-Yo Algorithm



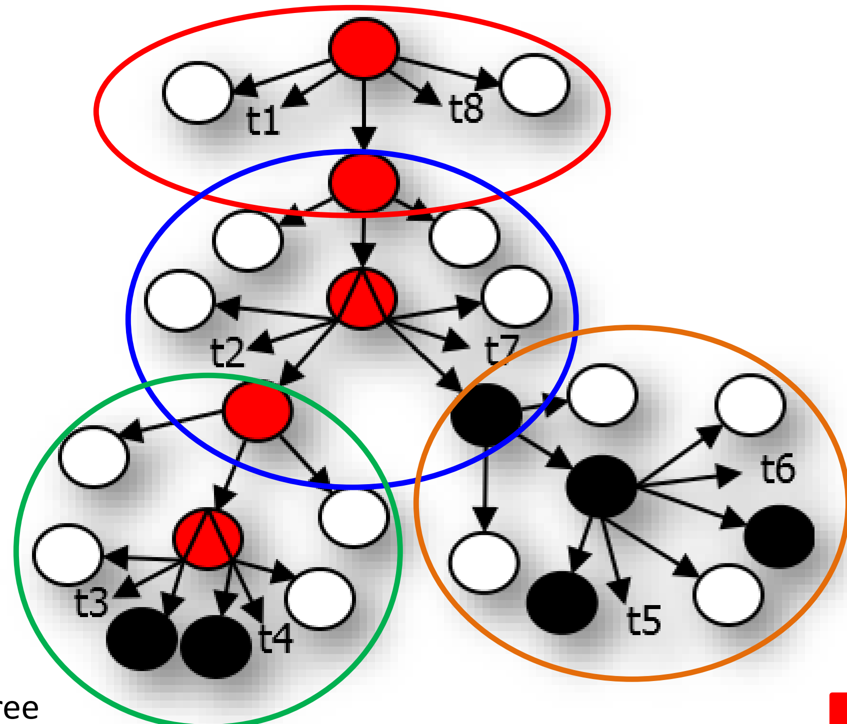


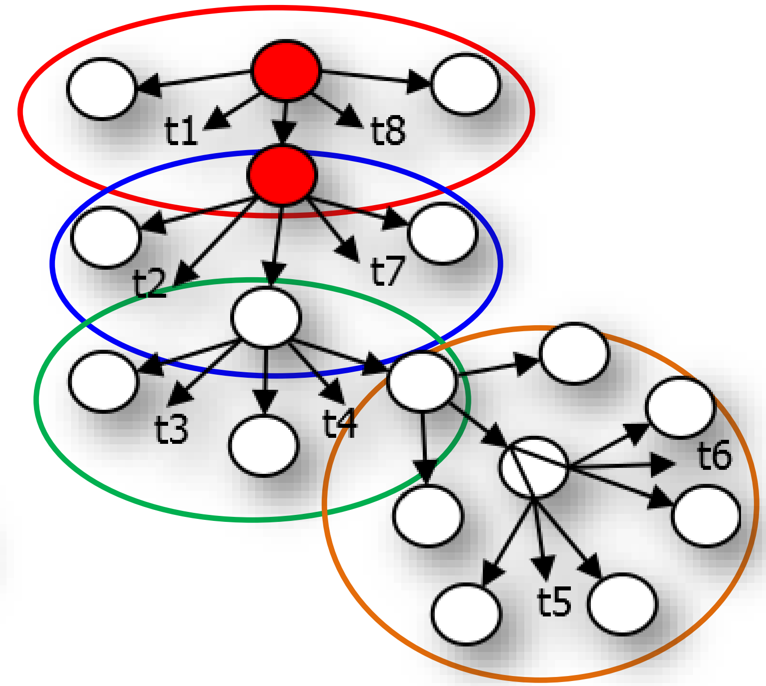
1. **Color old tree**
2. <**p-q**> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each next <**p-q**> in **F**,
   if q has colored root,
     if p is colored,
       color transfer between p, q
     else
       localPropagation(q)



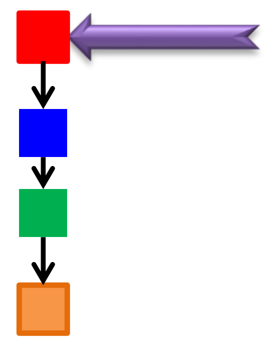
F

Yo-Yo Algorithm



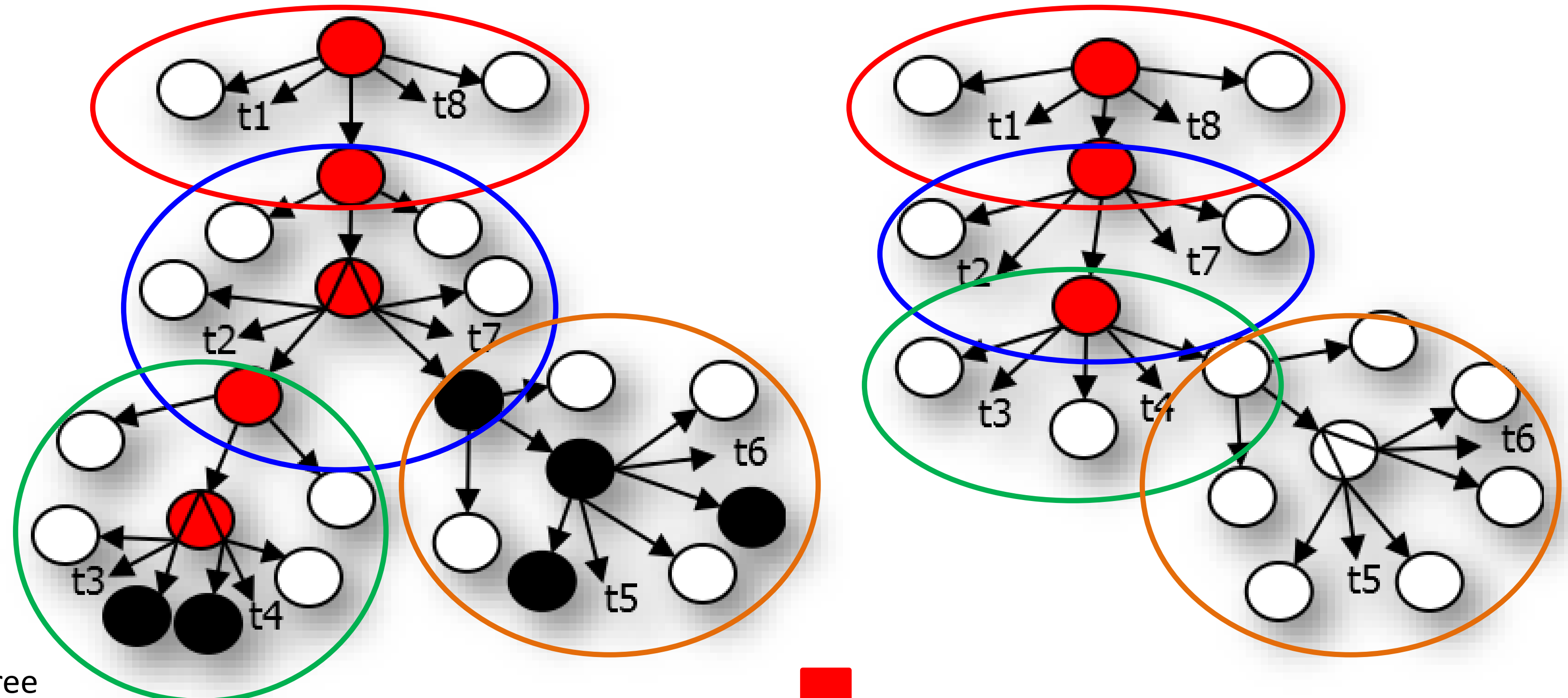


1. Color old tree
2. **<p-q> be 1st peer patterns to appear in folding order *F***
3. **Color transfer between p, q**
4. **for** each next <**p-q**> in **F**,
   - **if q** has colored root,
     - **if p** is colored,
       - color transfer between p, q
     - **else**
       - localPropagation(q)

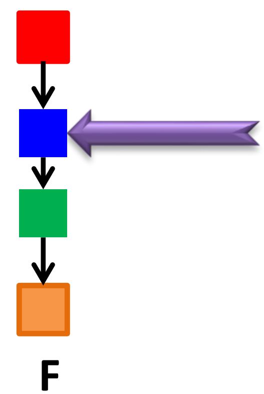


F

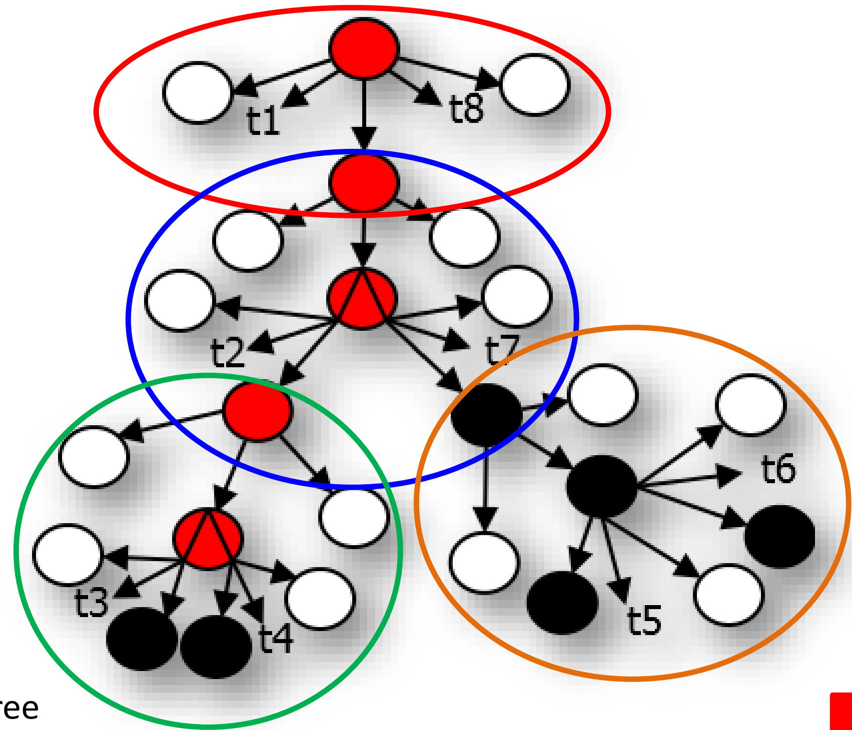
Yo-Yo Algorithm

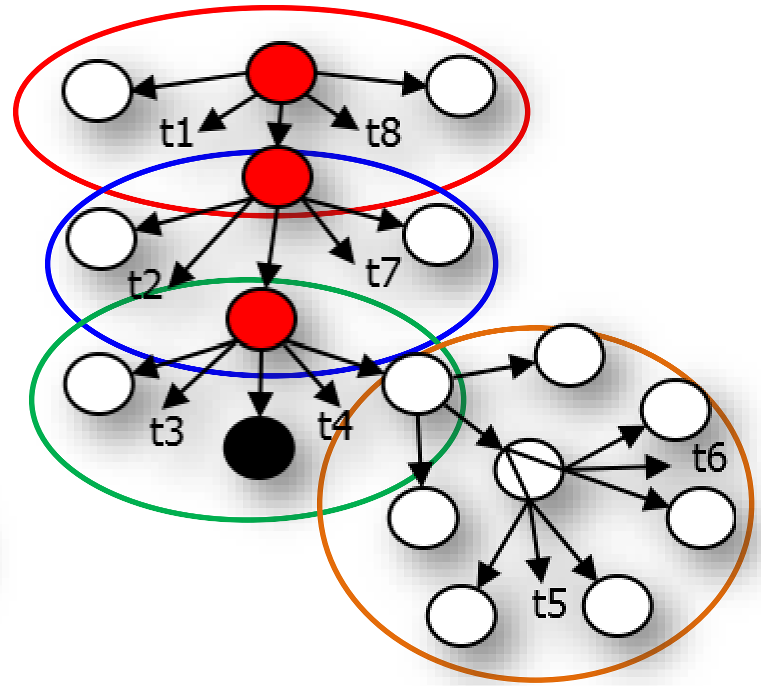


1. Color old tree
2. <p-q> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each next <p-q> in F,
   - if q has colored root,
     - if p is colored,
       - color transfer between p, q
     - **else**
       - localPropagation(q)

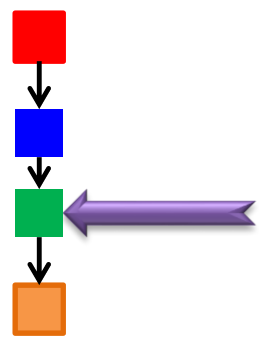


Yo-Yo Algorithm



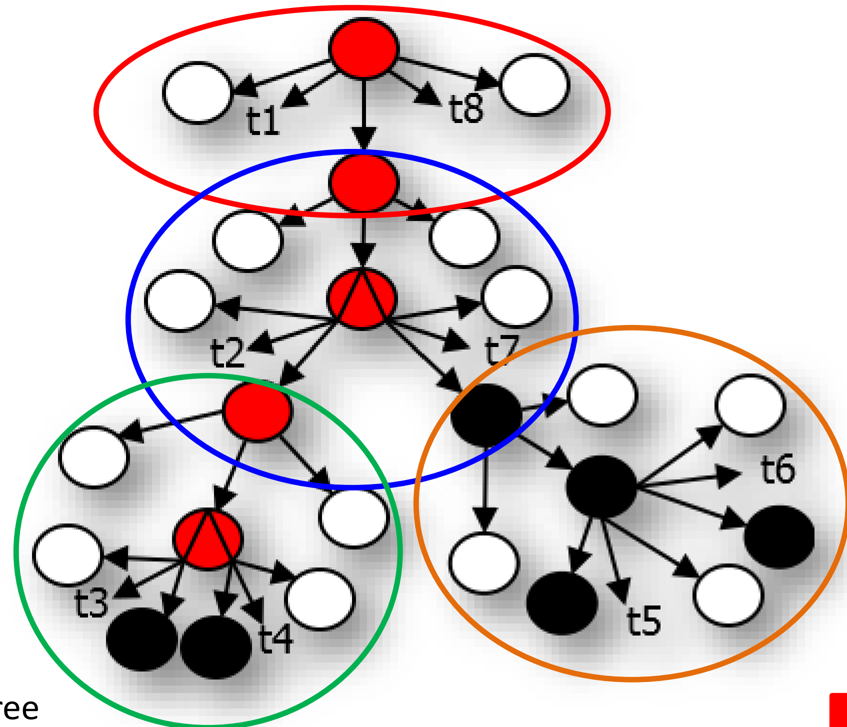


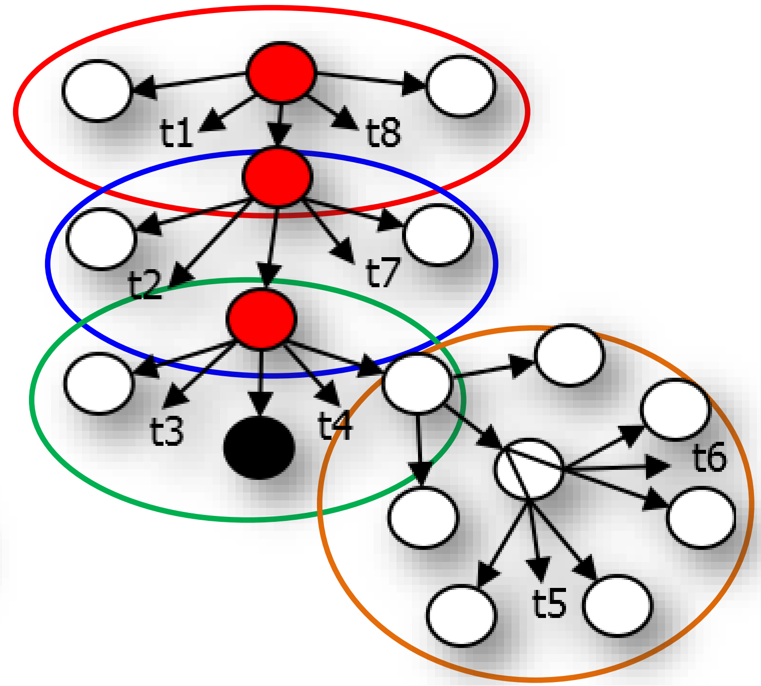
1. Color old tree
2. <p-q> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each **next <p-q> in F,**
   if q has colored root,
   if p is colored,
   color transfer between p, q
   else
   localPropagation(q)



F

Yo-Yo Algorithm





1. Color old tree
2. <p-q> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each **next <p-q> in F,**

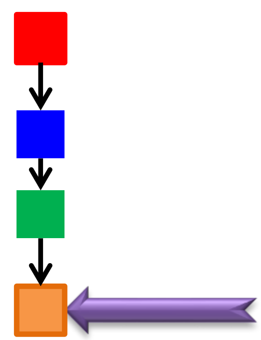
   if q has colored root, **false**

   if p is colored,

   color transfer between p, q

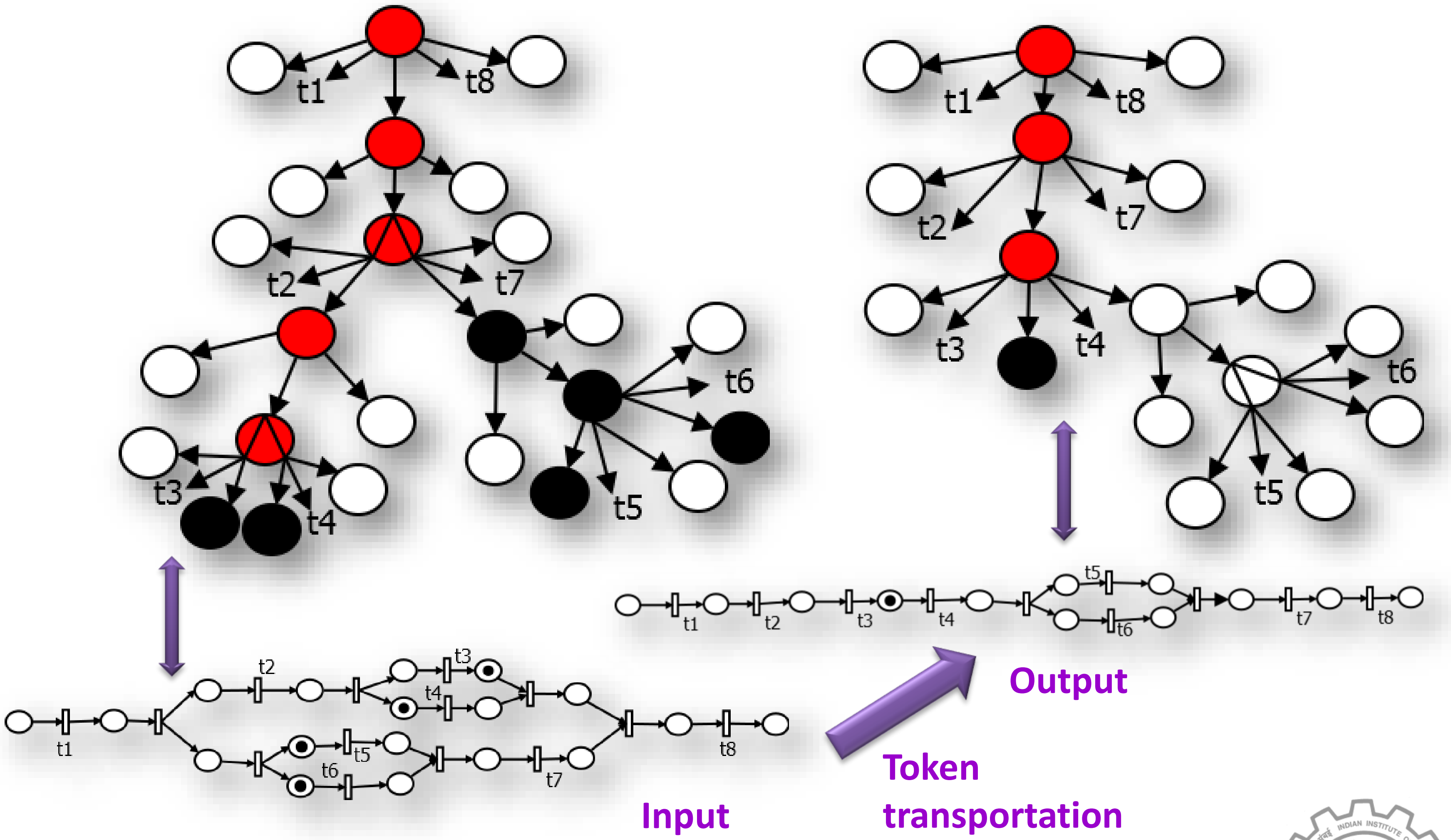
   else

   localPropagation(q)



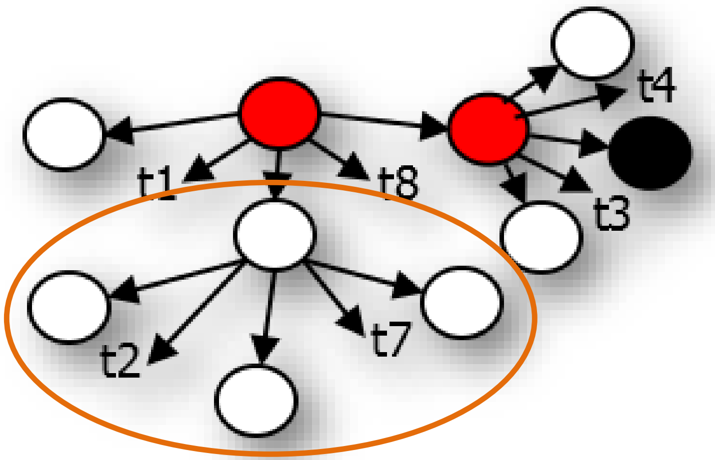
F

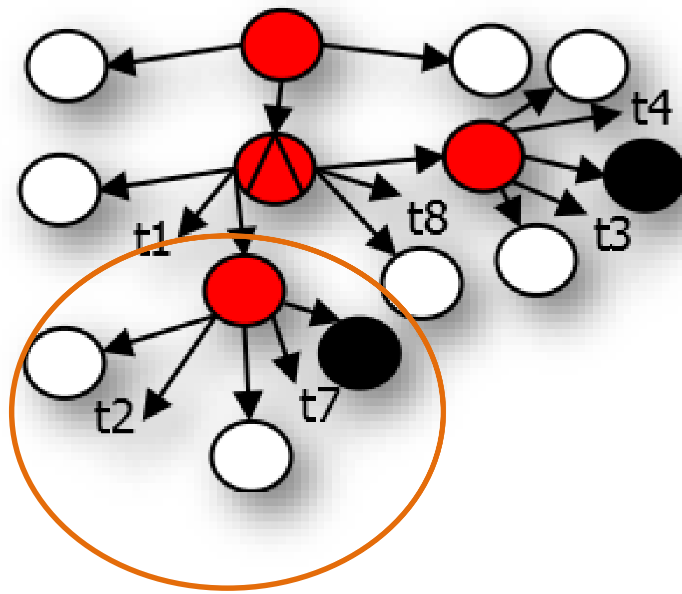
Yo-Yo Algorithm



Max. no. of Transportation Steps = no. of patterns (linear time complexity)

Yo-Yo Algorithm

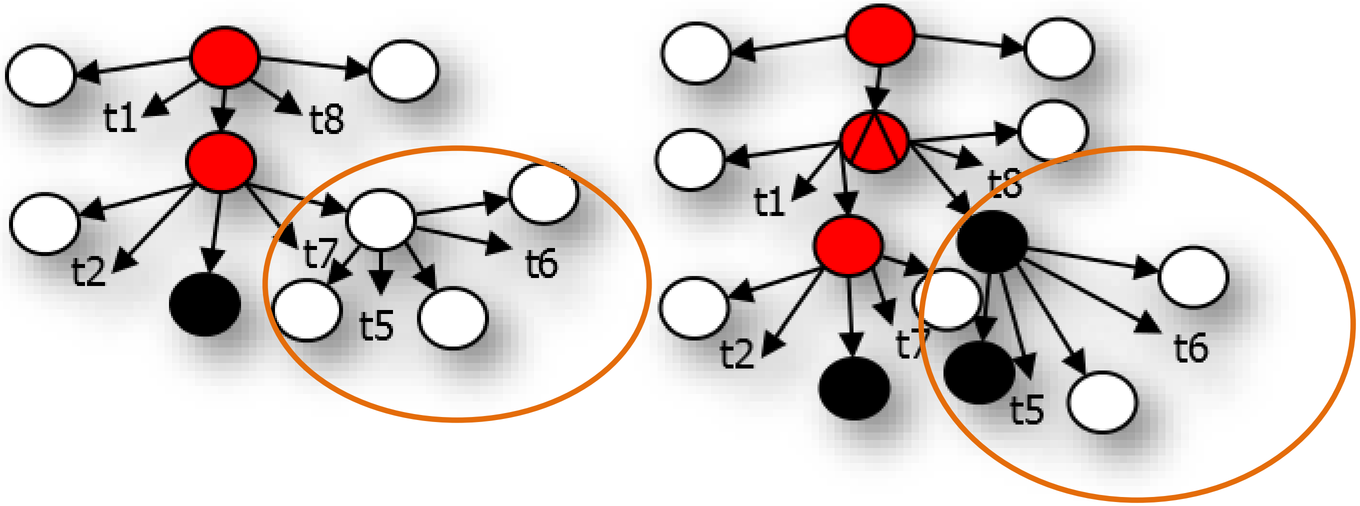




1. Color old tree
2. <**p-q**> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each next <**p-q**> in F,
   - **if q** has colored root,
     - **if p** is colored,
       - color transfer between p, q
     - **else**
       - localPropagation(q)

Red root → color rightmost child

Yo-Yo Algorithm



1. Color old tree
2. <**p-q**> be **1st peer patterns** to appear in **folding order *F***
3. Color transfer between p, q
4. **for** each next <**p-q**> in F,
   if q has colored root,
     if p is colored,
       color transfer between p, q
     else
       localPropagation(q)

Black root → color leftmost child

Correctness

Catalog Completeness:
Token transportation catalog is complete w.r.t. the 6 change patterns

Lemma 1:
For two Yo-Yo compatible derivation trees, consistent coloring between The top peer patterns guaranties consistent coloring between their immediate child peer patterns

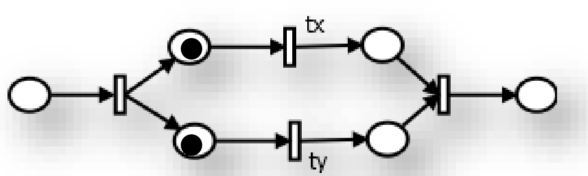
Lemma 2:
Lemma 1 can be repeated for all parent-child peer pairs across two Yo-Yo compatible derivation trees

Correctness: Catalog Completeness

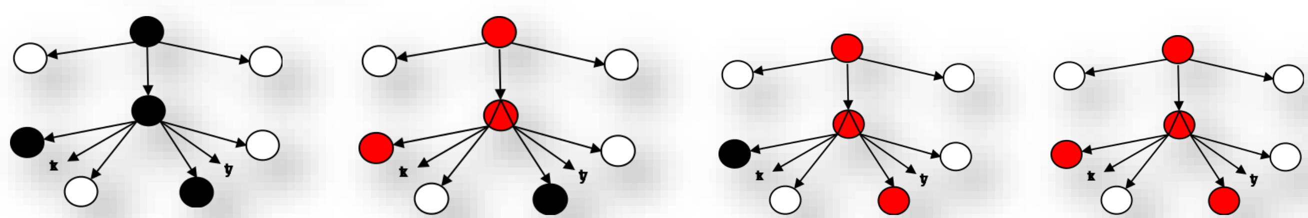
6 situations!

Type of Node	Marking Status	Execution Status	Color
Folded	Unmarked	Null/full-executed	Uncolored
Unfolded	Unmarked	NA	Uncolored
Folded	Marked	Null executed	Black
Unfolded	Marked	NA	Black
Folded	Marked	Partially executed	Red
Folded	Marked	Full executed	Red

3 color codes



1 marking, 2 x 4 x 4 = 32 situations
2 x 2 = 4 colorings

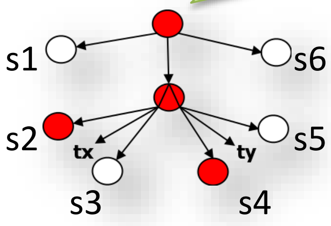


Correctness: Catalog Completeness

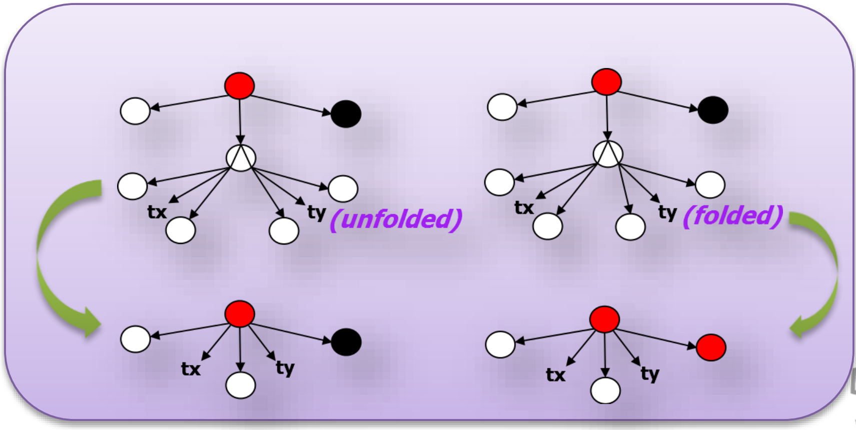
Pattern	# valid markings	# actual situations	# colorings In derivation trees	# non-migratable colorings	# colorings where node type changes mapping
SEQ	3	28	6	0	0
AND	6	420	20	3	2
XOR	6	116	12	2	2
			38	**-5**	**+4**

37 colorings in catalog

Yield is
s1 { s2 tx s3, s4 ty s5 } s6
SEQ:
s1 s2 tx s3 s4 ty s5 s6
XOR:
s1 s2 tx s3 s6 or **s1 s4 ty s5 s6**



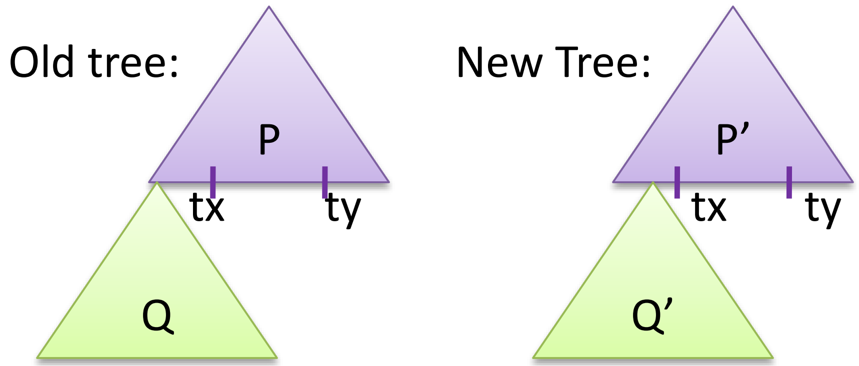
e.g. non-migratable



e.g. node type changes mapping

Correctness: Lemma 1

Roots of two derivation trees are yield compatible.
Consistent color transfer between the top patterns P and P’ → consistency ensured between their child peers Q and Q’

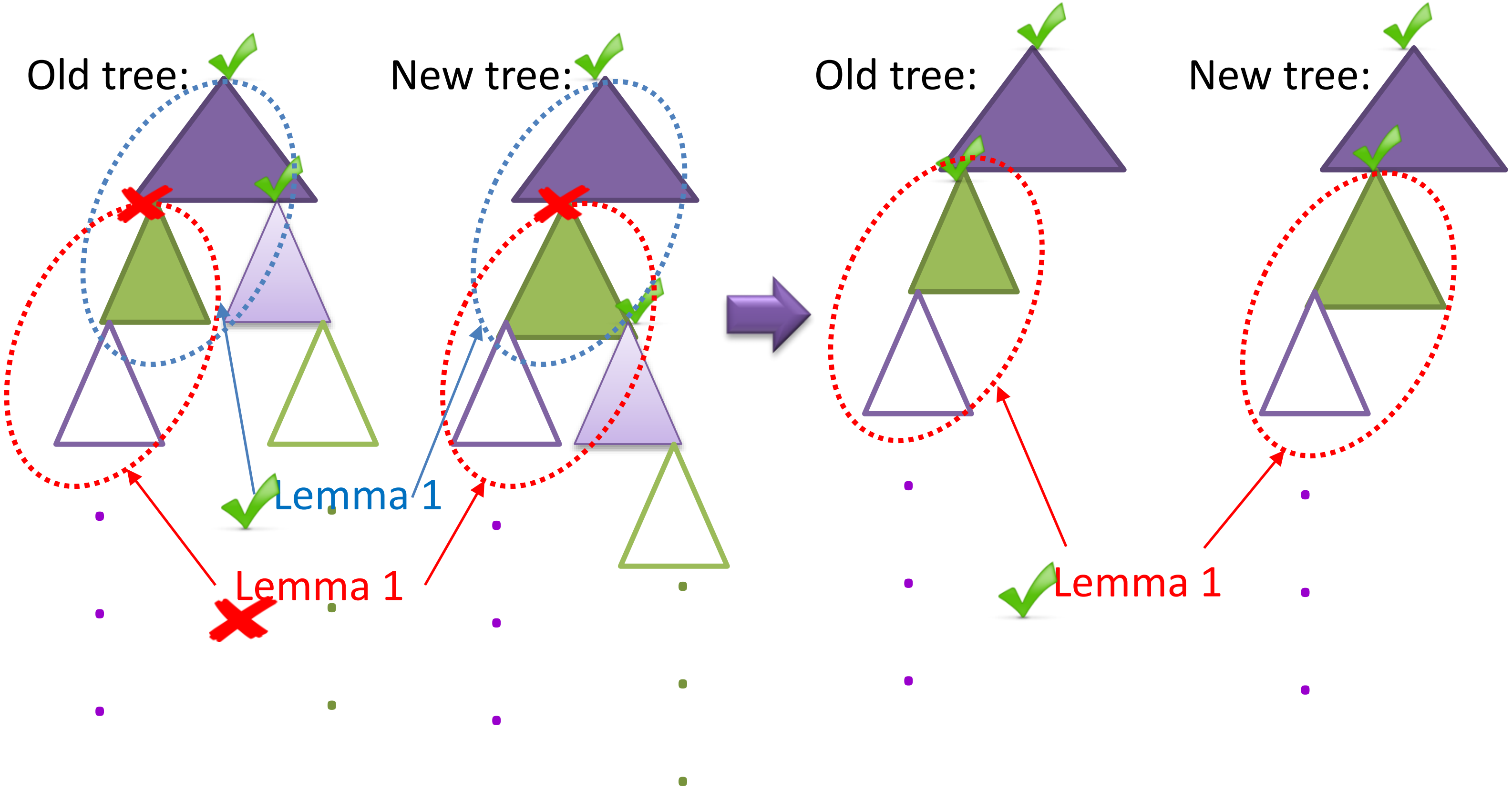


Same relative Positions of Q and Q’ w.r.t. P and P’

Root of Q	Red	Black	uncolored
Root of Q’	Red/uncolored	Black/uncolored	uncolored

Possible to refine root colors of Q and Q’ consistently

Correctness: Lemma 2

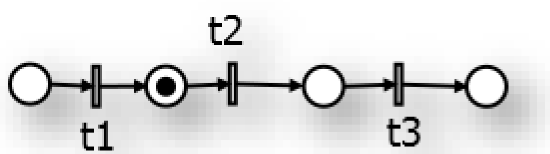


Preservation of yield compatibility through folding order

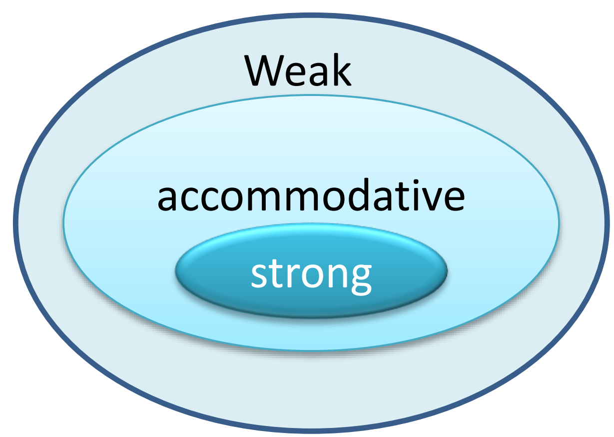
Lookahead Consistency Models

Lookahead Trace based Consistency

Consistency Model Name	Description
Strong Lookahead	same lookahead trace sets of consistent marking
Accommodative Lookahead	old lookahead trace set preserved in new
Weak Lookahead	at least one old lookahead trace preserved in new

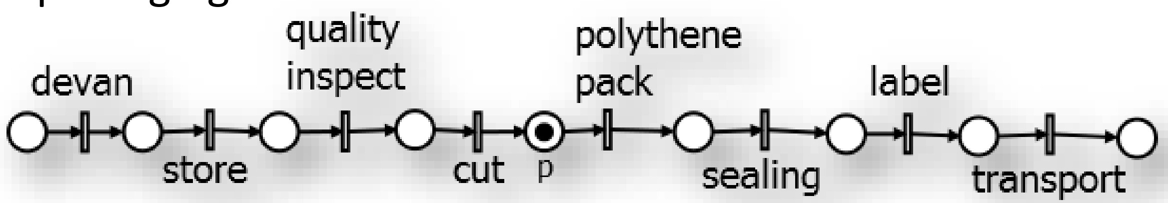


Lookahead trace: t2,t3



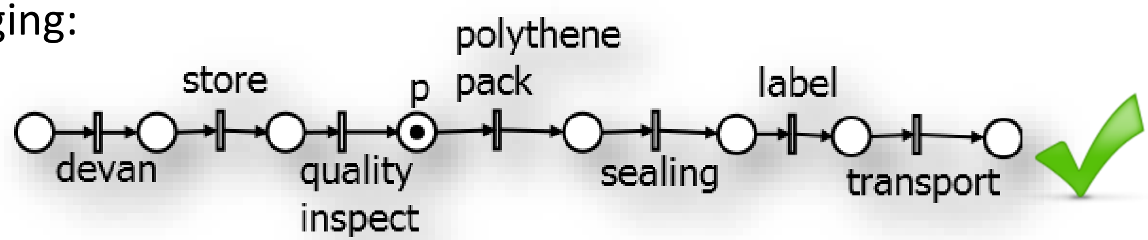
Strong lookahead

Milk-products packaging:



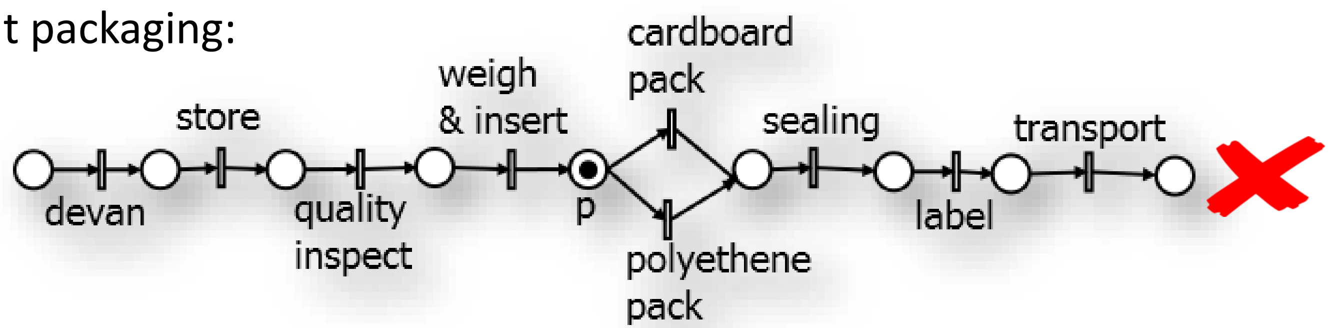
Polythene pack, sealing, label, transport

Chocolate packaging:



Polythene pack, sealing, label, transport

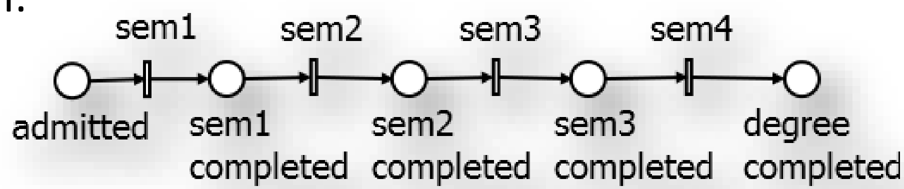
Dry fruit packaging:



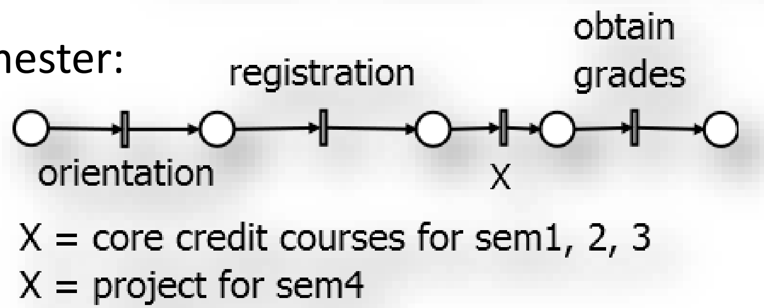
Polythene pack, sealing, label, transport; Cardboard pack, sealing, …

Accommodative lookahead

Academic program:

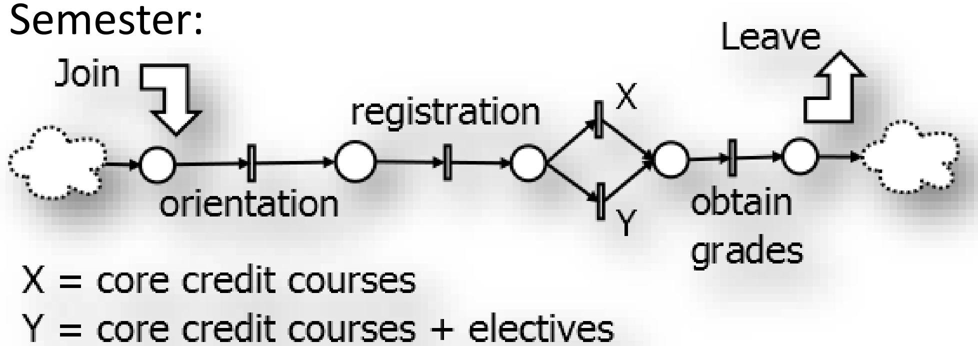


Home university Semester:



Orientation, reg., X, ob. grades

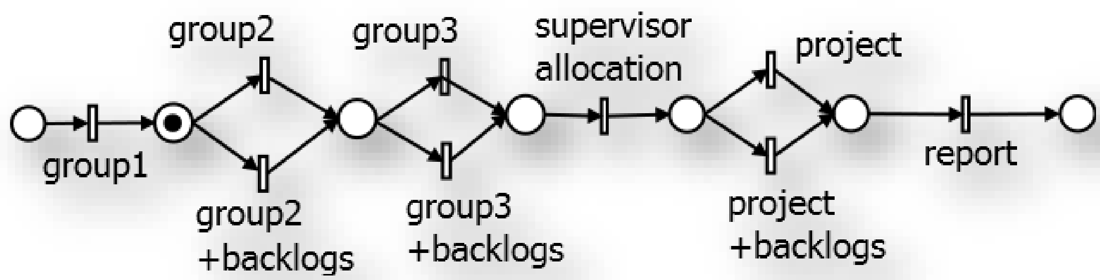
Foreign university Semester:



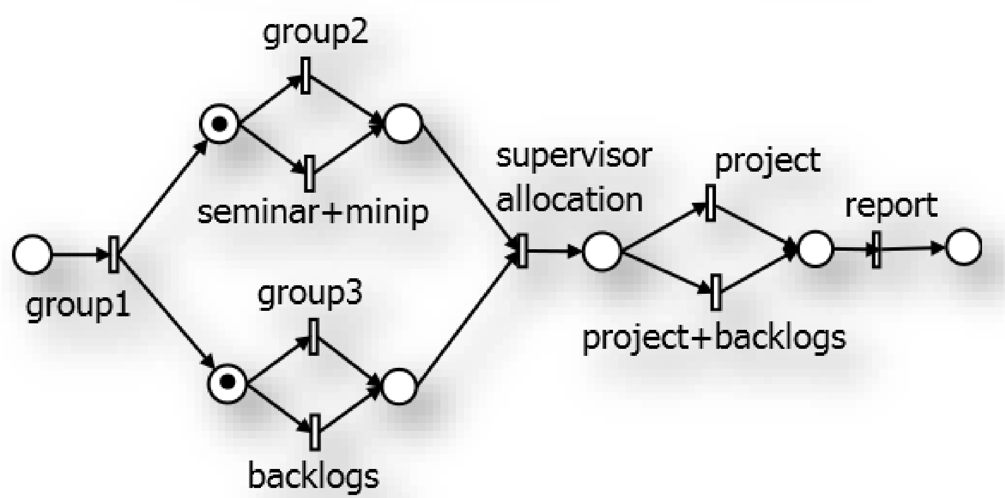
Orientation, reg., X, ob. Grades; <u>Orientation, reg., Y, ob. Grades</u>

Weak lookahead

Old Curriculum:



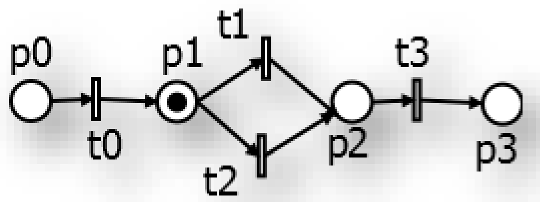
New Curriculum:

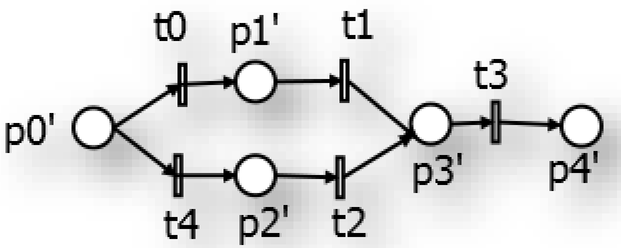


gr1, gr2, gr3, sup. alloc., project, report; …

<u>gr1, gr2+backlog, gr3, sup. alloc., project, report; …</u>

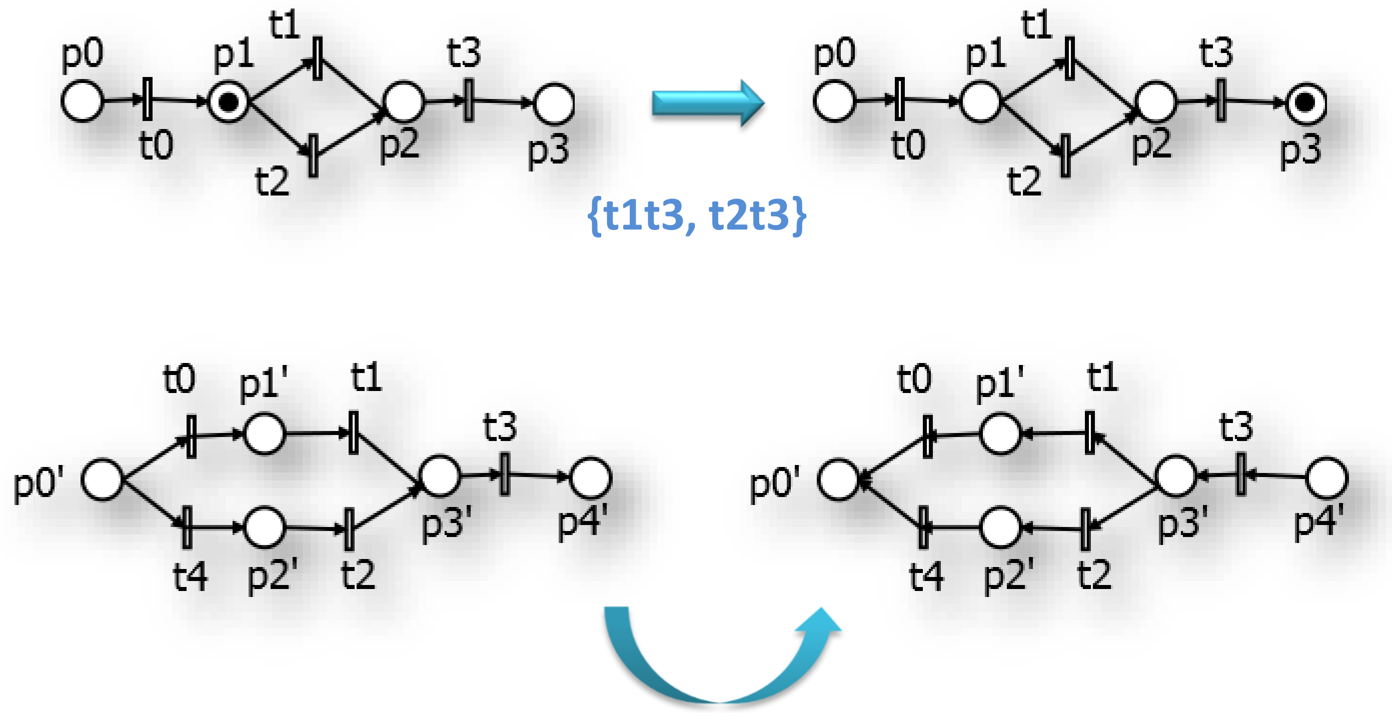
Algo 1: Computing weak lookahead marking



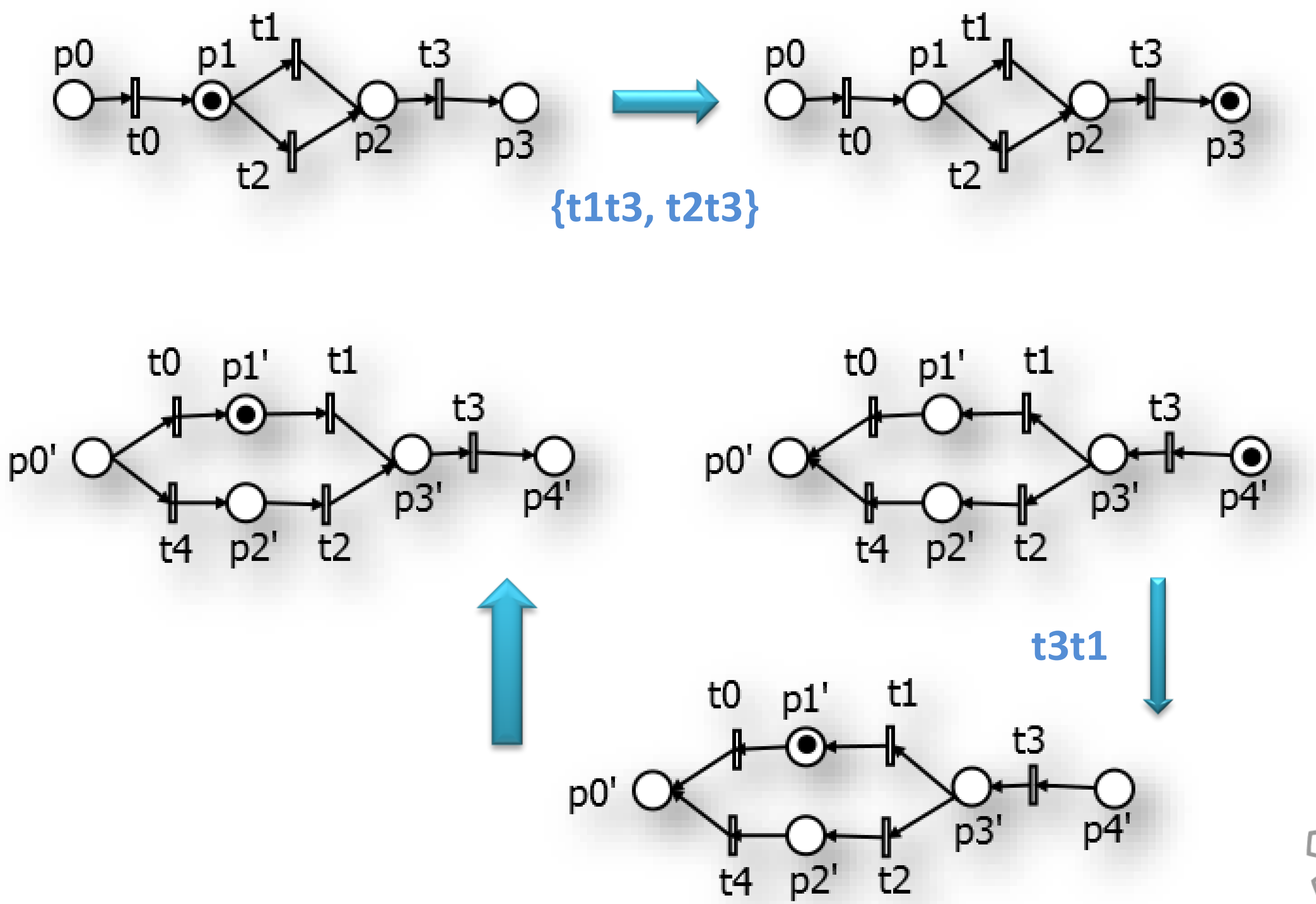


1. Acyclic nets
2. No duplicate transitions

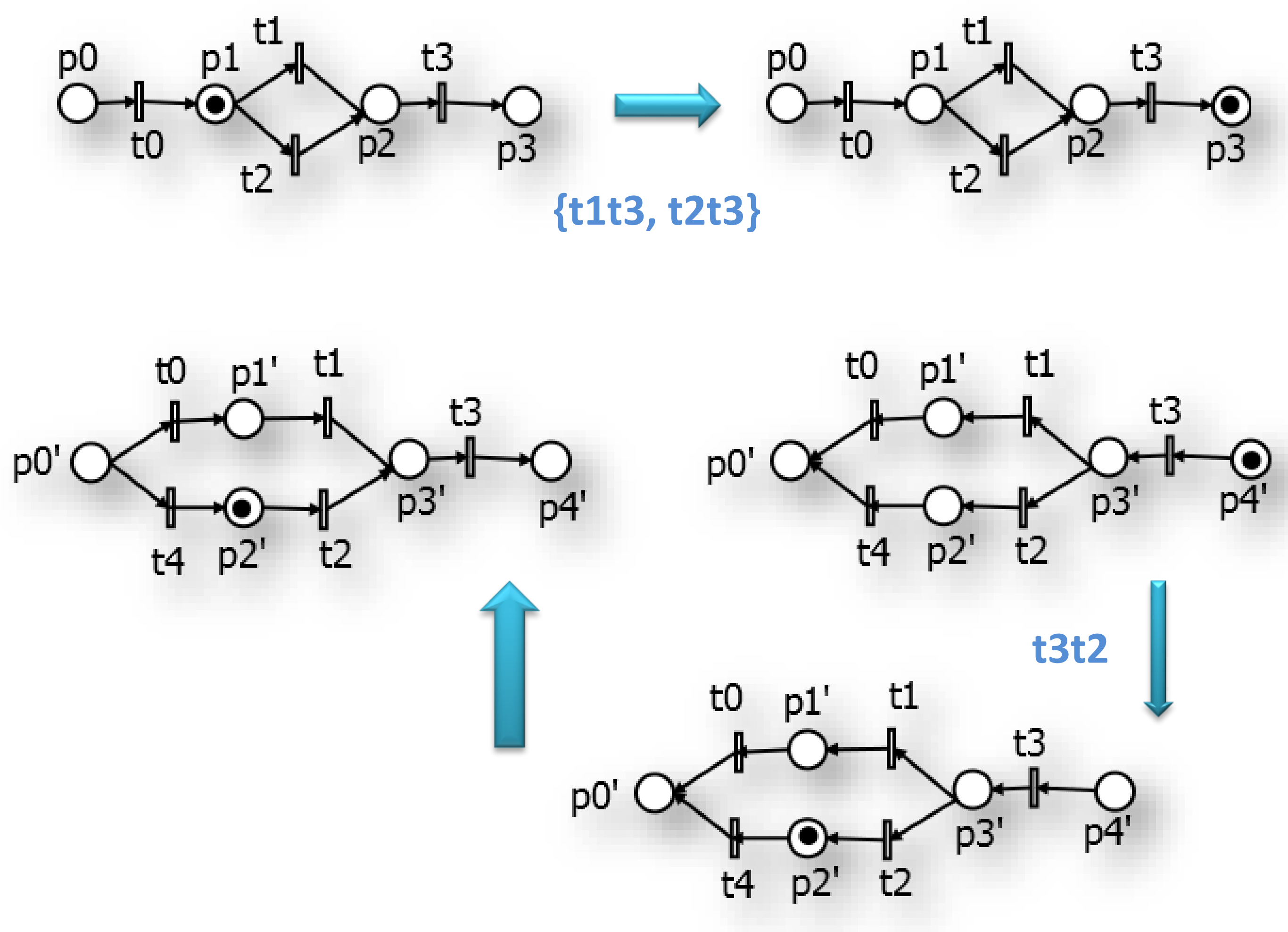
Algo 1: Computing weak lookahead marking



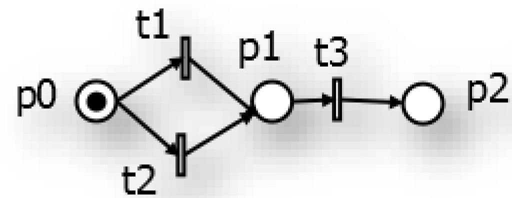
Algo 1: Computing weak lookahead marking

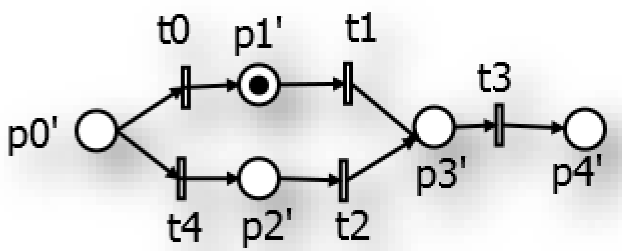


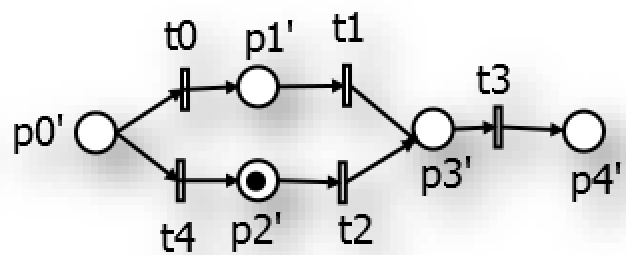
Algo 1: Computing weak lookahead marking



Algo 1: Computing weak lookahead marking





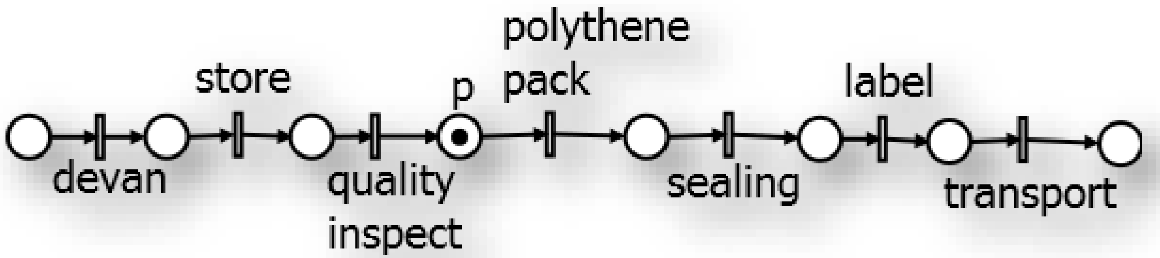


Traces = { t1t3, t2t3 } lookahead traces

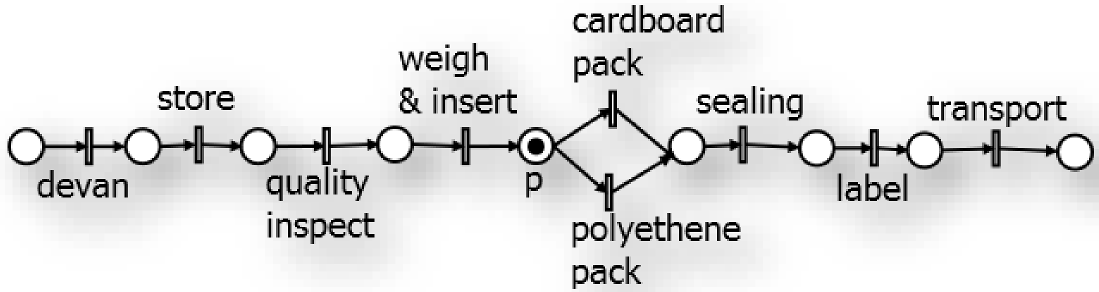
L = { t1t3, t2t3 } preserved lookahead traces

S = { {p1'}, {p2'} } weak lookahead consistent marking

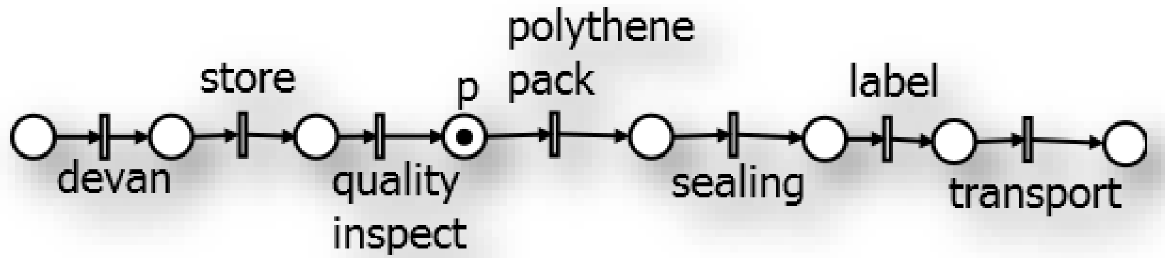
Algo 2: Accept/Reject Branching



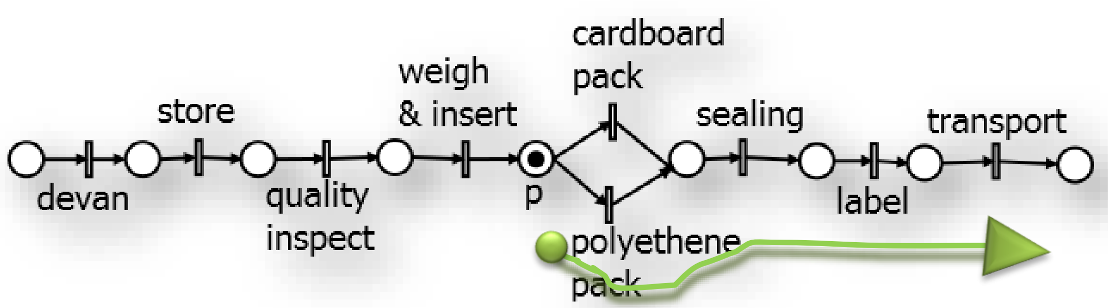
L = Polythene pack, sealing, label, transport



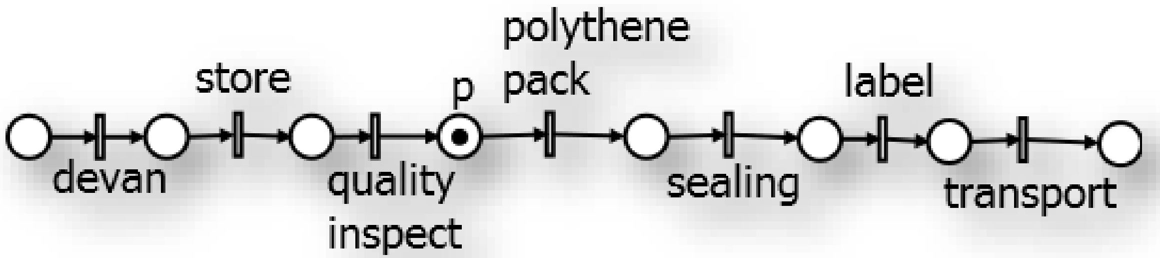
Algo 2: Accept/Reject Branching



L = Polythene pack, sealing, label, transport

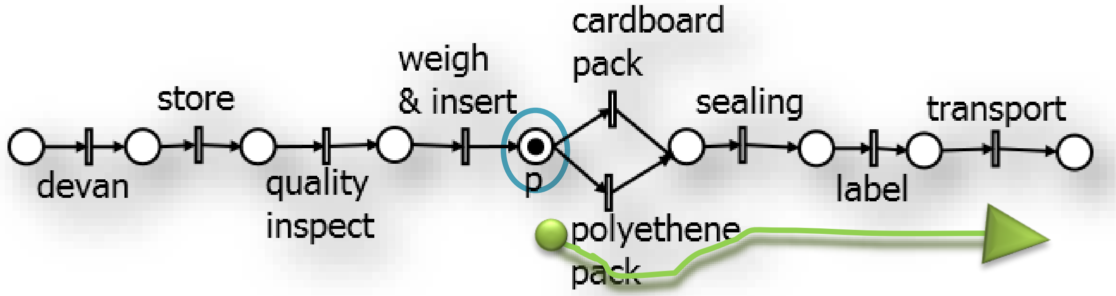


Algo 2: Accept/Reject Branching

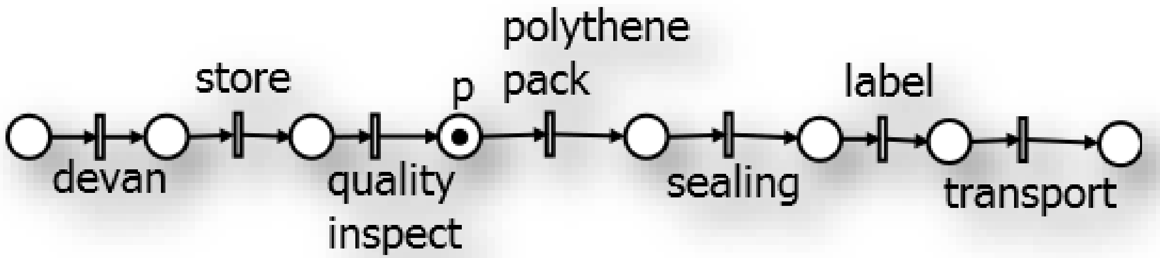


L = Polythene pack, sealing, label, transport

P_{XOR} = { p }



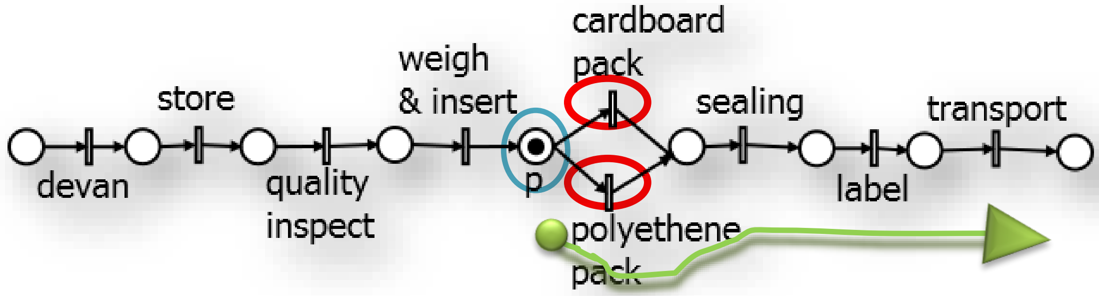
Algo 2: Accept/Reject Branching



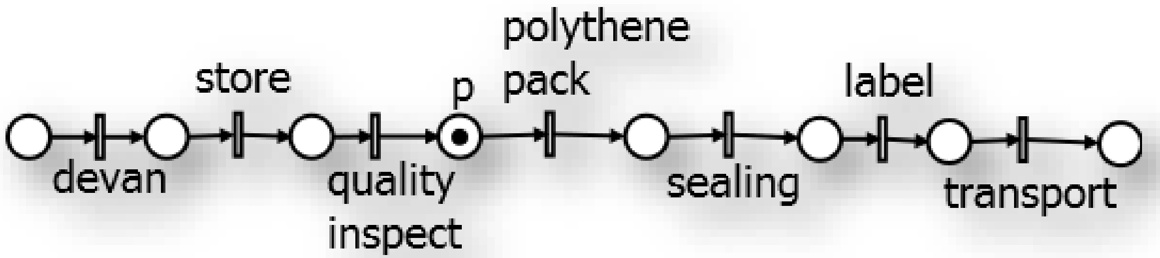
L = Polythene pack, sealing, label, transport

P_{XOR} = { p }

$T_{potential}$ = { cardboard pack, polythene pack }



Algo 2: Accept/Reject Branching

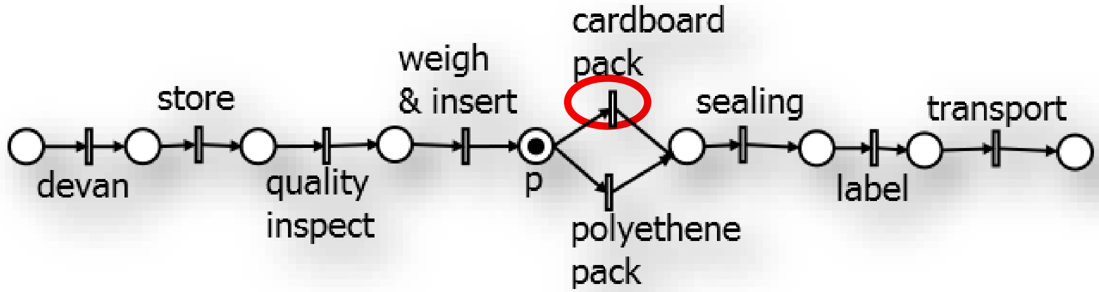


L = Polythene pack, sealing, label, transport

P_{XOR} = { p }

$T_{potential}$ = { cardboard pack, polythene pack }

T_{block} = {cardboard pack}



Inferences

$L \neq \{\ \} \rightarrow$ weak

$+\ |S| = 1, L = \text{Traces} \rightarrow$ accommodative

$+\ T_{block} = \{\ \} \rightarrow$ strong

$S = \{\ \} \rightarrow$ no lookahead

$|s|>1 \rightarrow$ no single marking can fire all preserved lookahead traces

Traces of lookahead traces

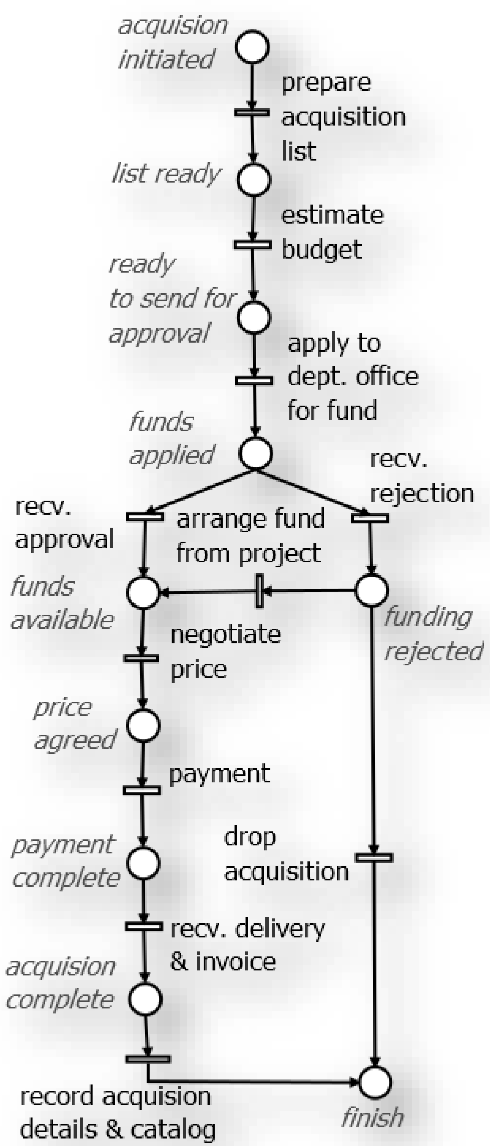
L preserved lookahead traces

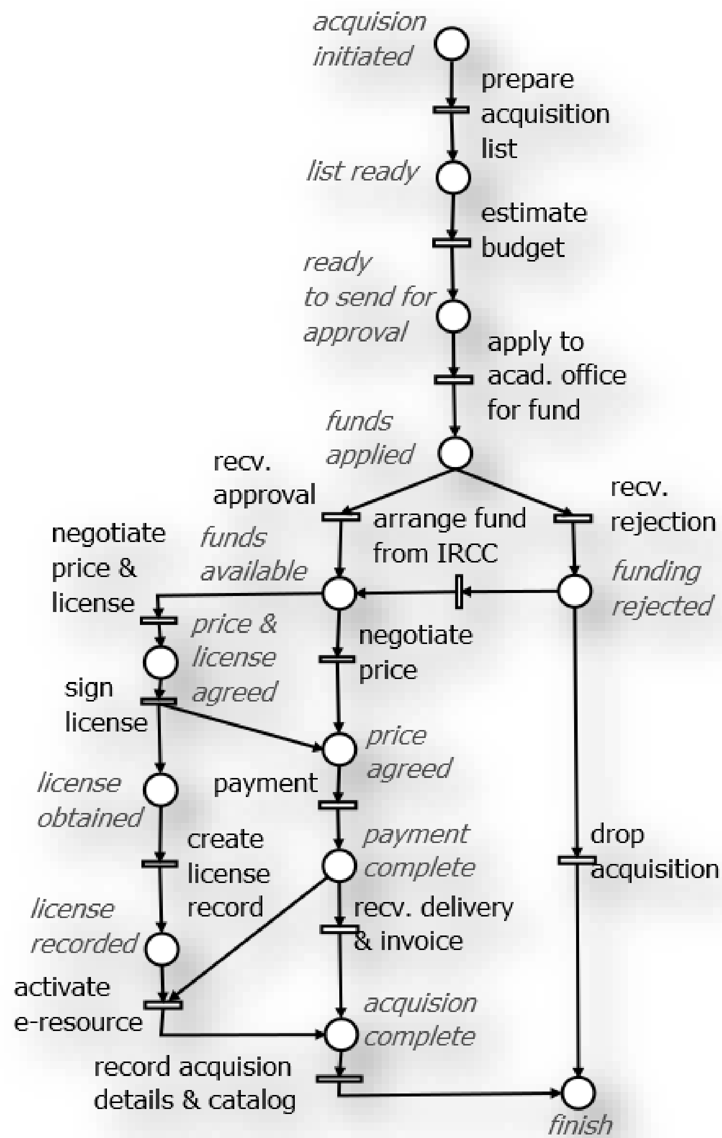
S weak consistent markings

T_{block} contradictory head-transitions

Practical Example: Resource Acquisition

Departmental Process

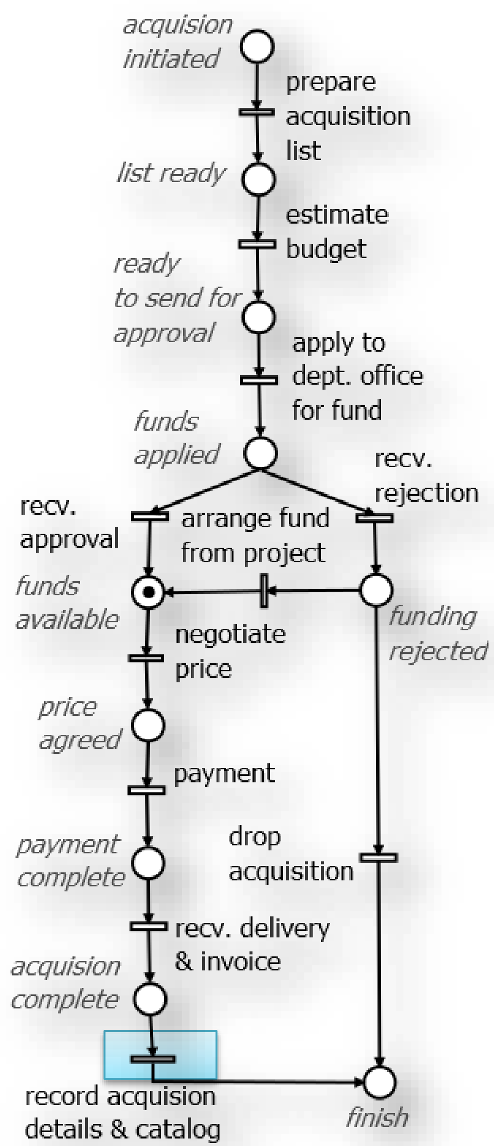




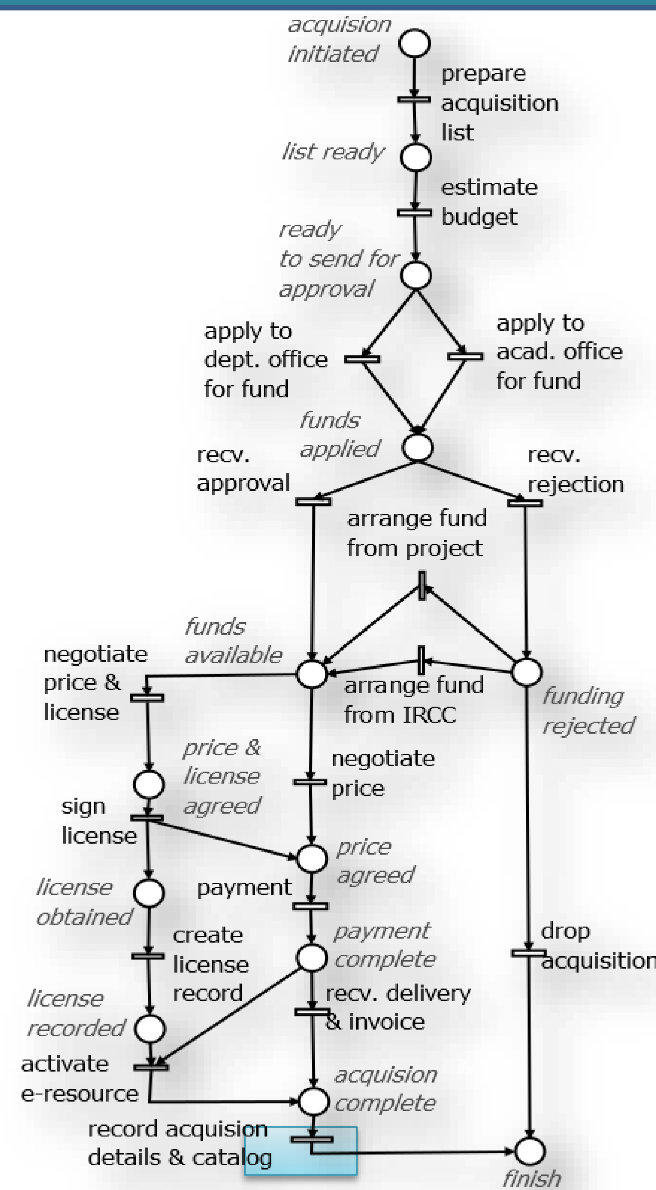
Central Library Process

Practical Example: Resource Acquisition

Departmental Process Instance

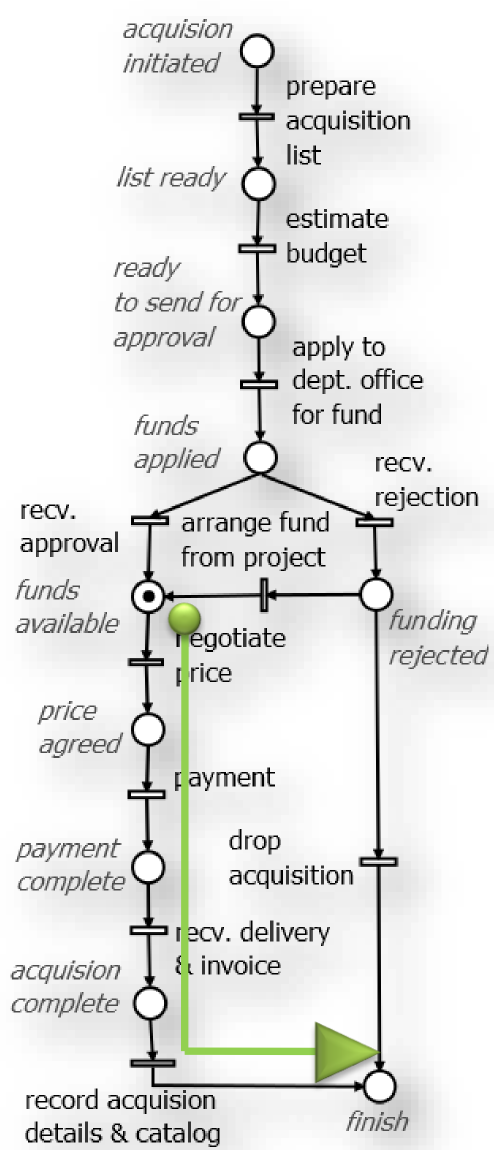


Re-engineered Process



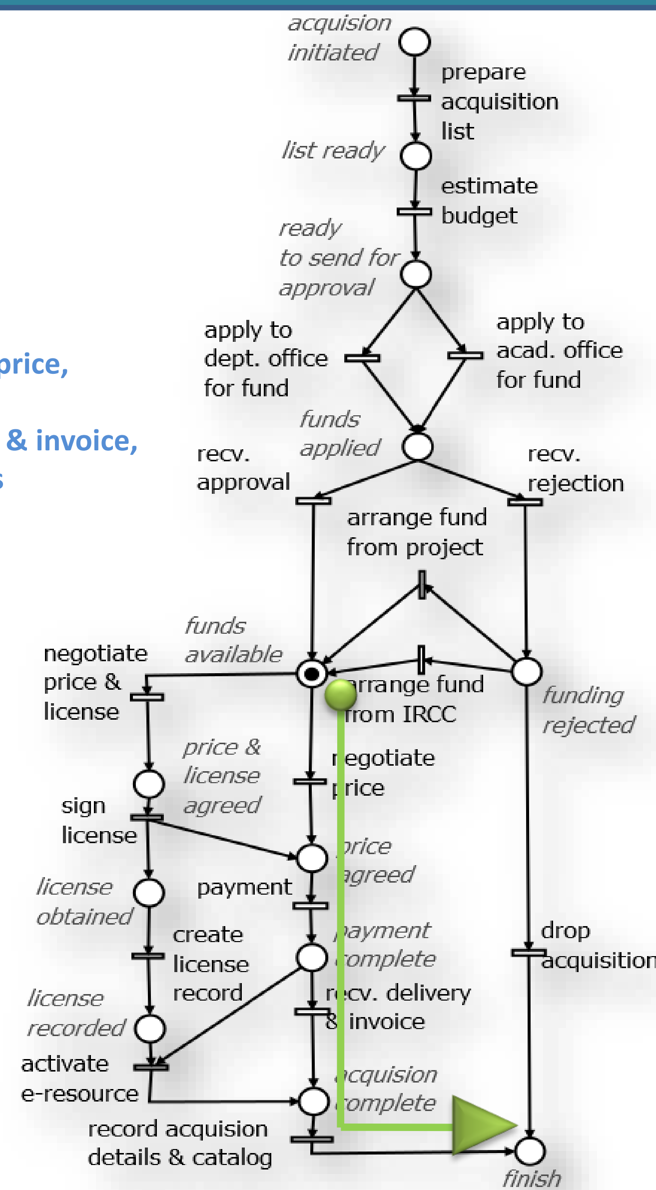
Practical Example: Resource Acquisition

Departmental Process Instance



(Algo 1)

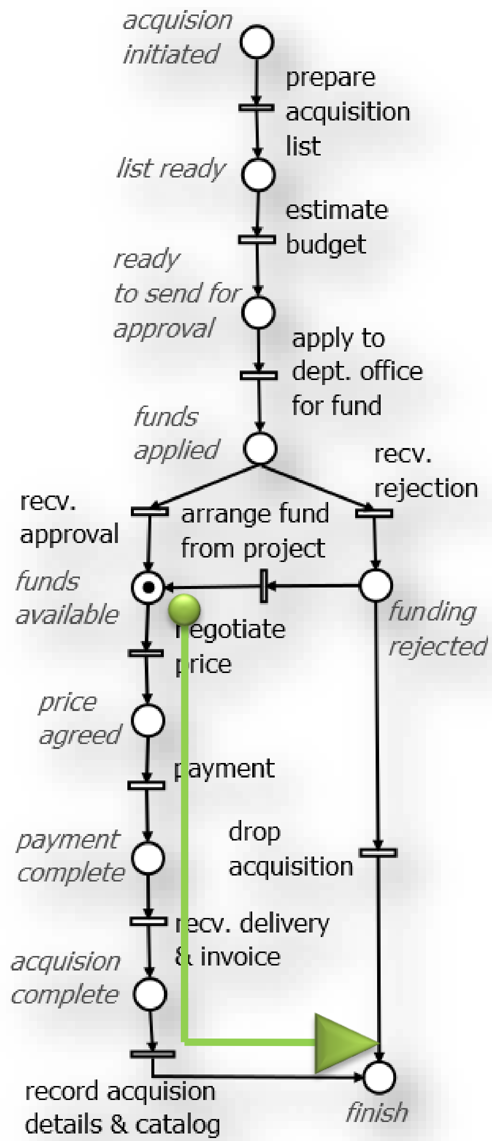
L = negotiate price,
Payment,
Recv. delivery & invoice,
Record details



Migrated Instance

Practical Example: Resource Acquisition

Departmental Process Instance



(Algo 1)

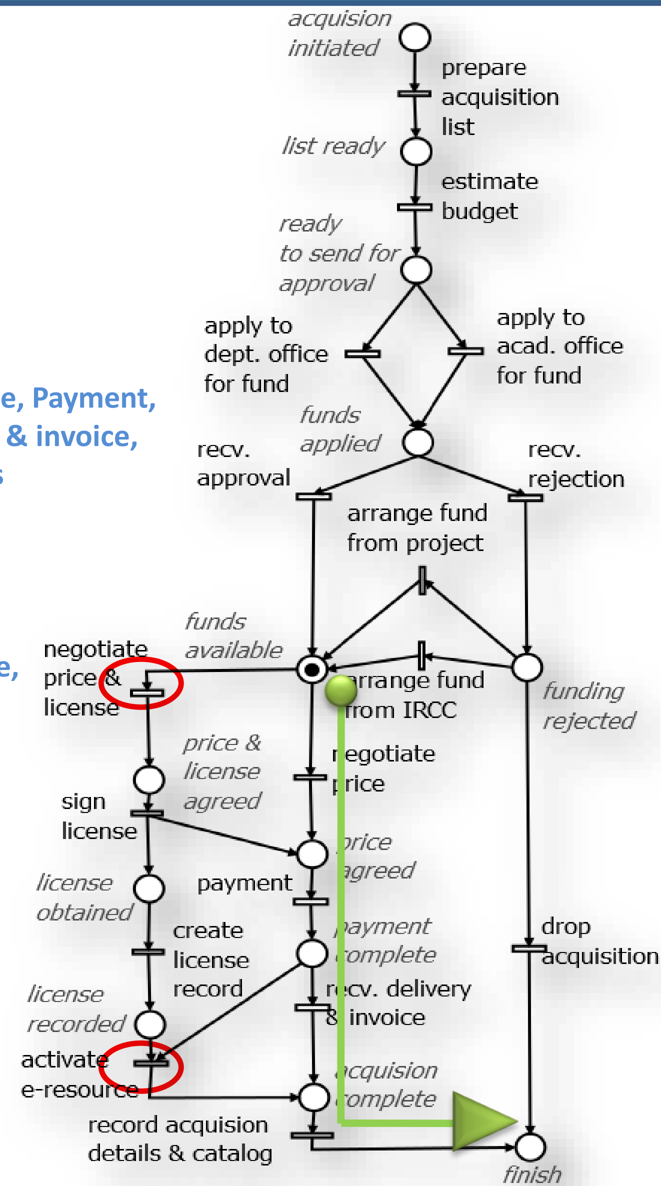
L =

negotiate price, Payment, Recv. delivery & invoice, Record details

(Algo 2)

T_{block} =

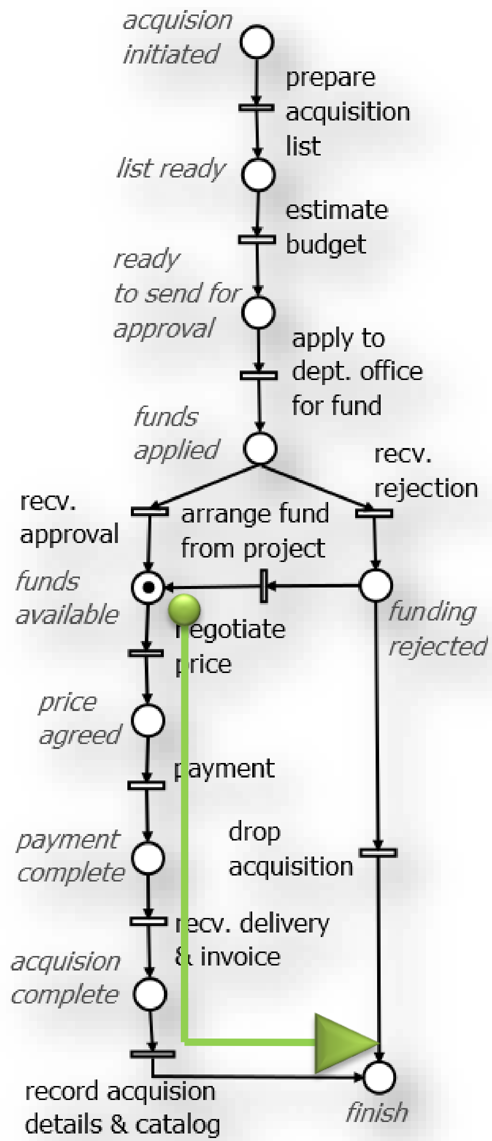
Negotiate price & license, Activate e-resource



Migrated Instance

Practical Example: Resource Acquisition

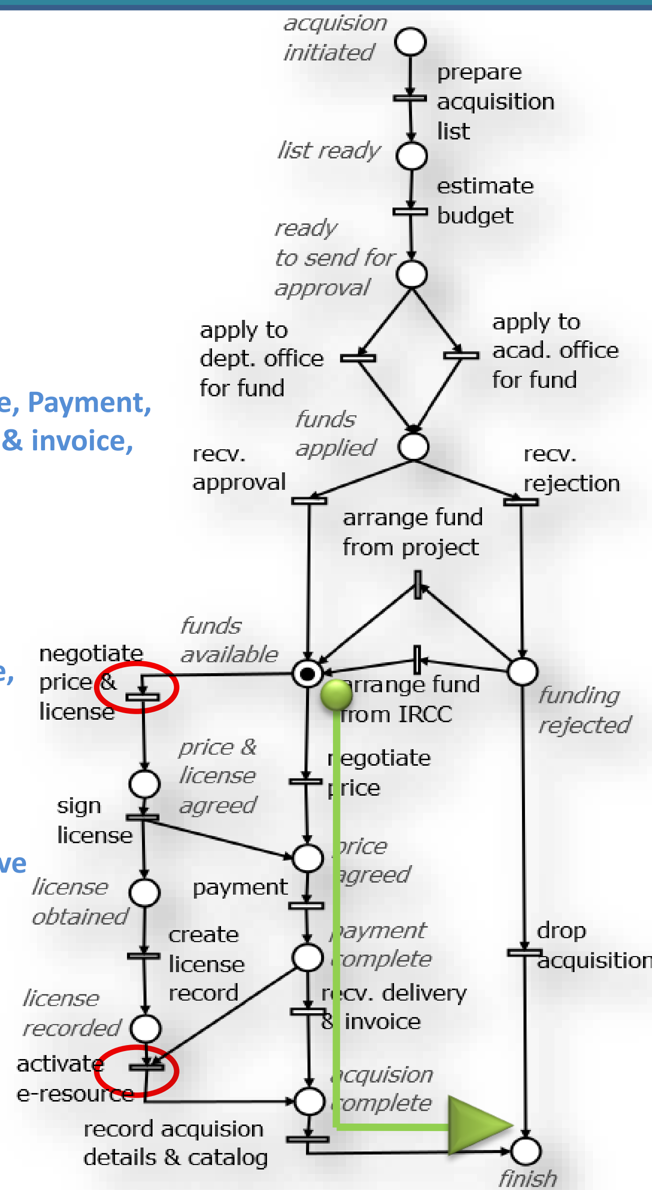
Departmental Process Instance



(Algo 1)
L =
negotiate price, Payment, Recv. delivery & invoice, Record details

(Algo 2)
T_{block} =
Negotiate price & license, Activate e-resource

Inferences:
Accommodative
Lookahead
Consistency by schema



Migrated Instance

Conclusion

New approach to the token transportation problem by Catalog based modular solution by YoYo algorithm.

Embedding history in the catalog results in history equivalent solutions without computing them in runtime.

Novel approach of derivation tree and its coloring for representing net, markings along with the hierarchy of composition.

Structural analysis pushed to the schema level and linear runtime complexity for token transportation at instance level for trace equivalent migration.

Developed lookahead trace based consistency models with varying flexibility

Demonstrated dynamic migration scenarios requiring future-based consistency notion, in contrast to trace based models

Algorithms for computing lookahead consistent markings, and inferences regarding the class of consistency

Support vs. enforcement of lookahead trace executions;
Practical migration situation requiring lookahead enforcement

Future Works

Consistency Models and Change Regions

Extending the scope of Yo-Yo Algorithm

Dynamic instance migration in distributed execution environment

Publications & Paper Presentations

1. [Full paper] Lookahead Consistency Models for Dynamic Migration of Workflow Processes : ** In Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15**), A satellite event of the conference: 36th International Conference on Application and Theory of Petri Nets and Concurrency 2015, Brussels, Belgium, pp: 267-286, June 22-23, 2015.

2. [Full paper] Catalog-based Token Transportation in Acyclic Block-Structured WF-nets : ** In Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15**), A satellite event of the conference: 36th International Conference on Application and Theory of Petri Nets and Concurrency 2015, Brussels, Belgium, pp: 287-307, June 22-23, 2015.

3. [Poster] Architecture of a light-weight non-threaded event oriented workflow engine : In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, pp: 342-345, May 26-29, 2014.

4. [Short paper] Token transportation in Petri net models of workflow patterns : In Proceedings of the 7th India Software Engineering Conference, Chennai, ISEC '14, Chennai, India, pp: 17:1-17:6, February 19-21, 2014.

THANK YOU