



Process Model

[Process Model](#)

[Transistion Systems](#)

[Petri Nets](#)

[Workflow Nets](#)

[YAWL](#)

[Business Process Modeling Notation \(BPMN\)](#)

[Event-Driven Process Chain \(EPCs\)](#)

[Casual Nets](#)

[Process Trees](#)

Process Model

▼ Transistion Systems

1. Định nghĩa

- Là hệ thống kí hiệu cơ bản nhất dùng trong việc biểu diễn các process model
- Hệ thống được biểu diễn bằng 3 chữ cái

$$TS = (S, A, T)$$

- Trong đó

- S : tập hợp các states
- $A \subseteq \mathcal{A}$: tập hợp các actions (activities)
- $T \subseteq S \times A \times S$: tập hợp các transitions

- Trạng thái S^{start} và S^{end} cũng nằm trong set S
- Về nguyên tắc thì S có thể vô hạn, nhưng thực tế hay như ảnh dưới đây, S lại là hữu hạn, hay còn được gọi là - Finite-State Machine (FSM)

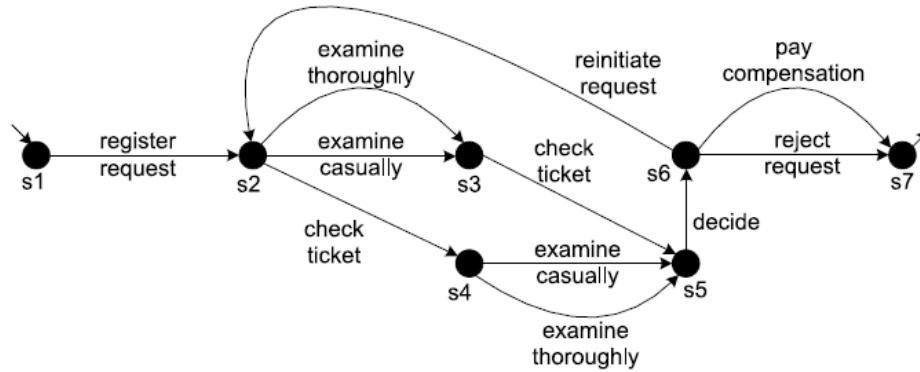


Fig. 3.1 A transition system having one initial state and one final state

The transition system depicted in Fig. 3.1 can be formalized as follows: $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, $S^{start} = \{s_1\}$, $S^{end} = \{s_7\}$, $A = \{\text{register request}, \text{examine thoroughly}, \text{examine casually}, \text{check ticket}, \text{decide}, \text{reinitiate request}, \text{reject request}\}$

request, pay compensation}, and $T = \{(s_1, \text{register request}, s_2), (s_2, \text{examine casually}, s_3), (s_2, \text{examine thoroughly}, s_3), (s_2, \text{check ticket}, s_4), (s_3, \text{check ticket}, s_5), (s_4, \text{examine casually}, s_5), (s_4, \text{examine thoroughly}, s_5), (s_5, \text{decide}, s_6), (s_6, \text{reinitiate request}, s_2), (s_6, \text{pay compensation}, s_7), (s_6, \text{reject request}, s_7)\}$.

- Một đường terminate thành công khi nó kết thúc ở 1 trong những state cuối cùng
- **Deadlock:** Là khi 1 đường đi đến non-final state mà không có transition thoát ra, dẫn đến việc hệ thống bị kẹt cứng
- Việc không nhìn thấy deadlock trên đồ thị không đảm bảo việc chuỗi kết thúc thành công → **Livelock:** có nghĩa là vẫn có các transitions nhưng không có đường nào dẫn tới trạng thái kết thúc

2. Behavioral Equivalence (Sự tương đương về hành vi)

- Tất cả process model có executable semantics đều có thể ánh xạ vào 1 transition system
- Executable Semantics: đề cập một process model không chỉ là 1 biểu diễn tĩnh mà còn có thể tạo các hành vi (behaviors) cụ thể, ví dụ như các chuỗi sự kiện (execution sequences) hay các trạng thái (states),...
- Ví dụ:
 - Petri Net : places và transitions

- BPMN : activities và gateway
- UML activity diagrams: nodes và edges (cạnh)
- ...
- Câu hỏi "Khi nào hai quy trình được coi là giống nhau từ góc độ hành vi?"
 - Có nhiều cách để định nghĩa sự tương đương này, tùy thuộc vào mức độ chi tiết mà chúng ta quan tâm:
 - Trace Equivalence (Tương đương về chuỗi thực thi)
 - Branching Bisimilarity (Tương đương phân nhánh) (moment of choice)

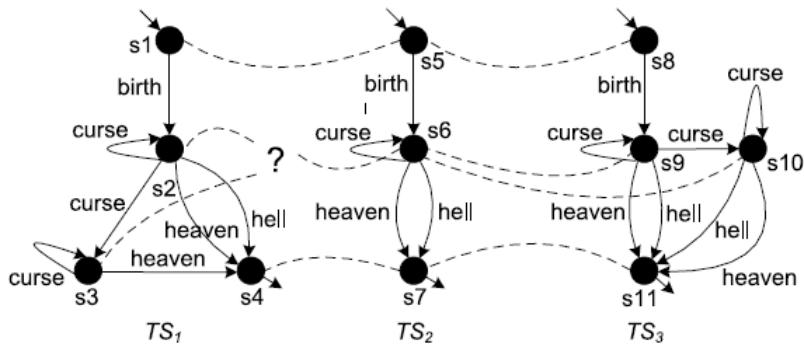


Fig. 6.18 Three trace equivalent transition systems: TS_1 and TS_2 are not bisimilar, but TS_2 and TS_3 are bisimilar

3. Tính toán

- Transition system đơn giản nhưng lại gặp vấn đề trong việc diễn đạt sự đồng thời
- Giả sử có n hoạt động song song, và tất cả n hoạt động cần thực thi và theo thứ tự bất kì
 - Vậy sẽ có $n!$ execution sequences khả thi
 - Với transition system yêu cầu 2^n trạng thái và $n * 2^{n-1}$ transitions
- Ví dụ: Giả sử có 10 hoạt động chạy song song
 - Số lượng chuỗi khả thi là $10! = 3,628,800$
 - Đồng thời, số trạng thái và số transition cần là 1024 và 5120
 - Trong khi đó, với mạng Petri tương ứng, cũng chỉ cần 10 places và 10 transitions

▼ Petri Nets

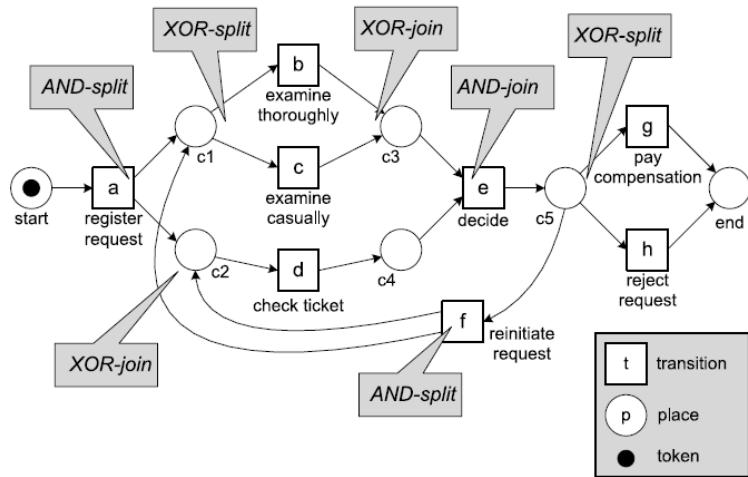


Fig. 3.2 A marked Petri net

1. Định nghĩa

- Là ngôn ngữ mô hình hóa quy trình lâu đài nhất và cho phép việc mô hình hóa đồng thời.
- Hệ thống được biểu diễn bằng 3 chữ cái

$$N = (P, T, F)$$

◦ Trong đó:

- P : tập hợp các places
- T : tập hợp hữu hạn các transition ($P \cap T = \emptyset$)
- $F \subseteq (P \times T) \cup (T \times P)$: tập hợp các cung có hướng, hay còn gọi là flow relation

The Petri net shown Fig. 3.2 can be formalized as follows: $P = \{start, c1, c2, c3, c4, c5, end\}$, $T = \{a, b, c, d, e, f, g, h\}$, and $F = \{(start, a), (a, c1), (a, c2), (c1, b), (c1, c), (c2, d), (b, c3), (c, c3), (d, c4), (c3, e), (c4, e), (e, c5), (c5, f), (f, c1), (f, c2), (c5, g), (g, end), (h, end)\}$.

Marked Petri net là 1 cặp (N, M) hay được biểu diễn là \mathcal{N} , trong đó

- $N = (P, T, F)$ là Petri net
- $M \in \mathbb{B}(P)$ là multi-set trên tập P
 - M là multi-set (Bag) biểu diễn marking (trạng thái của hệ thống), tức là lượng token trong từng place

$$M = \{c_1^2, c_2^3\}$$

- Ở ví dụ này, place c1 có 2 token và c2 có 3 token

▼ **Multi-set**

- Hay còn gọi là Bag
- Là một tập hợp mà các phần tử trong đó có thể xuất hiện nhiều lần

A multi-set (also referred to as *bag*) is like a set in which each element may occur multiple times. For example, $[a, b^2, c^3, d^2, e]$ is the multi-set with nine elements: one a , two b 's, three c 's, two d 's, and one e . The following three multi-set are identical: $[a, b, b, c^3, d, d, e]$, $[e, d^2, c^3, b^2, a]$, and $[a, b^2, c^3, d^2, e]$. Only the number of occurrences of each value matters, not the order. Formally, $\mathbb{B}(D) = D \rightarrow \mathbb{N}$ is the set of multi-sets (bags) over a finite domain D , i.e., $X \in \mathbb{B}(D)$ is a multi-set, where for each $d \in D$, $X(d)$ denotes the number of times d is included in the multi-set. For example, if $X = [a, b^2, c^3]$, then $X(b) = 2$ and $X(e) = 0$.

The sum of two multi-sets ($X \uplus Y$), the difference ($X \setminus Y$), the presence of an element in a multi-set ($x \in X$), and the notion of subset ($X \leq Y$) are defined in a straightforward way. For example, $[a, b^2, c^3, d] \uplus [c^3, d, e^2, f^3] = [a, b^2, c^6, d^2, e^2, f^3]$ and $[a, b] \leq [a, b^3, c]$. Moreover, we can also apply these operators to sets, where we assume that a set is a multi-set in which every element occurs exactly once. For example, $[a, b^2] \uplus \{b, c\} = [a, b^3, c]$.

The operators are also robust with respect to the domains of the multi-sets, i.e., even if X and Y are defined on different domains, $X \uplus Y$, $X \setminus Y$, and $X \leq Y$ are defined properly by extending the domain whenever needed.

2. Firing rule

- Gọi (N, M) là 1 marked Petri net với $N = (P, T, F)$ và $M \in \mathbb{B}(P)$
 - Transition $t \in T$ sẽ được "bật", kí hiệu là $(N, M)[t]$ chỉ khi $\bullet t \leq M$
 - Từ đó

$$(N, M)[t] \Rightarrow (N, M)[t](N, (M \setminus \bullet t) \uplus t\bullet)$$

- Ví dụ:

$$(N, [start])[a](N, [c1, c2]); (N, [c3, c4])[e](N, [c5])$$

- Chuỗi firing σ của (N, M_0) có dạng

$$(N, M_i)[t_{i+1}] \Rightarrow (N, M_i)[t_{i+1}](N, (M_{i+1}))$$

Trong đó:

$$◦ n \in \mathbb{N}; 0 \leq i < n$$

3. Labeled Petri Net

- Labeled Petri Net được biểu diễn bằng 1 tuple $N = (P, T, F, A, l)$
 - Trong đó:
 - (P, T, F) : Petri net
 - $A \subseteq \mathcal{A}$: là tập hợp activity labels

- $l \in T \rightarrow A$: là hàm gán nhãn
- Theo nguyên tắc, nhiều transitions khác nhau có thể có cùng nhãn
- Nhãn bình thường thì gọi là *observable action*
- Đối với nhãn *invisible* hay *silent* thì 1 transition t với nhãn $l(t) = \tau$ gọi là *unobservable*
- Có thể biến đổi Petri net thông thường thành labeled Petri net bằng cách cho $A = T, l(t) = t$
- Nhưng việc biến đổi ngược lại có thể lại không được vì khi nhiều transition có chung nhãn thì sẽ không thể biết được đâu là transition gốc
- Và có thể đổi một Labeled Petri Net đã được đánh dấu (marked) thành một transition system

4. Reachability Graph

- Gọi (N, M_0) với $N = (P, T, F, A, l)$ là marked labeled petri net
- (N, M_0) định nghĩa 1 hệ thống transition $TS = (S, A', T')$
 - Trong đó:
 - $S = [N, M_0], S^{start} = \{M_0\}, A' = A$
 - $T' = \{(M, l(t), M') \in S \times A \times S \mid \exists_{t \in T} (N, M)[t](N, M')\}$.

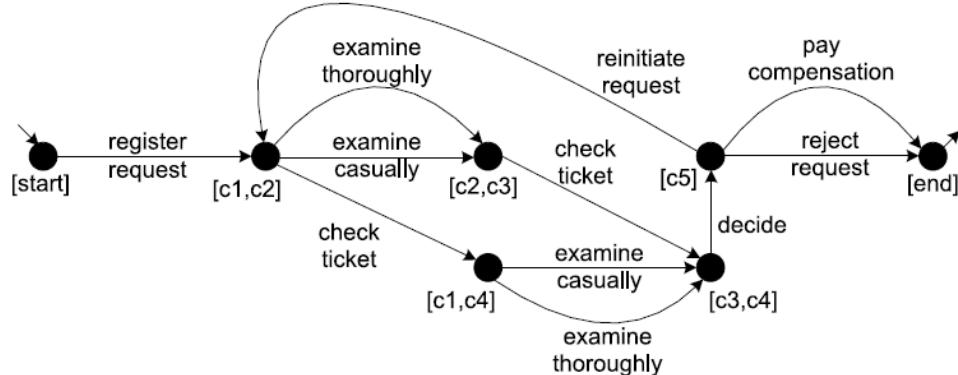
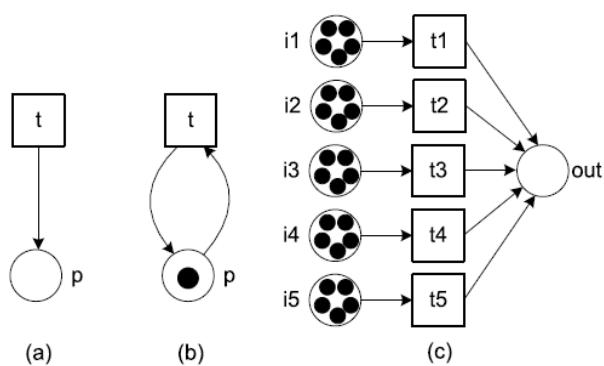


Fig. 3.3 The reachability graph of the marked Petri net shown in Fig. 3.2

Fig. 3.4 Three Petri nets:
 (a) a Petri net with an infinite state space, (b) a Petri net
 with only one reachable marking, (c) a Petri net with
 7776 reachable markings



▼ Giải thích

1. Hình a)

- **Cấu trúc:** Chỉ có một place p và một transition t.
- **Vấn đề:** Transition t không có **input place**, nhưng lại có một output place p.
- **Hậu quả:** Vì không có input, t có thể **luôn luôn kích hoạt (fire)**, và mỗi lần fire nó sẽ thêm một token vào p.
- **Kết quả:** Số lượng token trong p có thể là 0,1,2,3,..., tức là một tập vô hạn các marking. Vì vậy, Petri net này có số trạng thái vô hạn.

2. Hình b)

- **Cấu trúc:** Cũng chỉ có một place p và một transition t, nhưng lần này nó bắt đầu với một token trong p.
- **Vấn đề:** Transition t có cả input và output cùng là p.
- **Hậu quả:** Khi t kích hoạt, nó **lấy đi** một token từ p nhưng ngay lập tức lại **trả token đó về** p. Do đó, marking không thay đổi, luôn luôn là [p].
- **Kết quả:** Petri net này **chỉ có một trạng thái duy nhất**, dù transition có thể fire vô hạn lần.

4. Ảnh liên quan

▼ Ảnh

- A marked Petri net (N, M_0) is *k*-*bounded* if no place ever holds more than k tokens. Formally, for any $p \in P$ and any $M \in [N, M_0]$: $M(p) \leq k$. The marked Petri net in Fig. 3.4(c) is 25-bounded because in none of the 7776 reachable markings there is a place with more than 25 tokens. It is not 24-bounded, because in the final marking place *out* contains 25 tokens.
- A marked Petri net is *safe* if and only if it is 1-bounded. The marked Petri net shown in Fig. 3.2 is safe because in each of the seven reachable markings there is no place holding multiple tokens.
- A marked Petri net is *bounded* if and only if there exists a $k \in \mathbb{N}$ such that it is k -bounded. Figure 3.4(a) shows an unbounded net. The two other marked Petri nets in Fig. 3.4 (i.e., (b) and (c)) are bounded.
- A marked Petri net (N, M_0) is *deadlock free* if at every reachable marking at least one transition is enabled. Formally, for any $M \in [N, M_0]$ there exists a transition $t \in T$ such that $(N, M)[t]$. Figure 3.4(c) shows a net that is not deadlock free because at marking $[out]^{25}$ no transition is enabled. The two other marked Petri nets in Fig. 3.4 are deadlock free.
- A transition $t \in T$ in a marked Petri net (N, M_0) is *live* if from every reachable marking it is possible to enable t . Formally, for any $M \in [N, M_0]$ there exists a marking $M' \in [N, M]$ such that $(N, M')[t]$. A marked Petri net is live if each of its transitions is live. Note that a deadlock-free Petri net does not need to be live. For example, merge the nets (b) and (c) in Fig. 3.4 into one marked Petri net. The resulting net is deadlock free, but not live.

5. Tính chất của Marked Petri Net

- Một marked Petri net (N, M_0) là *k*-*bounded* nếu không tồn tại place nào chứa nhiều hơn k tokens
- Một marked Petri net (N, M_0) là *safe* nếu mỗi place là 1-bounded
- Một marked Petri net (N, M_0) là *bounded* nếu tồn tại $k \in \mathbb{N}$ rào số token trong mỗi place
- Một marked Petri net (N, M_0) là *deadlock free* nếu giữa mọi marking tồn tại 1 transition kết nối
- Một transition t trong một marked Petri net (N, M_0) là *live* nếu từ mọi marking có thể đến được, luôn yêu cầu phải sử dụng transition t

▼ Workflow Nets

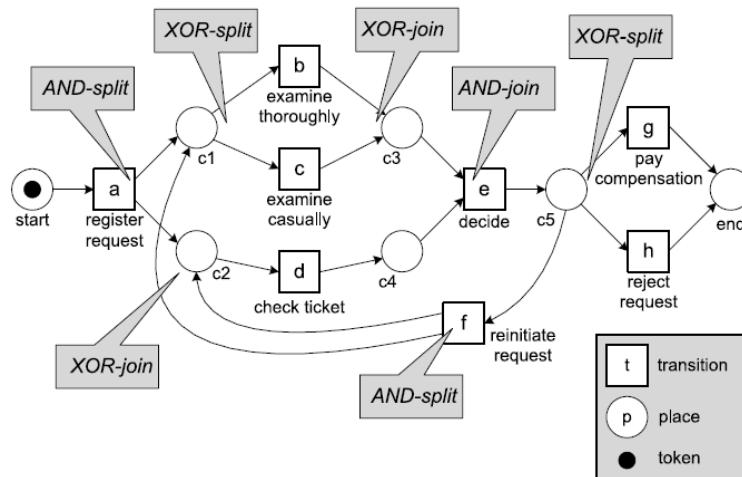


Fig. 3.2 A marked Petri net

1. Định nghĩa

Definition 3.6 (Workflow net) Let $N = (P, T, F, A, l)$ be a (labeled) Petri net and \bar{t} a fresh identifier not in $P \cup T$. N is a *workflow net* (WF-net) if and only if (a) P contains an input place i (also called source place) such that $\bullet i = \emptyset$, (b) P contains an output place o (also called sink place) such that $o\bullet = \emptyset$, and (c) $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(\bar{t}, \tau)\})$ is strongly connected, i.e., there is a directed path between any pair of nodes in \bar{N} .

- Là lớp con của Petri net và được thêm workflow-specific constraints, hay còn được gọi là WF-net
- Được thiết kế cho business process modeling
- Phải được đảm bảo mỗi process bao gồm:
 - 1 place bắt đầu
 - 1 place kết thúc
 - Kết nối phù hợp: nghĩa là mọi transition và place đều hướng đến đích end cuối cùng
- WF-net sẽ được coi là sound nếu như
 - (*safeness*) $(N, [i])$ is safe, i.e., places cannot hold multiple tokens at the same time;
 - (*proper completion*) for any marking $M \in [N, [i]]$, $o \in M$ implies $M = [o]$;
 - (*option to complete*) for any marking $M \in [N, [i]]$, $[o] \in [N, M]$; and
 - (*absence of dead parts*) $(N, [i])$ contains no dead transitions (i.e., for any $t \in T$, there is a firing sequence enabling t).

2. Lý do dùng WF-net

- Petri net là 1 model toán học chung để miêu tả hệ thống đồng thời và phân tán
- Tuy nhiên trong các model business process, thì cần đảm bảo
 - Mỗi process có các điểm bắt đầu và điểm kết thúc hợp lý
 - Không có deadlock và các task vô kết nối
 - Mỗi process khởi tạo theo 1 cách vận hành theo cấu trúc và có ý nghĩa

3. Bảng so sánh

Feature	Standard Petri Net	WF-net
General vs. Structured	Can represent any system, even non-business-related ones	Specifically designed for workflow processes
Start and End Conditions	May have multiple entry/exit points	Has one unique start and one unique end

Deadlocks & Soundness	May contain deadlocks or infinite loops	Designed to be sound (i.e., every case completes)
Concurrency & Synchronization	Supports concurrency but can be unstructured	Ensures proper synchronization and case handling
Process Mining Compatibility	Harder to apply mining algorithms directly	Well-suited for process mining due to structured execution paths

▼ YAWL

1. Định nghĩa

- Yet Another Workflow Language
- Mục tiêu: phục vụ trực tiếp cho nhiều khuôn mẫu nhưng vẫn giữ cho ngôn ngữ được đơn giản
- Các Activities trong YAWL gọi là *tasks*
- Các Places trong PN còn với YAWL là *conditions*
- Các task ở trong YAWL có thể là *atomic* hay *composite*

▼ Atomic và Composite

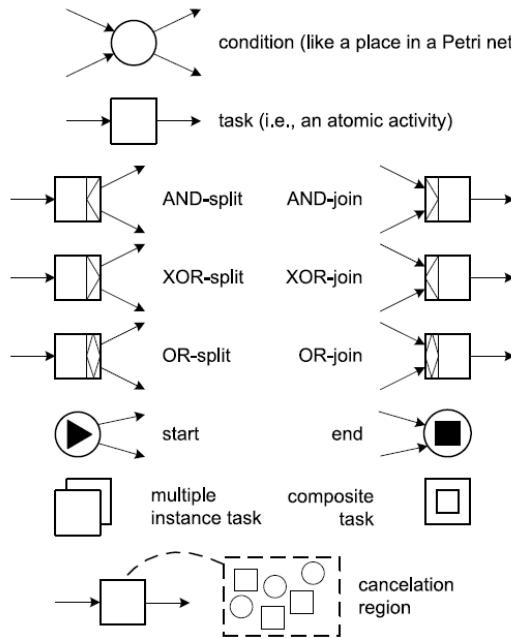
1. Atomic

- Là 1 đơn vị công việc duy nhất và không thể bị chia nhỏ
- No đại diện cho các task trong workflow
- Ví dụ:
"Approve Leave Request" in an
HR system

2. Composite

- Là task cấp cao bao gồm các workflow con (1 model YAWL con)
- Nó cho phép cấu trúc hóa workflow theo thứ bậc
- Ví dụ: "Process Loan Application" might be a **composite task** because it consists of:
 - "Check Credit Score"
 - "Verify Documents"
 - "Approve/Reject Loan"

2. Notation



▼ AND-join / AND-split

- Có hành vi như 1 transition
- Nó cần tiêu thụ 1 token thông qua mỗi cung đến và sinh ra 1 token cho mỗi cung đầu ra

1. AND-split (Parallel split)

- Kích hoạt tất cả cung đi ra cùng lúc, nghĩa là nó tạo các đường đi song song
- Ví dụ: khi nhận được 1 order, hệ thống phải làm cả 3 nhiệm vụ cùng lúc

Received Order → (AND-split) → {Pack Order, Generate Invoice, Send Email}

2. AND-join (Synchronization)

- AND-join sẽ đợi tất cả tokens từ tất cả các hướng rồi mới xử lý (đồng bộ hóa)
- Nó đảm bảo nhiệm vụ song song đã hoàn thành thì mới chuyển tiếp
- Ví dụ: Sau khi thực hiện 3 hành động cùng lúc thì

{Pack Order, Generate Invoice, Send Email} → (AND-join)
→ {Ship Order}

- Hành động ship hàng sẽ không thực hiện cho đến khi cả 3 hành động kia xong.

▼ XOR-join / XOR-split

1. XOR-split (Exclusive choice)

- Chọn lựa chính xác 1 token cho cung đầu ra
- Việc ra lựa chọn phụ thuộc vào việc đánh giá tình trạng data
- Ví dụ: Trong các trường hợp yêu cầu bồi thường bảo hiểm, nếu số tiền quá 500, thì sẽ được tự động chấp thuận còn không thì phải đợi xem xét

Submit Claim →

(XOR-split: if claim < 500 → Auto-Approve, else → Manual Review)

2. XOR-join (Simple Merge)

- Nó sẽ không đợi tất cả các cung tới và sẽ bắt đầu ngay khi có 1 token đến trước và sẽ không cần đồng bộ hóa
- Ví dụ: Nếu mà yêu cầu bồi thường chưa biết là dc chấp thuận hay cần xem xét, thì nó vẫn sẽ ra quyết định

(Auto-Approve) → XOR-Join → Final Decision

(Manual Review) → XOR-Join → Final Decision

- Nếu khách hàng chọn số tiền < 500 thì sẽ là auto-approve, XOR-join sẽ nhận token từ hướng đó và ra quyết định chứ không chờ token từ hướng còn lại

▼ OR-join / OR-split

1. OR-split (Conditional Forking, Multiple Paths Allowed) (Phân nhánh có điều kiện,...)

- Chọn 1 hoặc nhiều token cho cung đầu ra phụ thuộc vào việc đánh giá điều kiện,
- Không giống như XOR-split (chọn đúng 1 đường), OR-split sẽ chọn nhiều đường hơn
- Ví dụ: Khi khám bệnh, bác sĩ có thể chọn hình thức khám cho bệnh nhân

Patient Checkup → (OR-Split: Do casual exam, thorough exam, or both)

2. OR-join (Complex Synchronization)

- Yêu cầu ít nhất 1 input token và đồng thời cũng phải đồng bộ hóa khi đi theo cung đến hướng tới OR-join
- Khác với AND-join khi cần tất cả cung vào phải có token, OR-join chỉ cần ít nhất 1 token thôi và sẽ xem có token nào đang tới thì chờ (đồng bộ hóa)

- Ví dụ:

{Casual Exam, Thorough Exam} → OR-Join → Final Consultation

- Nếu chỉ chọn theo 1 hướng, nghĩa là khám theo 1 kiểu thì sẽ ra kết quả luôn
- Còn nếu chọn làm cả 2, thì nó sẽ đợi khi 2 bài kiểm định sức khỏe kết thúc mới ra kết quả

▼ *Cancellation regions*

- Là 1 cơ chế cho phép hủy các nhiệm vụ thông thường (không cần thiết) và reset 1 phần workflow
- 1 task có thể có cancellation region bao gồm các conditions, arcs, tasks
- 1 khi task được hoàn thành thì tất cả token đc loại bỏ khỏi vùng này trước khi các token cho task's output conditions sẽ được sinh ra
- Ví dụ: Khi làm thủ tục đặt vé máy bay qua mạng
 - Nếu khách hàng hủy chuyến bay định đặt thì YAWL sẽ khởi tạo vùng cancellation và sẽ hủy đi vé máy bay đó
 - Workflow sẽ được reset

▼ *Multiple instantiation of Task*

- Nhiều nhiệm vụ cần được thực thi song song nhiều lần
- Thay vì việc sử dụng loop construct, YAWL cho phép khởi tạo song song
- Ví dụ: 1 đơn hàng thương mại điện tử với nhiều đơn hàng chờ
 - 1 khách hàng đặt 3 sản phẩm khác nhau
 - Mỗi đơn hàng cần được xử lý riêng biệt
 - Mỗi nhiệm vụ như "Chuẩn bị đơn hàng" và "Giao hàng" được thực hiện đồng thời cho mỗi sản phẩm

3. Ví dụ

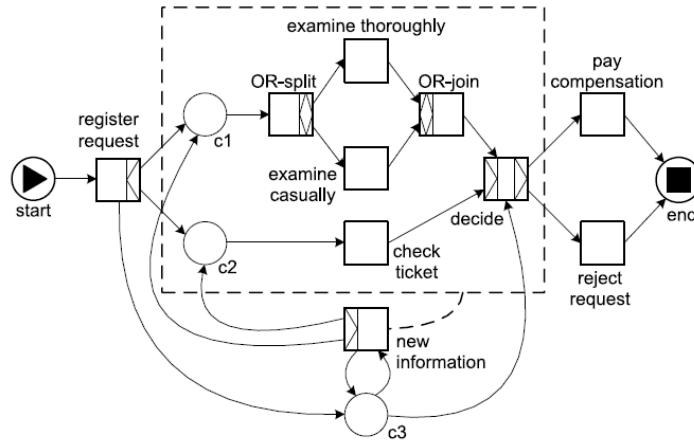


Fig. 3.6 Process model using the YAWL notation

▼ Business Process Modeling Notation (BPMN)

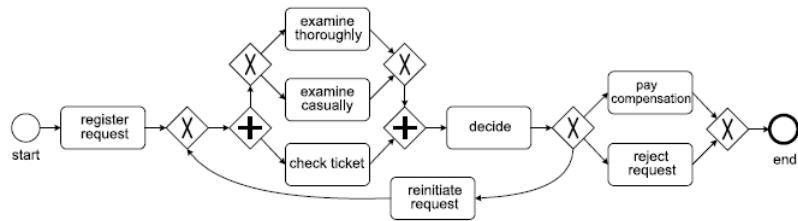
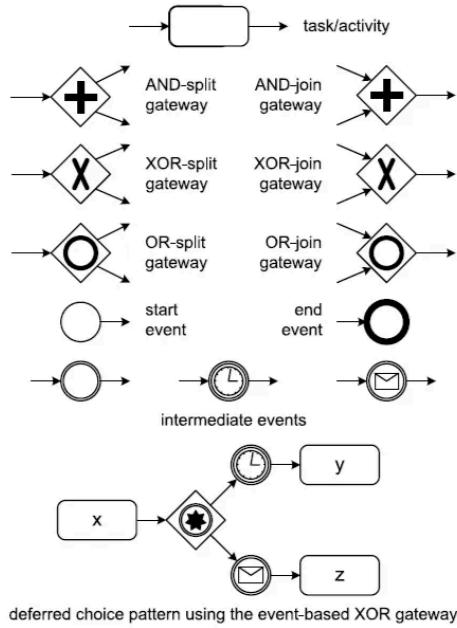


Fig. 3.7 Process model using the BPMN notation

1. Định nghĩa

- Là ngôn ngữ được sử dụng rộng rãi trong lĩnh vực business process model
- Bao gồm các events, activities và gateways
- Điểm khác so với YAWL là thay vì diễn đạt bằng task, thuật ngữ gateway được sử dụng

2. Notation



▼ Giải thích trong sách

To model the so-called *deferred choice* workflow pattern [155] one needs to use the event-based XOR gateway shown in Fig. 3.8. This illustrates the use of events. After executing task x there is a race between two events. One of the events is triggered by a timeout. The other event is triggered by an external message. The first event to occur determines the route taken. If the message arrives before the timer goes off, task z is executed. If the timer goes off before the message arrives, task y is executed. Note that this construct can easily be modeled in YAWL using a condition with two output arcs

▼ Event-Driven Process Chain (EPCs)

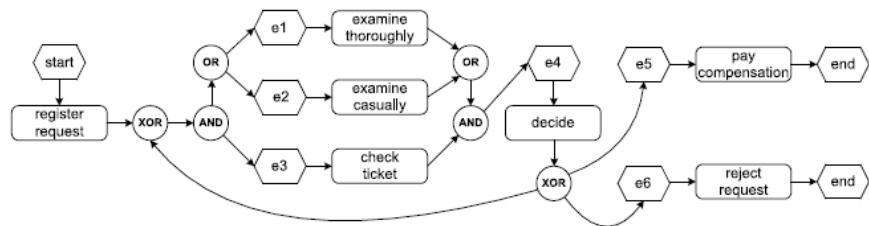
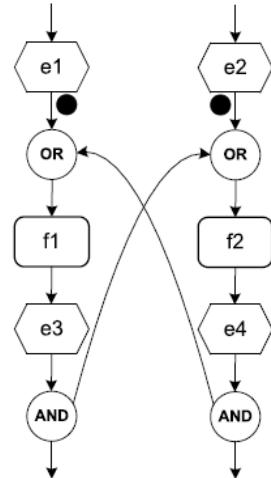


Fig. 3.10 Process model using the EPC notation

Fig. 3.11 The so-called “vicious circle” expressed using the EPC notation



- Event-driven Process Chains (EPCs) cung cấp một bộ ký hiệu cổ điển trong lĩnh vực model business processés
- Connectors trong EPC giống như gateways trong BPMN.
- Có ba loại chính:
 - **AND:** Tất cả các luồng được kích hoạt đồng thời.
 - **XOR:** Chỉ một trong các luồng được chọn.
 - **OR:** Một hoặc nhiều luồng có thể được chọn.
-
-

OR-Split và OR-Join trong EPC

- Trong Hình 3.11, ta thấy cùng một ký hiệu OR được dùng cho cả OR-Split và OR-Join.
- Điều này gây ra sự không rõ ràng:
 - Khi nào thì OR hoạt động như một **Split** (tạo token trên nhiều nhánh)?
 - Khi nào thì OR hoạt động như một **Join** (đợi token rồi hợp lại)?
 - Có cần đợi tất cả token không? Hay chỉ cần một token là đủ?
 - Nếu OR-Join chờ mãi một token không bao giờ đến thì sao? (\Rightarrow Deadlock như Hình 3.11)

Nguyên nhân

- Do cơ chế của OR-join:
 - Một OR-join phải quyết định có nên đợi thêm token không.

- Nếu OR-join bên trái đợi token từ f2, còn OR-join bên phải đợi token từ f1
→ Chúng chặn nhau vô thời hạn.
- OR-join hoạt động không rõ ràng trong EPC:
 - Trong BPMN hoặc YAWL, quy tắc rõ ràng hơn:
 - BPMN OR-join có thể kiểm tra luồng trước đó để quyết định có đợi không.
 - YAWL sử dụng một cơ chế đồng bộ rõ ràng hơn để tránh deadlock.

▼ Casual Nets

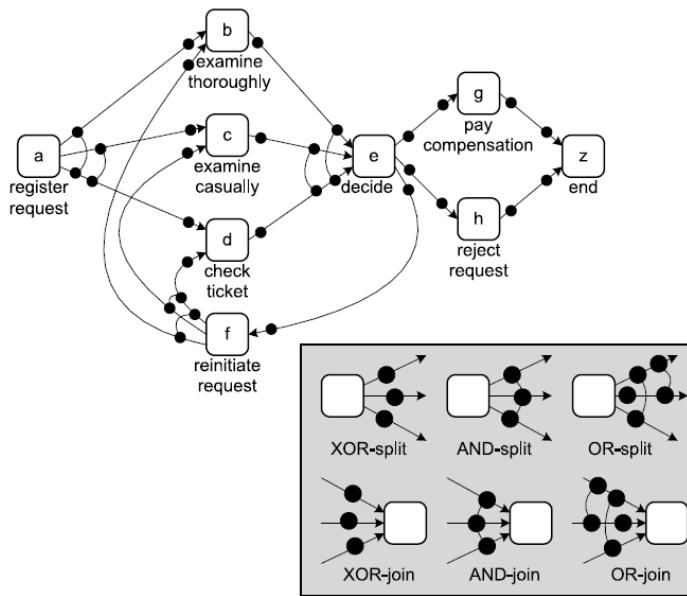


Fig. 3.12 Causal net C_1

1. Casual net

Một Casual net (C-net) được biểu diễn là $C = (A, a_i, a_o, D, I, O)$

- $A \subseteq \mathcal{A}$: là tập hợp hữu hạn của các *activities*
- $a_i \in A$: là hoạt động bắt đầu (*start activity*)
- $a_o \in A$: là hoạt động kết thúc (*end activity*)
- $D \subseteq A \times A$: là tập quan hệ phụ thuộc (*dependency relation*), tức là xác định các hoạt động nào phụ thuộc vào nhau, tức là 1 hoạt động cần phải hoàn thành trước khi 1 hoạt động khác có thể bắt đầu
- $I \in A \rightarrow AS$: là tập xác định các ràng buộc đầu vào cho mỗi hoạt động, tức là hoạt động đó cần những điều kiện gì để thực hiện
- $O \in A \rightarrow AS$: là tập xác định tất cả ràng buộc đầu ra cho mỗi hoạt động, tức là hoạt động đó sẽ ảnh hưởng đến hoạt động nào tiếp theo

- AS : là tập hợp các tập activities

Definition 3.8 (Causal net) A *Causal net* (C-net) is a tuple $C = (A, a_i, a_o, D, I, O)$ where:

- $A \subseteq \mathcal{A}$ is a finite set of *activities*;
- $a_i \in A$ is the *start activity*;
- $a_o \in A$ is the *end activity*;
- $D \subseteq A \times A$ is the *dependency relation*,
- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$;²
- $I \in A \rightarrow AS$ defines the set of possible *input bindings* per activity; and
- $O \in A \rightarrow AS$ defines the set of possible *output bindings* per activity,

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\};$
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in O(a_1)} as\};$
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\};$
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\};$ and
- all activities in the graph (A, D) are on a path from a_i to a_o .

- AS : tập của các tập hợp activities

Gọi $B = \{(a, as^I, as^O) \in A \times \mathcal{P}(A) \times \mathcal{P}(A) \mid as^I \in I(a) \wedge as^O \in O(a)\}$ là tập hợp các hoạt động có ràng buộc (activity bindings)

- Activity binding là bộ (a, as^I, as^O)
- Ví dụ như: $(e, \{b, d\}, \{f\})$ kí hiệu cho sự xảy ra của activity e được nối tiếp bởi b, d và tiếp đó là nối đến f
- Một chuỗi AB khả thi là
 $\langle (a, \emptyset, \{b, d\}), (b, \{a\}, \{e\}), (d, \{a\}, \{e\}), (e, \{b, d\}, \{g\}), (g, \{e\}, \{z\}), (z, \{g\}, \emptyset) \rangle.$

2. Đặc điểm

a. C-net chỉ xem xét "valid sequences"

- BPMN, Petri nets, EPCs, và YAWL đều sử dụng local firing rule để xác định token di chuyển

▼ Giải thích

- Ví dụ: Trong Petri net, nếu 1 transition có đủ token trong các input places, nó có thể firing và tạo token ở output places
- Đây là cách token-game semantics hoạt động, rằng chỉ phụ thuộc vào trạng thái hiện tại, chứ không cần nhìn vào toàn bộ quá trình trước đó.

- Còn đối với C-net, chỉ quan tâm đến toàn bộ chuỗi sự kiện (complete sequences)
- 1 chuỗi sự kiện chỉ có ý nghĩa khi nó tuân thủ theo quy tắc của mô hình
- Các chuỗi không hợp lệ sẽ bị loại bỏ ngay từ đầu
- Ví dụ: Giả sử quy trình mua hàng trực tuyến gồm các bước
 1. Chọn sản phẩm
 2. Thêm vào giỏ hàng
 3. Thanh toán
 4. Nhận hàng
 - Đối với Petri net hoặc BPMN, nếu khách hàng cố nhảy từ bước 1 → 3 mà không qua bước 2 thì phải kiểm tra từng bước lại để xem có hợp lệ không
 - Đối với C-net, nó chỉ cần xem xét cả chuỗi xem có đủ 1 → 2 → 3 → 4 không thì loại bỏ

b. **Tính chất "declarative"**

- C-net mô tả quy trình 1 cách khai báo (declarative), thay vì quy tắc từng bước như BPMN/Petri net
- Nghĩa là thay vì chỉ rõ từng trạng thái chuyển đổi, C-net chỉ quy định điều kiện đầu vào và đầu ra của 1 quy trình

3. Cách hoạt động

a. Binding

Definition 3.9 (Binding) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $B = \{(a, as^I, as^O) \in A \times \mathcal{P}(A) \times \mathcal{P}(A) \mid as^I \in I(a) \wedge as^O \in O(a)\}$ is the set of *activity bindings*. A *binding sequence* σ is a sequence of activity bindings, i.e., $\sigma \in B^*$.

- Ví dụ:

A possible binding sequence for the C-net of Fig. 3.12 is

$$(a, \emptyset, b, d), (b, a, e), (d, a, e), (e, b, d, g), (g, e, z), (z, g, \emptyset).$$

b. State

Với $S = \mathbb{B}(A \times A)$ là không gian trạng thái (state space), ta có hàm xác định trạng thái của chuỗi σ

- Hàm $\psi \in B^* \rightarrow S$ với
 - $\psi(\langle \rangle) = []$
 - $\psi(\sigma \oplus (a, as^I, as^O)) = (\psi(\sigma) \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$

- $\psi(\sigma)$ sẽ trả về trạng thái sau khi thực thi chuỗi có ràng buộc σ
- Ví dụ: nếu $(a, \emptyset, \{b, d\})$ xảy ra thì có thể tính là
$$\begin{aligned}\psi(\langle(a, \emptyset, \{b, d\})\rangle) \\ = \psi(\langle\rangle)\backslash(\emptyset \times \{a\}) \uplus (\{a\} \times \{b, d\}) \\ = [\] \backslash [(a, b), (a, d)] = [(a, b), (a, d)].\end{aligned}$$

Definition 3.10 (State) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $S = \mathbb{B}(A \times A)$ is the *state space* of C . $s \in S$ is a *state*, i.e., a multi-set of pending *obligations*. Function $\psi \in B^* \rightarrow S$ is defined inductively: $\psi(\langle \rangle) = []$ and $\psi(\sigma \oplus (a, as^I, as^O)) = (\psi(\sigma) \backslash (as^I \times \{a\})) \uplus (\{a\} \times as^O)$ for any binding sequence $\sigma \oplus (a, as^I, as^O) \in B^*$.³ $\psi(\sigma)$ is the state after executing binding sequence σ .

- Ví dụ:

$$\begin{aligned}\psi(\langle \rangle) &= [] \\ \psi(\langle(a, \emptyset, \{b, d\})\rangle) &= [(a, b), (a, d)] \\ \psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\})\rangle) &= [(a, b), (d, e)] \\ \psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\})\rangle) &= [(b, e), (d, e)] \\ \psi(\langle(a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset)\rangle) &= []\end{aligned}$$

c. Valid

Definition 3.11 (Valid) Let $C = (A, a_i, a_o, D, I, O)$ be a C-net and $\sigma = \langle(a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \dots, (a_n, as_n^I, as_n^O)\rangle \in B^*$ a binding sequence. σ is a *valid sequence* of C if and only if:

- $a_1 = a_i$, $a_n = a_o$, and $a_k \in A \setminus \{a_i, a_o\}$ for $1 < k < n$;
- $\psi(\sigma) = []$; and
- for any prefix $\langle(a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \dots, (a_k, as_k^I, as_k^O)\rangle = \sigma' \oplus (a_k, as_k^I, as_k^O) \in pref(\sigma)$: $(as_k^I \times \{a_k\}) \leq \psi(\sigma')$.

$V(C)$ is the set of all valid sequences of C .

d. Invalid Sequences

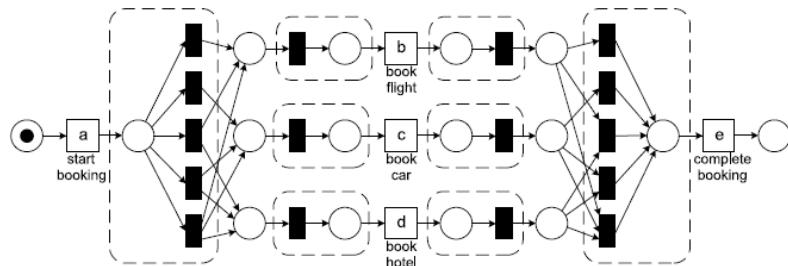


Fig. 3.14 A C-net transformed into a WF-net with silent transitions: every “sound run” of the WF-net corresponds to a valid sequence of the C-net C_2 shown in Fig. 3.13

- Note: là mạng WF nên có nhiều invalid seq
 - Ví dụ như ngay sau khi firing b có thể fire e mà k cần firing c và d
 - Việc này là bởi vì C-net cần hoàn thành toàn bộ chuỗi là kết thúc tại e để duyệt chuỗi

Fig. 3.13 Causal net C_2

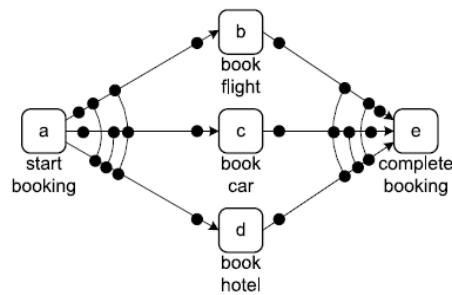
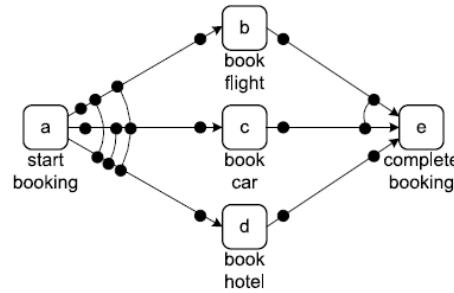
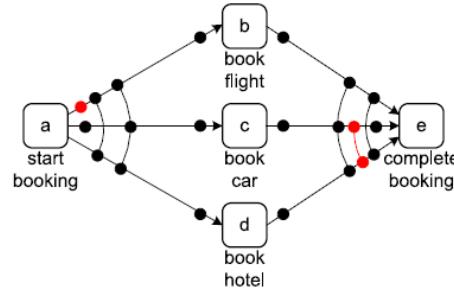


Fig. 3.15 Two C-nets that are not sound. The first net does not allow for any valid sequence, i.e., $V(C) = \emptyset$. The second net has valid sequences but also shows input/output bindings that are not realizable



(a) unsound because there are no valid sequences



(b) unsound although there exist valid sequences

giải thích tại sao lại là invalid seq

e. Soundness of C-net

Definition 3.12 (Soundness of C-nets) A C-net $C = (A, a_i, a_o, D, I, O)$ is *sound* if (a) for all $a \in A$ and $as^I \in I(a)$ there exists a $\sigma \in V(C)$ and $as^O \subseteq A$ such that $(a, as^I, as^O) \in \sigma$, and (b) for all $a \in A$ and $as^O \in O(a)$ there exists a $\sigma \in V(C)$ and $as^I \subseteq A$ such that $(a, as^I, as^O) \in \sigma$.

- Để đảm bảo C-net là valid sequences thì mỗi phần trong C-net phải là các valid seq

▼ Process Trees

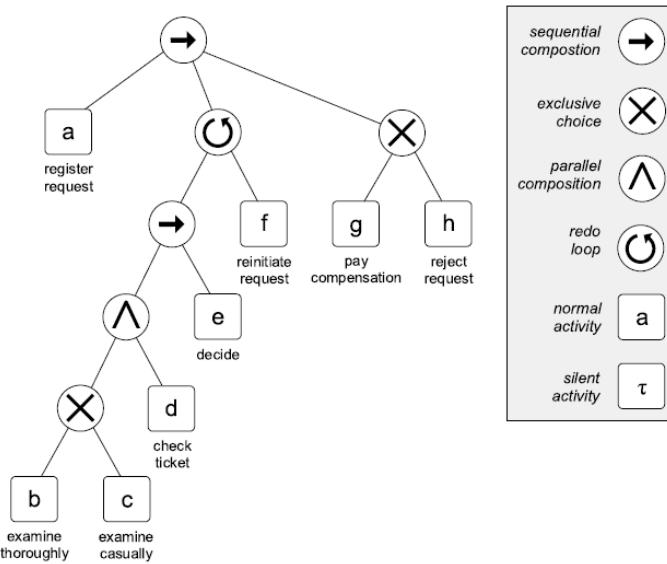


Fig. 3.17 Process tree $\rightarrow(a, \circlearrowright(\rightarrow(\Lambda(\times(b, c), d), e), f), \times(g, h))$ showing the different process tree operators

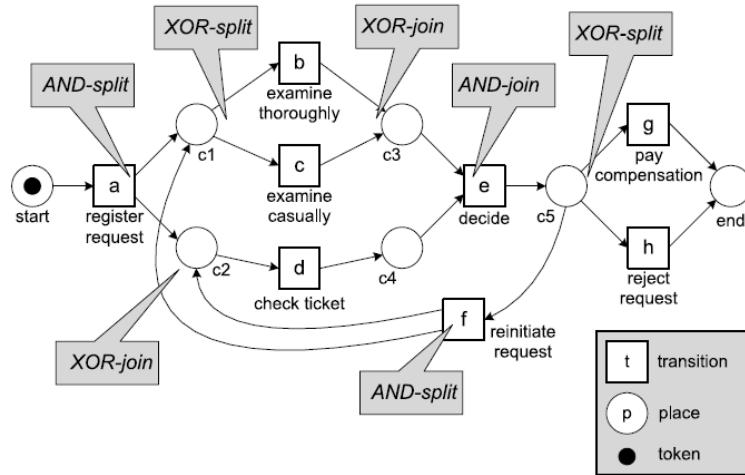


Fig. 3.2 A marked Petri net

1. Định nghĩa

- Process trees có dạng là block-structured model

Definition 3.13 (Process tree) Let $A \subseteq \mathcal{A}$ be a finite set of activities with $\tau \notin A$.

$\oplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ is the set of *process tree operators*.

- If $a \in A \cup \{\tau\}$, then $Q = a$ is a process tree,
- If $n \geq 1$, Q_1, Q_2, \dots, Q_n are process trees, and $\oplus \in \{\rightarrow, \times, \wedge\}$, then $Q = \oplus(Q_1, Q_2, \dots, Q_n)$ is a process tree, and
- If $n \geq 2$ and Q_1, Q_2, \dots, Q_n are process trees, then $Q = \circlearrowleft(Q_1, Q_2, \dots, Q_n)$ is a process tree.

\mathcal{Q}_A is the set of *all process trees* over A .

- Toán tử redo \circlearrowleft yêu cầu ít nhất là có 2 nodes con

- Ví dụ: Process tree $\circlearrowleft(a, b, c)$ tạo ra các trace sau:

$$\{\langle a \rangle, \langle a, b, a \rangle, \langle a, c, a \rangle, \langle a, b, a, b, a \rangle, \langle a, c, a, c, a \rangle, \langle a, c, a, b, a \rangle, \langle a, b, a, c, a \rangle, \dots\}$$

- Phần "do" là a và phần "redo" là b hoặc c , nên mỗi lần chạy b, c sẽ thực thi và loop lại qua a .

- Các activities giống nhau có thể xuất hiện nhiều lần trong cây

- Ví dụ: $\rightarrow(a, a, a)$ là chuỗi 3 activities a
 - Đổi với trường hợp $\rightarrow(a, a, a)$ và $\wedge(a, a, a)$ là không phân biệt được vì trace của chúng đều là $\langle a, a, a \rangle$

- Silent activity τ

- Process tree $\times(a, \tau)$: nghĩa là activity a có thể được skip
 - Process tree $\circlearrowleft(a, \tau)$: nghĩa là activity a có thể được thực thi ít nhất 1 lần, phần "do" là a , còn phần "redo" là τ nên process có thể loop nhưng không thực thi activity nào
 - Process tree $\circlearrowleft(\tau, a)$: nghĩa là activity a loop vô hạn lần, do phần "do" là τ (bên trái), còn phần "redo" là a (bên phải)

2. Process Tree \rightarrow WF-net

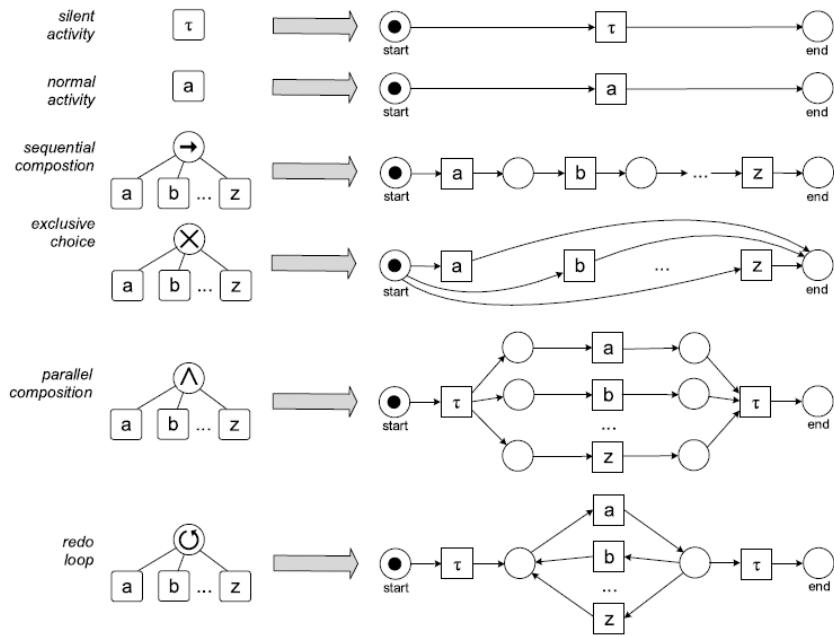


Fig. 3.18 Mapping process trees onto WF-nets

- Việc ánh xạ như hình 3.18 có thể chuyển đổi sang các ngôn ngữ khác như BPMN, YAWL, EPCs, UML activity diagrams, statecharts,...

3. Cách hoạt động

The semantics of process trees can also be defined directly (without a mapping to WF-nets). To do this we first define two operators on sequences, concatenation (\cdot) and shuffle (\diamond).

Let $\sigma_1, \sigma_2 \in A^*$ be two sequences over A . $\sigma_1 \cdot \sigma_2 \in A^*$ concatenates two sequences, e.g., $\langle w, o \rangle \cdot \langle r, d \rangle = \langle w, o, r, d \rangle$. Concatenation can be generalized to sets of sequences. Let $S_1, S_2, \dots, S_n \subseteq A^*$ be sets of sequences over A . $S_1 \cdot S_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in S_1 \wedge \sigma_2 \in S_2\}$. For example, $\{\langle w, o \rangle, \langle \rangle\} \cdot \{\langle r, d \rangle, \langle k \rangle\} = \{\langle w, o, r, d \rangle, \langle w, o, k \rangle, \langle r, d \rangle, \langle k \rangle\}$. $\bigodot_{1 \leq i \leq n} S_i = S_1 \cdot S_2 \cdots S_n$ concatenates an ordered collection of sets of sequences.

$\sigma_1 \diamond \sigma_2$ generates the set of all interleaved sequences (shuffle). For example, $\langle w, o \rangle \diamond \langle r, d \rangle = \{\langle w, o, r, d \rangle, \langle w, r, o, d \rangle, \langle r, w, o, d \rangle, \langle w, r, d, o \rangle, \langle r, w, d, o \rangle, \langle r, d, w, o \rangle\}$. Note that the ordering in the original sequences is preserved, e.g., d cannot appear before r . Another example is $\langle w, o, r \rangle \diamond \langle d \rangle = \{\langle w, o, r, d \rangle, \langle w, o, d, r \rangle, \langle w, d, o, r \rangle, \langle d, w, o, r \rangle\}$. The shuffle operator can also be generalized to sets of sequences. $S_1 \diamond S_2 = \{\sigma \in \sigma_1 \diamond \sigma_2 \mid \sigma_1 \in S_1 \wedge \sigma_2 \in S_2\}$. The shuffle operator is commutative and associative, i.e., $S_1 \diamond S_2 = S_2 \diamond S_1$ and $(S_1 \diamond S_2) \diamond S_3 = S_1 \diamond (S_2 \diamond S_3)$. We write $\diamond_{1 \leq i \leq n} S_i = S_1 \diamond S_2 \diamond \cdots \diamond S_n$ to interleave sets of sequences.

Definition 3.14 (Semantics) Let $Q \in \mathcal{Q}_A$ be a process tree over A . $\mathcal{L}(Q)$ is the language of Q , i.e., the set of traces that can be generated by it. $\mathcal{L}(Q)$ is defined recursively:

- $\mathcal{L}(Q) = \{\langle a \rangle\}$ if $Q = a \in A$,
- $\mathcal{L}(Q) = \{\langle \rangle\}$ if $Q = \tau$,
- $\mathcal{L}(Q) = \bigodot_{1 \leq i \leq n} \mathcal{L}(Q_i)$ if $Q = \rightarrow(Q_1, Q_2, \dots, Q_n)$,
- $\mathcal{L}(Q) = \bigcup_{1 \leq i \leq n} \mathcal{L}(Q_i)$ if $Q = \times(Q_1, Q_2, \dots, Q_n)$,
- $\mathcal{L}(Q) = \diamond_{1 \leq i \leq n} \mathcal{L}(Q_i)$ if $Q = \wedge(Q_1, Q_2, \dots, Q_n)$,
- $\mathcal{L}(Q) = \{\sigma_1 \cdot \sigma'_1 \cdot \sigma_2 \cdot \sigma'_2 \cdots \sigma_m \in A^* \mid m \geq 1 \wedge \forall_{1 \leq j \leq m} \sigma_j \in \mathcal{L}(Q_1) \wedge \forall_{1 \leq j < m} \sigma'_j \in \bigcup_{2 \leq i \leq n} \mathcal{L}(Q_i)\}$ if $Q = \circlearrowleft(Q_1, Q_2, \dots, Q_n)$.

The following examples further illustrate the process tree operators and their semantics:

- $\mathcal{L}(\tau) = \{\langle \rangle\}$,
- $\mathcal{L}(a) = \{\langle a \rangle\}$,
- $\mathcal{L}(\rightarrow(a, b, c)) = \{\langle a, b, c \rangle\}$,
- $\mathcal{L}(\times(a, b, c)) = \{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$,
- $\mathcal{L}(\wedge(a, b, c)) = \{\langle a, b, c \rangle, \langle a, c, b \rangle, \langle b, a, c \rangle, \langle b, c, a \rangle, \langle c, a, b \rangle, \langle c, b, a \rangle\}$,
- $\mathcal{L}(\circlearrowleft(a, b, c)) = \{\langle a \rangle, \langle a, b, a \rangle, \langle a, c, a \rangle, \langle a, b, a, c, a \rangle, \langle a, c, a, b, a \rangle, \dots\}$,
- $\mathcal{L}(\rightarrow(a, \times(b, c), \wedge(a, a))) = \{\langle a, b, a, a \rangle, \langle a, c, a, a \rangle\}$,
- $\mathcal{L}(\times(\tau, a, \tau, \rightarrow(\tau, b), \wedge(c, \tau))) = \{\langle \rangle, \langle a \rangle, \langle b \rangle, \langle c \rangle\}$, and
- $\mathcal{L}(\circlearrowleft(a, \tau, c)) = \{\langle a \rangle, \langle a, a \rangle, \langle a, a, a \rangle, \langle a, c, a \rangle, \langle a, a, c, a \rangle, \langle a, c, a, c, a \rangle, \dots\}$.

Process trees are *sound by construction*. Process discovery algorithms may exploit this when searching for a process model describing the event data. There are some similarities with other notations. *Process calculi* such as CSP and CCS use similar operators to model processes. Process trees can be viewed as a carefully