A Theory of Dynamic Evolution in Petri Net Models of Business Processes

Presynopsis presentation
by
Ahana Pradhan
under the guidance of
Prof. Rushikesh K. Joshi

Department of Computer Science and Engineering, IIT Bombay
August, 2016
Business Processes

- Complex Flow of activities to achieve a business goal of an Organization
- Examples:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Business Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finance</td>
<td>Billing Process</td>
</tr>
<tr>
<td>Human Resource Mgmt</td>
<td>Vacation Request/Approval Process</td>
</tr>
<tr>
<td>Banking</td>
<td>Account Opening Process</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Product Delivery Process</td>
</tr>
<tr>
<td>Travel</td>
<td>Ticket Booking Process</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Product Assembly Process</td>
</tr>
<tr>
<td>Public Service</td>
<td>Passport Application Process</td>
</tr>
<tr>
<td>Academic</td>
<td>Admission Process</td>
</tr>
</tbody>
</table>

- Activities in business process: manual tasks, user assisted automated tasks, web-services, other business processes etc.
Business Processes Modeling

Marketing trainee

1. Common Orientation Talks
2. Product-range acquaintance
3. Associate with requirement refinement team
4. Associate with Customer talks
5. Ready to participate in Project Initiation?
6. No
7. Associate with Project Initiator team
8. Common Exit Talks
9. Training Process For batch of Fresh Recruits

Design trainee

1. Common Orientation Talks
2. Product Introduction
3. Associate with requirement collection team
4. Associate with requirement collection team
5. Associate with requirement collection team
6. Associate with Model team
7. Associate with Interface Design team
8. Associate with testing team
9. Yes
10. Training Process For batch of Fresh Recruits
Dynamic Evolution in Business Processes

Requires migration of old process
Instances to the new one so that
Trainees continue in the new program
With the partially completed old training

- Design trainees need to know requirement
- Separate training for testers

Evolved Training Process
Motivations

• Various approaches exist for model specific solutions
• Different situations require different notions of correctness
• Subtle interplays among notion of correctness, available algorithms and concepts not classified so far
• Requires consolidation of theoretical approaches to move forward in the research field
• Solutions not practice since theory has not developed enough
• Challenging problems in theory

Problem Statement

• Given old and new schema, explore the problem of state-transfer under different notions of correctness
• Algorithmic solution to state-transfer that avoids state-space search
• Theoretical approach general enough to adapt in different modeling approaches
• Explore properties and proofs related to the problem
Scope and Assumptions

- Control-flow structure of business processes
- Schema structures are correct
- Original and Evolved schema are provided
- Well-formed schema

Not in Scope

Access control and user perspectives, Data-flow concerns, Methodology involved in evolution, Unstructured workflows, Deployment issues
Contributions

**Algorithms**
- YoYo algorithm for instance migration
- Algorithm for weak lookahead
- Accept/reject branching algorithm for strong lookahead
- PSCR computation algorithm
- Change region computation algorithm
- Distributed change region computation algorithm

**Properties**
- YoYo compatibility, peer patterns
- Generator of Concurrent Submarking (GCS)
- Dysfunctional C-tree and Break-off Set
- Marking Preserving Embedding (MPE)
- Change properties
- Perfect Member and Overestimation
- Perfect Structural Change Region (PSCR)
- Fragmentation

**Taxonomy Framework for Consistency Models**
- Structural equivalence
- Trail-based models
  - history equivalence
  - trace equivalence
  - purged-history equivalence
  - purged-trace equivalence
- Live model
- Lookahead models
  - strong accommodating
  - weak

**Proofs**
- Non-migratability lemma
- Perfect Member lemma
- Overestimation lemma
- SCR & PSCR lemma
- Proof of correctness for algorithms

**Workflow Specification Languages**
- CWS, ECWS

**Representation Techniques for Analysis & Application**
- C-tree, Derivation Tree
- Token transportation catalog
- Token transportation bridge

**New Consistency Models**
- Strong lookahead
- Accommodative lookahead
- Weak lookahead
Notions of Consistency

Consistency:
Formal criteria of correctness of instance migration (state-transfer/token transportation)

State:
Marking in Petri net models

Old and New markings can be considered as migration equivalent (consistent) in various ways

Past/Present/Future of a State
Taxonomy and New Models of Consistency

<table>
<thead>
<tr>
<th>Consistency Models</th>
<th>Parameters</th>
<th>Future based</th>
<th>Trace based</th>
<th>Structure based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Past based</td>
<td>Present based</td>
<td>Future based</td>
<td>Trace based</td>
</tr>
<tr>
<td></td>
<td>equal</td>
<td>subset</td>
<td>superset</td>
<td>Set based</td>
</tr>
<tr>
<td>Structural equivalence</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Trace equivalence</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>History equivalence</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Purged-trace eq.</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Purged-history eq.</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Live</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Strong lookahead</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Accommodative lookahead</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Weak lookahead</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Dynamic Instance Migration

State-transfer Approach Vs. Change Region Approach

Marking Transfers From old to new Net explicitly as per the Chosen Consistency model

Marking Transfers as it is Outside change region, No consistent migration Inside change region (no explicit state-transfer)

Old ticket booking process

New ticket booking process
Dynamic Instance Migration

Citation to the Literature | Consistency      | State-transfer       | Change Region                                                                                                                                               |
---------------------------|----------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
Ellis et al. 1995, ACM COCS | Trace Equivalence | Some marking to Some marking | Inside: non-migratable/migration to different marking
Outside: migration as it is
Construction: Not specified (intuition given for SESE region) |
Van der Aalst, 2001, Info. Sys. Frontiers (Springer) | Live Consistency | Some marking to Same marking | Inside: non-migratable/migration as it is
Outside: migration as it is
Construction: minimal SESE region covering structural changes
(modified SESE reg. black box, same state-space outside) |
Sun et al., 2009, Info. Soft. Tech. (Elsevier) | Purged History Equivalence | Some marking to Some marking | Inside: non-migratable/migration to different marking
Outside: migration as it is if every marking inside change region is migratable
Construction: minimal SESE region covering structural changes |
Cicirelli et al., 2010, J. Sys. Soft. (Elsevier) | History Equivalence | Some marking to Same marking | Van der Aalst, 2001 |
Zou et al., 2010, IEEE Advanced. Serv. Comp. | Trace Equivalence | Some marking to Some marking | Van der Aalst, 2001 |

Migration to same marking is Live Consistency

migration to different marking inside Change region violates Live consistency

Structural changes may retain state-reachability, e.g. loop to xor

For SESE change region, Migration as it is outside Change region may violate Trace/history based consistency e.g. downstream the change region
State-transfer Approach
History Equivalence Consistency Model

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

Validity
reachability of marking in the new net

Inconsistent!
Missing token in parallel branch

Invalid!
Correct

Done task tx

Done task ty
YoYo Approach

Token transportation by: Folding, transport, Unfolding

Old Net:

New Net:

Pre-computed transportation
Folding

Old Net:

New Net:
Transport & Unfolding

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Transport & Unfolding

Old Net:

New Net:

transport
Transport & Unfolding

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Transport & Unfolding

Old Net:

New Net:

No transport required
### CWS Grammar

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Specification</th>
<th>Net Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ:</td>
<td>tx ty</td>
<td>[Diagram of SEQ pattern]</td>
</tr>
<tr>
<td>AND:</td>
<td>(tx) (ty)</td>
<td>[Diagram of AND pattern]</td>
</tr>
<tr>
<td>XOR:</td>
<td>[tx] [ty]</td>
<td>[Diagram of XOR pattern]</td>
</tr>
</tbody>
</table>

**Example:**

- Presynopsis: Dynamic Evolution in Petri Net Business Processes
- 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]

Acyclic Workflow nets
Even no. of transitions

**Folding steps follow such order of derivation..**
Derivation Tree

Presynopsis: Dynamic Evolution in Petri Net Business Processes
<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf/Non-leaf</td>
<td>Unmarked folded/unfolded place</td>
</tr>
<tr>
<td>Leaf</td>
<td>marked place in net</td>
</tr>
<tr>
<td>Non-leaf</td>
<td>abstraction of null-executed subnet</td>
</tr>
<tr>
<td>Non-leaf</td>
<td>abstraction of subnets where at least one labeled transition has been fired</td>
</tr>
</tbody>
</table>

**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
YoYo Compatibility

Yield of $r_1 =$
\[ t_1 \{ t_2 \{ t_3, t_4 \}, \{ t_5, t_6 \} \} \} t_7 \} t_8 \]

Yield of $r_2 =$
\[ t_1 t_2 t_3 t_4 \{ t_5, t_6 \} t_7 t_8 \]

Both can generate the same sequence $t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 \rightarrow Folding order exists$
Folding Order

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Every possible Color-mapping Between peer patterns
YoYo Algorithm

Presynopsis: Dynamic Evolution in Petri Net Business Processes

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\( (q) \)
1. Color old tree

2. \(<p-q>\) be 1st peer patterns to appear in folding order \(F\)

3. Color transfer between \(p, q\)

4. \(\text{for}\) each next \(<p-q>\) in \(F\),
   - \(\text{if}\ \) \(q\) has colored root,
     - \(\text{if}\ \) \(p\) is colored,
       - color transfer between \(p, q\)
   - \(\text{else}\)
     - localPropagation\((q)\)
1. Color old tree
2. <p-q> be 1\textsuperscript{st} peer patterns to appear in folding order $F$
3. Color transfer between $p$, $q$
4. \textbf{for} each next $<p,q>$ in $F$,
   \hspace{1em} if $q$ has colored root, 
   \hspace{1em} if $p$ is colored, 
   \hspace{1em} color transfer between $p$, $q$
   \hspace{1em} else
   \hspace{1em} localPropagation($q$)

**Presynopsis:** Dynamic Evolution in Petri Net Business Processes
YoYo Algorithm

1. Color old tree
2. \(<p-q>\) be \(1^{st}\) 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\((q)\)
YoYo 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)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
YoYo Algorithm

Presynopsis: Dynamic Evolution in Petri Net Business Processes

Max. no. of Transportation Steps = no. of patterns (linear time complexity)
**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 guarantees 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
Lookahead Consistency Models

<table>
<thead>
<tr>
<th>Consistency Model Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Lookahead</td>
<td>same lookahead trace sets of consistent marking</td>
</tr>
<tr>
<td>Accommodative Lookahead</td>
<td>old lookahead trace set preserved in new</td>
</tr>
<tr>
<td>Weak Lookahead</td>
<td>at least one old lookahead trace preserved in new</td>
</tr>
</tbody>
</table>

**State-transfer Algorithms:**
1. Determine existence of Weak Lookahead
2. Inferences about Strong/Accommodative Lookahead
3. Given Accommodative Lookahead, enforce Strong Lookahead
Gist of State-transfer Approaches

Petri Net is not trace-accumulative model, needs some kind of trace-replay in order to obtain trace-based consistent migrations

YoYo algorithm (for existing model of consistency)
- Vertical trace-replay (through hierarchy of derivation tree) in comparison to traditional horizontal trace-replay (token game)
- Efficient due to ready-made solutions (catalog)
- Restricted scope to CWS-nets and pattern changes

Lookahead algorithms (for new models of consistency) based on trace-reply
Change Region Approach
Presynopsis: Dynamic Evolution in Petri Net Business Processes

Two consistent markings have the same set of marked places.
When available before Migration of hundreds of active instances, One can know which of them can be Migrated safely and immediately Without consulting the state space!

Presynopsis: Dynamic Evolution in Petri Net Business Processes
**Existing Approach**

Presynopsis: Dynamic Evolution in Petri Net Business Processes

Change Region:
Smallest SESE region covering Structural changes
(Reasoning: Whatever change in happens to State-space, remains confined in This region)

(a) Old Net

(b) New Net

No change in Reachable states In the state-space

False-negatives!
Our Approach

- Structural Model of all possible markings – C-tree
- Analyze causes of non-migratability – Change Properties
- When false-negatives are unavoidable? – overestimation
- Change region devoid of overestimation – PSCR
- Computation of PSCR through change properties
ECWS Grammar for Workflow Nets

Sequence
p1 t1 p2 t2 p3

Parallel (AND)
p1 t1 ( p2 t2 p3 ) ( p4 t3 p5 ) t4 p6

Choice (XOR)
p1 [ t1 ] [ t2 ] p2

Loop
{ p1 t1 p2 } { t2 }

Net \rightarrow Pnet
Pnet \rightarrow PLACE
  | Pnet TRANS PLACE
  | Pnet TRANS loop TRANS Pnet
  | Pnet TRANS and TRANS Pnet
  | Pnet xor Pnet
Tnet \rightarrow TRANS | TRANS Pnet TRANS
loop \rightarrow \{ Pnet \} \{ Pnet \}
xor \rightarrow \[ Tnet \] [ Tnet ] | [ Tnet ] xor
and \rightarrow ( Pnet ) ( Pnet ) | ( Pnet ) and
Conjoint Tree (C-tree) abstracts all markings structurally: efficient to compare the old and the new set of markings

Net Structure

C-tree Structure

Hierarchy of Nested Concurrency
1. sequential places at root
2. Concurrent places at non-root
3. C-block (□) for every AND-block
4. Children of a C-block are AND-branches
5. Recursive structure for nested-AND
6. Places (and C-blocks) in parent-child nodes are non-concurrent to each other (also places in same node are non-concurrent)
Generator of Concurrent Submarking (GCS): captures the concurrent part of the net w.r.t. a place.

- GCS is used to generate reachable markings.
- GCS is the key to identify any change happened in the Concurrency when two nets are considered.

Example:
- Initial marking: \{ p_1 \}
- Marking: \{ p_{16}, p_{15}, p_9 \}

C-tree captures all markings without the complexity of the state-space.
**Dysfunctional C-tree & Break-off Set**

If after removal of some places from a C-tree, the mutated C-tree cannot generate a marking that can be generated from the original C-tree, the mutated C-tree is called dysfunctional, and the set of places removal of which renders the tree dysfunctional is called as a break-off set for that C-tree.

Net Structure

ECWS spec

\[
p_1 t_1 \{p_2\} t_2 p_3 t_3 \} t_4 (p_4 t_5 (p_5 t_6 p_7)(p_6 t_7 p_8) t_8 p_9)(p_{10} t_9 p_{11}) t_{10} p_{12}
\]

Empty path from Root to leaf

C-tree

Dysfunctional C-tree, Break-off set: \{p_1,p_2,p_3,p_4,p_6,p_8,p_9,p_{12}\}
Marking Preserving Embedding (MPE)

Definition:
An MPE of C in C’ is a mapping from C to C’ defined recursively: (i) places(root(C)) is subset of places(root(C’)), and (ii) C-blocks in root(C) are injectively mapped to C-blocks in root(C’) such that within each C-block to C-block mapping pair (b, b’), there is a bijective MPE of the children C-trees of b to those of b’.

Non-migratability Lemma:
Given two C-trees C and C’, if an MPE of C in C’ does not exists, then at least one marking constructible from C cannot be constructed from C’. The converse is also true.
### Effect of Concurrency of a place on Migratability

<table>
<thead>
<tr>
<th>Case id</th>
<th>Concurrent in Old Net</th>
<th>Concurrent in New Net</th>
<th>Migratability</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>✓</td>
<td>✓</td>
<td>Conditionally Migratable</td>
</tr>
<tr>
<td>C2</td>
<td>✓</td>
<td>×</td>
<td>Non-migratable</td>
</tr>
<tr>
<td>C3</td>
<td>×</td>
<td>✓</td>
<td>Non-migratable</td>
</tr>
<tr>
<td>C4</td>
<td>×</td>
<td>×</td>
<td>Migratable</td>
</tr>
<tr>
<td>C5</td>
<td>Don’t care</td>
<td>Absent</td>
<td>Non-migratable</td>
</tr>
</tbody>
</table>

**Example:**

C1: \{p1,p2\} → \{p1,p2\}; \{p1,p2\} → \{p1,p2,p3\}....
C2: \{p1,p2\} → \{p1\}
C3: \{p1\} → \{p1,p2\}
C4: \{p1\} → \{p1\}
C5: \{p1\} or \{p1,p2\} → no marking with p1 available

A concurrent place has more than one marking in an ECWS-net
A non-concurrent place has only one marking in an ECWS-net
(C5) Removal
No marking involving $p$ is reachable in $N'$. ($p$ is present in $C$, $p$ is absent in $C'$).

(C2) Lost Concurrency
Markings involving $p$ are concurrent in $N$ but not in $N'$ ($p$ in a non-root node in $C$ but in the root node in $C'$).

(C3) Acquired Concurrency
Only one standalone marking involving $p$ in $N$, but concurrent markings in $N'$ ($p$ is in root node of $C$ but in a non-root node in $C'$).

(C1.1) Weak Reformed Concurrency
(i) $p$ is in concurrent markings in both $N$ and $N'$ ($p$ in non-root nodes in both $C$ and $C'$)
(ii) at least one concurrent marking involving $p$ in $N$ is not reachable in $N'$ due to addition or reduction of concurrency of $p$ ($GCS(p,C)$ does not have an MPE in $GCS(p,C')$).

(C1.2) Strong Reformed Concurrency
(i) $p$ is in concurrent markings in $N$ and $N'$ ($p$ in non-root nodes both in $C$ and $C'$)
(ii) all concurrent markings involving $p$ in $N$ are not reachable in $C'$. (either the set of places $\{\text{places}(GCS(p,C)) - \text{places}(GCS(p,C'))\}$ is a break-off set w.r.t. C-tree $GCS(p,C)$, or the set of places $\{\text{places}(GCS(p,C')) - \text{places}(GCS(p,C))\}$ is a break-off set w.r.t. C-tree $GCS(p,C')$.)

Justification: $p$ in non-root nodes implies involvement in concurrency. When condition is satisfied, it means that the places which are concurrent to $p$ in both nets are not capable of generating any common valid marking involving $p$. 
### Structural Change Region (SCR)

Given a migration net pair $N$ and $N'$, $\text{SCR}(N,N')$ is a subset of places in $N$ s.t.
for every non-migratable marking $M$ from $N$ to $N'$, $M$ includes a member from $\text{SCR}(N,N')$.

### Perfect Structural Change Region (PSCR)

Given a migration net pair $N$ and $N'$, $\text{PSCR}(N,N')$ is a subset of places in $N$ s.t.

1. for every place $p$ in $\text{PSCR}(N,N')$, there exists a non-migratable marking from $N$ to $N'$ involving $p$,
2. for every non-migratable marking $M$ from $N$ to $N'$, $M$ includes a member from $\text{PSCR}(N,N')$.

### Perfect Member

A place $p$ in the old net $N$ is a perfect member in $N$, w.r.t. the new net $N'$ iff all markings in $N$ involving $p$ are non-migratable.

### Overestimation

A place $p$ in the old net $N$ is an overestimation w.r.t. the new net $N'$, iff there exists a migratable marking and also a non-migratable marking involving $p$ in $N$.

### Safe Member

A place $p$ in the old net $N$ is a safe member w.r.t. the new net $N'$, iff every marking involving $p$ in $N$ is migratable.
## Change Region Lemmas

<table>
<thead>
<tr>
<th>Lemma</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SCR Lemma</strong></td>
<td>The union of all overestimations and all perfect members in old net N w.r.t. new net N’ is SCR in N w.r.t. N’.</td>
</tr>
<tr>
<td><strong>PSCR Lemma</strong></td>
<td>PSCR exists in a given old net N w.r.t. new net N’ iff every non-migratable marking in N includes at least one perfect member. (the proof constructs the set of perfect members as the PSCR)</td>
</tr>
<tr>
<td><strong>Perfect Member Lemma</strong></td>
<td>If a place p in old net N satisfies one of Removal, Lost Concurrency, Acquired Concurrency and Strong Reformed Concurrency w.r.t. new net N’, then the place is a perfect member and vice-versa.</td>
</tr>
<tr>
<td><strong>Overestimation Lemma</strong></td>
<td>If a place p in old net N satisfies Weak Reformed Concurrency but not Strong Reformed Concurrency w.r.t. new net N’, it is an overestimation w.r.t. N’ and the vice-versa.</td>
</tr>
</tbody>
</table>
Computation of PSCR

Old C-tree $C$, New C-tree $C'$, set of Perfect Members in $C$ w.r.t. $C'$ be $\text{Perf}$.
If there is no overestimation in $C$, $\text{PSCR} \leftarrow \text{Perf}$.
Else if PSCR exists as per the following table, $\text{PSCR} \leftarrow \text{Perf}$.

<table>
<thead>
<tr>
<th>C has markings without Perfect Members</th>
<th>$C'$ has markings without Perfect Members</th>
<th>All markings without Perfect Members in C can be generated from $C'$</th>
<th>PSCR Exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>× Perf is break-off set for $C$</td>
<td>Don't care</td>
<td>Don't care</td>
<td>✓</td>
</tr>
<tr>
<td>✓ Perf is not break-off set for $C$</td>
<td>× Perf is break-off set for $C'$</td>
<td>Don't care</td>
<td>×</td>
</tr>
<tr>
<td>✓ Perf is not break-off set for $C$</td>
<td>✓ Perf is not break-off set for $C'$</td>
<td>× delete($C$, Perf) doesn't have MPE in delete($C'$, Perf)</td>
<td>×</td>
</tr>
<tr>
<td>✓ Perf is not break-off set for $C$</td>
<td>✓ Perf is not break-off set for $C'$</td>
<td>✓ delete($C$, Perf) has a MPE in delete($C'$, Perf)</td>
<td>✓</td>
</tr>
</tbody>
</table>
Experimental Results

Presynopsis: Dynamic Evolution in Petri Net Business Processes

Training Process

Hypothetical Process

Claims Process
Distributed Business Processes

- Business processes often cross departments, organizations
- Distributed deployments lack centralized view
- Individual process-fragments may evolve independently
- Individual change regions can be globally conflicting
Fragmented Net has no Global View

C-tree nodes are split, but fan-out of C-blocks doesn’t change

Places with one pre- and post-transitions in the global net are Eligible to be boundaries

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Inspection of Places
1. \{ p3 \} \rightarrow \text{no marking with p3 available (deleted)}
2. \{ p1 \} \rightarrow \{ p1, _ \} (\text{root} \rightarrow \text{non-root})
3. \{ p2, _ \} \rightarrow \{ p2 \} (\text{non-root} \rightarrow \text{root})
4. \{ q3, _ \} \rightarrow \{ q3, _ \}, \text{but \{q3, p2\} not available (places of old GCS missing in new)}
5. \{ q8, _ \} \rightarrow \{ q8, _ , _ \} (\text{new branches in GCS})
Effect of Fragmentation

Assumptions
A place does not jump to another fragment (root $\leftrightarrow$ non-root locally visible)
No boundary place becomes internal
Deletion of boundary is consistent between peer fragments

(place deletion locally visible)
Conflict resolution between fragments using boundaries

Global View

Remaining C-tree

p1 p2 p3 p4 p5 p6 p7...

Old Net

Changed Remaining C-tree

p1 p2 p3 p4 p5 p6 p7...

New Net

Changed Remaining C-tree

p1 p2 p3 p4 p5 p6 p7...

Fragmented View

Detects GCS change

GCS change is same for p1, p2, p3, p4, p5, p6, p7, ... since they are together before and after the change

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Fragmented Process: Employee Transfer

Only Accounts and Reporting Section Receives change spec.
The Distributed Algorithm using Asynchronous Events: Hierarchical CPN based description

EVENT TYPES → Color definitions. Event parameters → values in tokens

Each module runs once

Modules run in loop until Termination condition reached

INITIATE → SAFE_B → NOCHANGE → CHANGE
Event Substrate: example for EVOLVE event type

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Initiation Round (Module 1)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Initiation Round (Module 1)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Initial Change Region, Broadcast Boundary Notifications (Module 2)

**SAFE_B(...)**

**CR_B(...)**

---

Can be empty set

**SAFE_B(<set of places>)**

**CR_B(<set of places>)**

---

**SAFE_B** is the list of boundary places in the old fragment.

**CR_B** is constant value, the list of boundary places in the old fragment.

---

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Receive Boundary Notifications (Module 3)

CR_B (p10) ➔ SAFE_B (p4) ➔ CR_B ()
SAFE_B (p12, p16)

CR_B (p10) ➔ SAFE_B (p4) ➔ CR_B ()

Records remote-status of Local boundaries, ignores other Incoming parameters

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Initial Change Region, Broadcast Boundary Notifications, Receive Boundary Notifications (Modules 2+3)

Boundary p16, p12
SAFE_ext p12 p16
CR_ext

Boundary p4, p18, p12
CR_ext
SAFE_ext p12 p18 p4

Conflict! (locally safe, remotely unsafe)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Conflict Resolution (Module 4)

Presynopsis: Dynamic Evolution in Petri Net Business Processes

If a safe boundary & other relevant places are made unsafe, CHANGE event is published with additional boundaries that became unsafe.
Termination (Module 5)

Local boundaries arriving with CHANGE event are put in set CR_ext to investigate further conflict in next round of Module 4 (loop).

When everybody sent NOCHANGE, and the node itself sent NOCHANGE in Module 4, Algorithm terminates.

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Conflict Resolution & Termination (Module 4+5) (iteration 1)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Conflict Resolution & Termination (Module 4+5) (iteration 2)

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Result

Presynopsis: Dynamic Evolution in Petri Net Business Processes
Lemma 1
The effect of deletion of a place in the change region in a fragment is fully covered by the fragment in which the place is deleted.

Lemma 2
If there is any change in concurrency of a place in the global net, some fragment detects it.

Bounded Wait

Termination
Gist of Change Region Approach

- Focus on Marking Reachability (effect of changes) rather than structural changes
- Structural Modeling of markings through C-tree
- Capturing non-migratability through properties
- Approach restricted to structured nets (ECWS grammar)
  structured AND is mandatory for C-tree construction, unstructured XOR, LOOP can be handled in C-tree since they are all sequential (w.r.t. token game) regions. Specification through balanced parenthesis is convenient for programming.
- Centralized and Distributed Computation algorithms developed
Future Work

• Distributed instance migration
• Interplay among consistency models and change operations
• Extending the theory for unstructured workflows
• Implementation Issues
Presented Works:


Unpublished Works:

- **A Structural Approach to Dynamic Migration in Petri Net Models of Structured Workflows** by Ahana Pradhan and Rushikesh K. Joshi [under review in IEEE TSE]

- **A Survey of Consistency Models for Dynamic Workflow Migration** by Ahana Pradhan and Rushikesh K. Joshi [in preparation]
Thank You
Outcomes

**Algorithms**
- YoYo algorithm for instance migration
- Algorithm for weak lookahead
- Accept/reject branching algorithm for strong lookahead
- PSCR computation algorithm
- Change region computation algorithm
- Distributed change region computation algorithm

**Properties**
- YoYo compatibility, peer patterns
- Generator of Concurrent Submarking (GCS)
- Dysfunctional C-tree and Break-off Set
- Marking Preserving Embedding (MPE)
- Change properties
- Perfect Member and Overestimation
- Perfect Structural Change Region (PSCR)
- Fragmentation

**Taxonomy Framework for Consistency Models**
- Structural equivalence
- Trail-based models
  - history equivalence
  - trace equivalence
  - purged-history equivalence
  - purged-trace equivalence
- Live model
- Lookahead models
  - strong
  - accommodative
  - weak

**Proofs**
- Non-migratability lemma
- Perfect Member lemma
- Overestimation lemma
- SCR & PSCR lemma
- Proof of correctness for algorithms

**Workflow Specification Languages**
- CWS, ECWS

**Representation Techniques for Analysis & Application**
- C-tree, Derivation Tree
- Token transportation catalog
- Token transportation bridge

**New Consistency Models**
- Strong lookahead
- Accommodative lookahead
- Weak lookahead