

Supervisory Control of Discrete Event Systems

Part 2: Automata Models, Analysis and Operations

Dr Andrea Paoli
Associate Professor in Robotics and Automation

Loughborough University
Wolfson School of Mechanical, Electrical and Manufacturing Engineering

Fall semester 2026-2027
University of Ferrara

Outline

- 1 Formal Automata & String Evaluation
- 2 Structural Analysis & State Properties
- 3 Blocking Characterization
- 4 System Interconnections & Composition Operations
- 5 Conclusions
- 6 Review Questions & Self-Assessment

Part 2: Intended Learning Outcomes (ILOs)

- 1 **Define and evaluate deterministic automata:** Formalize state-based generators and evaluate trace execution via the recursive extended transition function .
- 2 **Distinguish generated and marked languages:** Differentiate between ongoing physical execution and completed task specifications generated by an automaton.
- 3 **Analyze state-space reachability properties:** Compute accessible and co-accessible components, and apply the Trim operation to obtain minimal, non-redundant state models.
- 4 **Detect and characterize blocking anomalies:** Identify and formally distinguish between static termination (**Deadlock**) and dynamic cyclic traps (**Livelock**), verifying the non-blocking condition.
- 5 **Synthesize modular systems via composition:** Compute the **Product** and **Parallel Composition** of concurrent subsystems, managing shared-event synchronization and private execution.

Deterministic Finite Automata: Formal Definition

The 6-Tuple Model

A **Deterministic Finite-State Automaton (DFA)** is formally defined as: $G = (X, \Sigma, f, \Gamma, x_0, X_m)$

- X : Finite set of discrete **states** ($|X| < \infty$).
- Σ : Finite set of discrete **events** (alphabet).
- $f : X \times \Sigma \rightarrow X$: **Partial transition function**. $f(x, \sigma) = x'$ denotes that event σ transitions the system from state x to state x' . If no transition is possible, $f(x, \sigma)$ is undefined ($f(x, \sigma)!$ denotes that it is defined).
- $\Gamma : X \rightarrow 2^\Sigma$: **Active event function** (feasible event set):

$$\Gamma(x) = \{\sigma \in \Sigma \mid f(x, \sigma)!\}$$

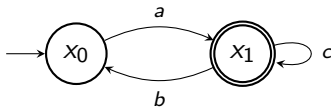
- $x_0 \in X$: The unique **initial state**.
- $X_m \subseteq X$: Set of **marked states** (accepting, completed task, or safe return states).

State Transition Graph Representation

Visual Syntax

An automaton G is represented as a directed graph where:

- **States (X):** Circles. Single border for $x \in X \setminus X_m$; double border for marked states $x \in X_m$.
- **Initial state (x_0):** Indicated by an incoming free arrow.
- **Transitions (f):** Directed edges labeled with event $\sigma \in \Sigma$.



Graph Components

- $X = \{x_0, x_1\}$, x_0 initial, $X_m = \{x_1\}$
- $\Sigma = \{a, b, c\}$
- $\Gamma(x_0) = \{a\}$; $\Gamma(x_1) = \{b, c\}$
- $f(x_0, a) = x_1$, $f(x_1, b) = x_0$, $f(x_1, c) = x_1$
- $f(x_0, b)$ is **undefined** ($b \notin \Gamma(x_0)$)

The Extended Transition Function $f(x, s)$

From Single Events to Trace Execution

To model the state evolution under a sequence of events (a string $s \in \Sigma^*$), we extend $f : X \times \Sigma \rightarrow X$ to $f : X \times \Sigma^* \rightarrow X$ by mathematical induction.

Inductive Definition

For any state $x \in X$, string $s \in \Sigma^*$, and event $\sigma \in \Sigma$:

$$f(x, \varepsilon) := x \quad (\text{Base case: empty string causes no state change})$$

$$f(x, s\sigma) := f(f(x, s), \sigma) \quad (\text{Inductive step})$$

Existence Condition: Notation $f(x, s)!$

$f(x, s)!$ means that the trace s is physically feasible from state x :

$$f(x, s\sigma)! \iff [f(x, s)! \wedge \sigma \in \Gamma(f(x, s))]$$

Note: If any intermediate transition is undefined, the entire extended function is undefined.

Languages Generated and Marked by G

Generated Language $L(G)$

The set of all event traces that the system can physically execute starting from x_0 :

$$L(G) := \{s \in \Sigma^* \mid f(x_0, s)!\}$$

Fundamental Property: By construction, $L(G)$ is always **prefix-closed**:

$$L(G) = \overline{L(G)}$$

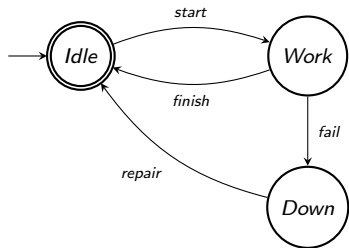
Marked Language $L_m(G)$

The subset of traces in $L(G)$ that terminate in an accepted/marked state:

$$L_m(G) := \{s \in L(G) \mid f(x_0, s) \in X_m\} \subseteq L(G)$$

Physical Meaning: Represents completed tasks, operational cycles, or safe target states. $L_m(G)$ is **not** necessarily prefix-closed.

Example: Formal Model & Language Derivation



Task for Students

Inspect the state transition graph above and determine:

- 1 **System Tuple:** Explicitly write out X , Σ , x_0 , and X_m .
- 2 **Active Events:** Determine $\Gamma(\text{Idle})$, $\Gamma(\text{Work})$, and $\Gamma(\text{Down})$.
- 3 **Marked Language $L_m(G)$:** Express $L_m(G)$ as a regular expression over Σ .
- 4 **Generated Language $L(G)$:** Relate $L(G)$ mathematically to $L_m(G)$.

Guiding Question

Notice that $X_m = \{\text{Idle}\}$. What physical condition does reaching this marked state signify for both normal and faulty production runs?

Solution: Formal Model & Language Derivation

1. Formal 6-Tuple Model

- $X = \{Idle, Work, Down\}$
- $\Sigma = \{start, finish, fail, repair\}$
- $x_0 = Idle$
- $X_m = \{Idle\}$

2. Active Event Function $\Gamma(x)$

- $\Gamma(Idle) = \{start\}$
- $\Gamma(Work) = \{finish, fail\}$
- $\Gamma(Down) = \{repair\}$

3. Marked Language $L_m(G)$

All traces leading to cycle completion at *Idle*:

$$L_m(G) = (start \cdot (finish + fail \cdot repair))^*$$

4. Generated Language $L(G)$

Every feasible prefix along normal or fault branches:

$$L(G) = \overline{L_m(G)}$$

E.g., $start \cdot fail \in L(G)$, but $start \cdot fail \notin L_m(G)$.

Key Takeaway

$L_m(G)$ isolates full operational milestones (nominal processing and recovered cycles), whereas $L(G)$ captures every intermediate physical state.

Reachability: The Accessible State Space X_{ac}

Motivation

Automata models often contain states that can never be reached from the initial state x_0 via any sequence of physically feasible events.

- These unreachable states contribute nothing to system trajectories.
- Pruning them simplifies models and reduces computational complexity.

Formal Definition of Reachable States

The **accessible (reachable) state set** $X_{ac} \subseteq X$ consists of all states connected to x_0 by at least one valid string:

$$X_{ac} := \{x \in X \mid \exists s \in \Sigma^* \text{ such that } f(x_0, s) = x\}$$

Algorithmic Intuition

X_{ac} is computed via standard Breadth-First Search (BFS) or Depth-First Search (DFS) starting exclusively from $\{x_0\}$ and following forward active transitions.

The Accessible Sub-Automaton: $Ac(G)$

Definition of the Accessible Sub-Automaton

Given $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, its **accessible component** is:

$$Ac(G) := (X_{ac}, \Sigma, f_{ac}, \Gamma_{ac}, x_0, X_{m,ac})$$

where:

- $f_{ac} := f|_{X_{ac} \times \Sigma \rightarrow X_{ac}}$ (restriction of f to X_{ac})
- $\Gamma_{ac}(x) := \Gamma(x) \cap \{\sigma \in \Sigma \mid f(x, \sigma) \in X_{ac}\}$
- $X_{m,ac} := X_m \cap X_{ac}$ (reachable marked states)

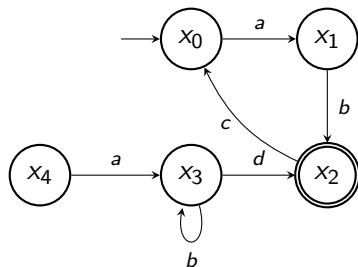
Fundamental Invariance Property

Eliminating unreachable states never alters the linguistic behavior:

$$L(Ac(G)) = L(G) \quad \text{and} \quad L_m(Ac(G)) = L_m(G)$$

$Ac(G)$ is linguistically equivalent to G , but structurally minimal with respect to reachability.

Example: Computing the Accessible Component $Ac(G)$



System Overview

$X = \{x_0, x_1, x_2, x_3, x_4\}$, $\Sigma = \{a, b, c, d\}$
 x_0 initial, $X_m = \{x_2\}$

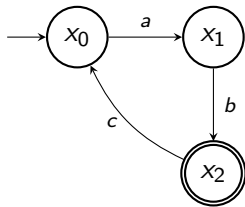
Task for Students

- 1 Trace forward reachability starting from x_0 .
- 2 Identify the set X_{ac} .
- 3 Why is state x_3 **not** in X_{ac} , even though there is a transition $x_3 \xrightarrow{d} x_2$?
- 4 Draw or define $Ac(G)$.

Reflection Question

Does the incoming edge $x_3 \xrightarrow{d} x_2$ make x_3 reachable from x_0 ? Pay close attention to transition directionality.

Solution: Accessible Component $Ac(G)$



1. Forward Trajectory Evaluation

- Base: $f(x_0, \varepsilon) = x_0 \in X_{ac}$
- Step 1: $f(x_0, a) = x_1 \in X_{ac}$
- Step 2: $f(x_0, ab) = x_2 \in X_{ac}$
- Step 3: $f(x_0, abc) = x_0$ (cycle closed)

2. Pruned States

- x_4 : No incoming transitions from x_0 .
- x_3 : Has an outgoing edge to x_2 , but **no path from** x_0 . Forward reachability is strictly directed!
- Hence: $X \setminus X_{ac} = \{x_3, x_4\}$.

3. Formally Pruned Model $Ac(G)$

$X_{ac} = \{x_0, x_1, x_2\}$, $X_{m,ac} = \{x_2\}$, $\Sigma_{ac} = \{a, b, c\}$ (event d is eliminated).

Notice: $L(Ac(G)) = L(G) = \overline{(abc)^*} \cdot \{\varepsilon, a, ab\}$, preserving identical language dynamics.

Co-Reachability: The Co-Accessible State Space X_{coac}

Motivation

A state might be reachable from x_0 , yet physically incapable of ever reaching a marked state in X_m .

- Entering such a state means the system is trapped: it can never complete its assigned task or return to a nominal resting point.
- These states represent dead ends or dynamic traps (precursors to blocking).

Formal Definition of Co-Reachable States

The **co-accessible (co-reachable) state set** $X_{coac} \subseteq X$ consists of all states that can reach at least one marked state via some valid trace:

$$X_{coac} := \{x \in X \mid \exists s \in \Sigma^* \text{ such that } f(x, s) \in X_m\}$$

Algorithmic Intuition (Backward Search)

While X_{ac} is found by forward search from $\{x_0\}$, X_{coac} is computed via **backward search** along reversed transitions starting from the target set X_m .

The Co-Accessible Sub-Automaton: $\text{CoAc}(G)$

Definition of the Co-Accessible Sub-Automaton

Given $G = (X, \Sigma, f, \Gamma, x_0, X_m)$, its **co-accessible component** is:

$$\text{CoAc}(G) := (X_{\text{coac}}, \Sigma, f_{\text{coac}}, \Gamma_{\text{coac}}, x_{0,\text{coac}}, X_m)$$

where:

- $f_{\text{coac}} := f|_{X_{\text{coac}} \times \Sigma \rightarrow X_{\text{coac}}}$ (restriction of f to X_{coac})
- $\Gamma_{\text{coac}}(x) := \Gamma(x) \cap \{\sigma \in \Sigma \mid f(x, \sigma) \in X_{\text{coac}}\}$
- $x_{0,\text{coac}} := x_0$ if $x_0 \in X_{\text{coac}}$, otherwise $\text{CoAc}(G)$ is the empty automaton.

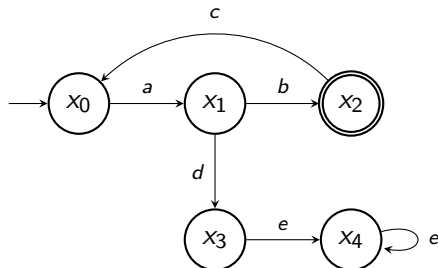
Language Relationship

Pruning non-co-accessible states preserves the marked language, but may reduce the generated language:

$$L_m(\text{CoAc}(G)) = L_m(G), \quad L(\text{CoAc}(G)) \subseteq L(G)$$

Traces removed from $L(G)$ are precisely those that lead to dead ends with no path to X_m .

Example: Computing the Co-Accessible Component $\text{CoAc}(G)$



System Overview

$X = \{x_0, x_1, x_2, x_3, x_4\}$, $\Sigma = \{a, b, c, d, e\}$
 x_0 initial, $X_m = \{x_2\}$

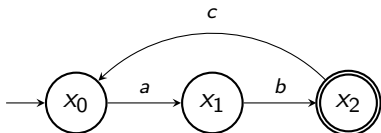
Task for Students

- 1 Identify the target marked set X_m .
- 2 Perform a backward path search from X_m along incoming transitions.
- 3 Determine which states belong to X_{CoAc} .
- 4 What physical behavior do states x_3 and x_4 represent?

Question

Once the system fires event d from state x_1 , can it ever execute event c or reach x_2 again?

Solution: Co-Accessible Component $\text{CoAc}(G)$



1. Backward Search from $X_m = \{x_2\}$

- Base: $x_2 \in X_m \implies x_2 \in X_{\text{CoAc}}$ (path ε)
- Step 1: $f(x_1, b) = x_2 \implies x_1 \in X_{\text{CoAc}}$ (path b)
- Step 2: $f(x_0, a) = x_1 \implies x_0 \in X_{\text{CoAc}}$ (path ab)

2. Eliminated States ($X \setminus X_{\text{CoAc}}$)

- x_3, x_4 : From x_3 , the only transitions lead to $\{x_3, x_4\}$. No sequence of transitions connects x_3 or x_4 back to x_2 .
- They form an unrecoverable trap region!

3. Resulting Sub-Automaton $\text{CoAc}(G)$

$$X_{\text{CoAc}} = \{x_0, x_1, x_2\}, \quad x_0 \in X_{\text{CoAc}}, \quad X_{m, \text{CoAc}} = \{x_2\}.$$

Linguistic check: $L_m(\text{CoAc}(G)) = L_m(G)$, while $L(\text{CoAc}(G)) \subset L(G)$ because string prefixes such as $a \cdot d$ and $a \cdot d \cdot e$ have been pruned.

The Trim Operation: $\text{Trim}(G)$

Definition: Trim Automaton

An automaton $G = (X, \Sigma, f, \Gamma, x_0, X_m)$ is said to be **trim** if it is simultaneously:

- **Accessible:** Every state can be reached from the initial state ($X = X_{ac}$).
- **Co-accessible:** Every state can reach at least one marked state ($X = X_{coac}$).

The Trim Sub-Automaton

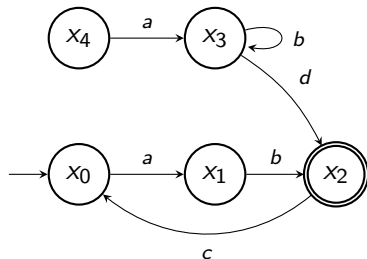
Trim operation prunes unreachable and dead-end states: $\text{Trim}(G) := \text{Ac}(\text{CoAc}(G)) = \text{CoAc}(\text{Ac}(G))$.

The state space of the trim automaton is: $X_{trim} = X_{ac} \cap X_{coac}$

Properties and Order Independence

- **Commutativity:** The order does not matter: pruning unreachable states first or non-coreachable states first yields the exact same result.
- **Marked Language Invariance:** $L_m(\text{Trim}(G)) = L_m(G)$.
- **Generated Language Contraction:** $L(\text{Trim}(G)) \subseteq L(G)$, retaining only traces that can be extended into completed tasks.

Exercise: Applying the Trim Operation



System Overview

$X = \{x_0, x_1, x_2, x_3, x_4, x_5\}$, $X_m = \{x_2\}$
 $\Sigma = \{a, b, c, d, e\}$, x_0 initial

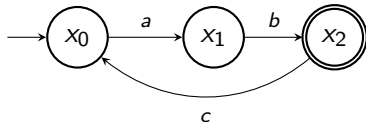
Task for Students

- 1 Compute the accessible set X_{ac} via forward search from x_0 .
- 2 Compute the co-accessible set X_{coac} via backward search from X_m .
- 3 Determine $X_{trim} = X_{ac} \cap X_{coac}$.
- 4 Draw the resulting automaton $\text{Trim}(G)$.

Check

Does evaluating $\text{Ac}(\text{CoAc}(G))$ produce the identical state space and graph as $\text{CoAc}(\text{Ac}(G))$?

Solution: Applying the Trim Operation



1. Set Intersections

- **Ac:** $X_{ac} = \{x_0, x_1, x_2, x_4, x_5\}$ (x_3 pruned)
- **Co-Ac:** $X_{coac} = \{x_0, x_1, x_2, x_3\}$ (x_4, x_5 pruned)
- **Trim Set:** $X_{trim} = X_{ac} \cap X_{coac} = \{x_0, x_1, x_2\}$

2. Eliminated Components

- $x_3 \notin X_{ac}$: disconnected from x_0 .
- $x_4, x_5 \notin X_{coac}$: sink trap region with no path to x_2 .
- Events $\{d, e\}$ are eliminated entirely.

3. Language Summary

$L_m(\text{Trim}(G)) = L_m(G) = (abc)^*$, while unrecoverable prefix deviations (such as ad) are pruned from the generated behavior: $L(\text{Trim}(G)) \subset L(G)$.

Blocking Condition & Nonblocking Property

Linguistic Definition of Nonblocking Automata

An automaton G is called **nonblocking** (or strictly nonblocking) if: $\overline{L_m(G)} = L(G)$.

Every feasible prefix $s \in L(G)$ can always be extended by some string $t \in \Sigma^*$ such that $st \in L_m(G)$.

Formal Definition of Blocking

An automaton G is **blocking** if:

$$\overline{L_m(G)} \subset L(G) \iff \exists s \in L(G) \text{ such that } \forall t \in \Sigma^*, st \notin L_m(G)$$

Once trace s is executed, the system is permanently precluded from reaching any marked state.

Structural Characterization

Assuming G is accessible ($G = \text{Ac}(G)$):

$$G \text{ is nonblocking} \iff G \text{ is co-accessible} \iff G = \text{Trim}(G)$$

Taxonomy of Blocking: Deadlock vs. Livelock

Deadlock (Static Halt)

The system enters an unmarked state where **no further events can occur**:

$$\exists x \in X \setminus X_m \quad \text{such that} \quad \Gamma(x) = \emptyset$$

- Complete cessation of physical motion/activity.
- Examples: resource exhaustion, mechanical jam, unhandled fault state.
- Inspection: *sink nodes* outside X_m .

Livelock (Dynamic Trap)

The system remains **perpetually active**, but cannot reach a marked state:

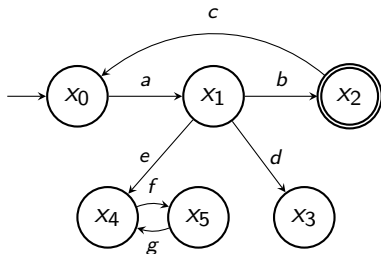
$$\forall x \in X_{trap}, \Gamma(x) \neq \emptyset, \quad \text{yet} \quad X_{trap} \cap X_{coac} = \emptyset$$

- Events continue to fire indefinitely in a cycle without ever achieving the operational goal.
- Examples: infinite polling loop, starvation cycle, fruitless negotiation.
- Structural origin: strongly connected sub-graphs outside X_{coac} .

Key Supervisory Goal

In Supervisory Control Theory (SCT), the central objective is synthesizing a supervisor S such that the closed-loop system S/G is guaranteed to be nonblocking: $\overline{L_m(S/G)} = L(S/G)$.

Exercise: Diagnosing Deadlock and Livelock



System Specifications

$X = \{x_0, x_1, x_2, x_3, x_4, x_5\}$, x_0 initial

$\Sigma = \{a, b, c, d, e, f, g\}$, $X_m = \{x_2\}$

Task for Students

- 1 Verify whether the automaton is accessible ($X = X_{ac}$).
- 2 Identify any **Deadlock** states and specify the trace that leads there.
- 3 Identify any **Livelock** regions and explain why events can fire forever.
- 4 Is G nonblocking? Formally check whether $\overline{L_m(G)} = L(G)$.

Diagnostic Test

What is the fundamental difference between the system's physical state after trace $a \cdot d$ versus trace $a \cdot e \cdot f$?

Solution: Diagnosing Deadlock and Livelock

1. Structural Reachability

- $X_{ac} = X$ (every state is reachable from x_0).
- Backward search from $X_m = \{x_2\}$ yields:

$$X_{coac} = \{x_0, x_1, x_2\}$$

- Non-coreachable subset: $\{x_3, x_4, x_5\}$.

2. Deadlock Identification

- **State:** $x_3 \notin X_m$ with $\Gamma(x_3) = \emptyset$ (*sink*).
- **Trace:** $s = a \cdot d \in L(G)$.
- Complete halt: no event can execute.

3. Livelock Identification

- **States:** $\{x_4, x_5\} \cap X_{coac} = \emptyset$.
- **Behavior:** Active events $\Gamma(x_4) = \{f\}$, $\Gamma(x_5) = \{g\}$ keep alternating dynamically:

$$s = a \cdot e \cdot (f \cdot g)^* \in L(G)$$

- Perpetual cycling without ever reaching X_m .

4. Nonblocking Check

G is **blocking**: $\overline{L_m(G)} \subset L(G)$.
Strings ad and aef belong to $L(G)$ but cannot be completed to any string in $L_m(G)$.

Remediation

Synthesizing a supervisory control policy must disable events $\{d, e\}$ at state x_1 to guarantee nonblocking behavior.

Modular Modeling & Interconnection Taxonomy

Why Modular Composition?

Real-world industrial systems consist of multiple interacting physical sub-units (e.g., robotic arms, CNC machines, conveyors, buffers).

- **Monolithic modeling** becomes unfeasible due to state-space explosion.
- **Modular modeling** represents each sub-component as an individual DFA G_i , synthesizing the overall plant via formal algebraic composition operations.

Event Classification

Given two systems G_1 and G_2 with alphabets Σ_1 and Σ_2 , the pooled alphabet $\Sigma_1 \cup \Sigma_2$ partitions naturally into three disjoint subsets:

- **Exclusive to G_1 :** $\Sigma_1 \setminus \Sigma_2$ (events recognized solely by G_1).
- **Exclusive to G_2 :** $\Sigma_2 \setminus \Sigma_1$ (events recognized solely by G_2).
- **Common / Shared:** $\Sigma_1 \cap \Sigma_2$ (events recognized by both subsystems).

How these three subsets are permitted to execute defines the fundamental distinction between Product and Parallel composition.

Composition Paradigms: Product vs. Parallel

The Core Question: How Do Private Events Behave?

When systems G_1 and G_2 run together, they have private events ($\Sigma_1 \setminus \Sigma_2$, $\Sigma_2 \setminus \Sigma_1$) and shared events ($\Sigma_1 \cap \Sigma_2$). How they handle unshared events defines the composition rules.

Product: $G_1 \times G_2$

- Both systems **must agree** on every single event.
- Private events are **blocked**: neither system moves without the other (rendez-vous).
- **Intuition**: Two-hand safety control circuit proceeds if and only if both physical switches are engaged simultaneously.

Parallel: $G_1 \parallel G_2$

- Private events execute **asynchronously** (one moves, the other waits).
- Shared events require **lock-step synchronization** (handshake).
- **Intuition**: Two workstations operating independently, synchronizing only to pass a workpiece.

Equivalence Condition

When alphabets are identical ($\Sigma_1 = \Sigma_2$), private events vanish and: $G_1 \parallel G_2 \equiv G_1 \times G_2$

The Synchronous Product: Formal Definition

Mathematical Structure

Given two automata $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0,1}, X_{m,1})$, $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0,2}, X_{m,2})$, their **product** is:

$$G_1 \times G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{\times}, \Gamma_{\times}, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

- **State Space:** Cartesian product of individual state sets.
- **Initial State:** Joint initial condition.
- **Marked States:** Both components must simultaneously reach a marked state.

Transition Function & Active Event Set

For any compound state $(x_1, x_2) \in X_1 \times X_2$ and event $\sigma \in \Sigma_1 \cup \Sigma_2$:

$$f_{\times}((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

The active event set is the intersection of local active sets: $\Gamma_{\times}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2) \subseteq \Sigma_1 \cap \Sigma_2$.

The Synchronous Product: Language Properties

The Fate of Exclusive Events

Consider an event exclusive to one system, e.g., $\sigma \in \Sigma_1 \setminus \Sigma_2$:

- $\sigma \notin \Sigma_2 \implies \sigma \notin \Gamma_2(x_2) \implies \sigma \notin \Gamma_1(x_1) \cap \Gamma_2(x_2), \quad \forall (x_1, x_2).$
- **Core Consequence:** Any exclusive event is **permanently disabled** in the product system.

Language Intersection Property

The product enforces strict simultaneous execution over common alphabet strings:

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2) \cap (\Sigma_1 \cap \Sigma_2)^*$$

$$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2) \cap (\Sigma_1 \cap \Sigma_2)^*$$

Note: When $\Sigma_1 = \Sigma_2 = \Sigma$, this represents exact regular language intersection.

Control Interpretation

The product represents **logical conjunction** (“AND” logic): a trace is permissible in $G_1 \times G_2$ if and only if it is fully accepted by both subsystems.

Product Synthesis: Nominal Space vs. Accessible Part

State Space Explosion vs. Lock-Step Pruning

The nominal state space of the product is the full Cartesian set: $X_{\times} = X_1 \times X_2 \implies |X_{\times}| = |X_1| \cdot |X_2|$.

- Computing all pairwise states (x_1, x_2) grows exponentially with the number of sub-components.
- However, because transitions require simultaneous agreement ($\Gamma_{\times} = \Gamma_1 \cap \Gamma_2$), large regions of $X_1 \times X_2$ are **never reached** from $(x_{0,1}, x_{0,2})$.

Algorithmic Focus: The Accessible Part $\text{Ac}(G_1 \times G_2)$

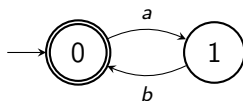
In practice and control synthesis, we only construct the **accessible part**:

$$G_{1 \times 2, ac} := \text{Ac}(G_1 \times G_2) = (X_{ac}, \Sigma, f_{\times}|_{X_{ac}}, \Gamma_{\times}, (x_{0,1}, x_{0,2}), X_m \cap X_{ac})$$

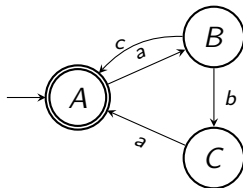
Once $\text{Ac}(G_1 \times G_2)$ is extracted, we apply the diagnostics from Sections 2.2–2.3: testing whether every accessible state is coaccessible (CoAc check) to guarantee the absence of deadlocks.

Benchmark Systems: G_1 and G_2

Automaton G_1



Automaton G_2



Formal Definitions

- $\Sigma_1 = \{a, b\}$, $X_1 = \{0, 1\}$, $X_{m,1} = \{0\}$
- $\Sigma_2 = \{a, b, c\}$, $X_2 = \{A, B, C\}$, $X_{m,2} = \{A\}$
- Common Alphabet: $\Sigma_1 \cap \Sigma_2 = \{a, b\}$
- Exclusive to G_2 : $\Sigma_2 \setminus \Sigma_1 = \{c\}$
- Initial state: $(0, A) \in X_1 \times X_2$
- Marked state: $(0, A) \in X_{m,1} \times X_{m,2}$

Structural Characteristics

- G_1 oscillates between 0 and 1 via the string $(ab)^*$.
- G_2 offers a branching choice from B: either return via c or proceed via $b \rightarrow a$.

Example: Computing the Reachable Product $Ac(G_1 \times G_2)$

Task for students

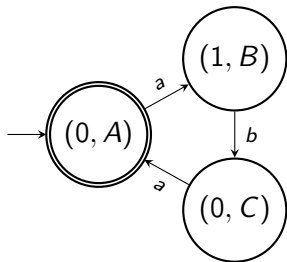
Compute the synchronous product: $G_1 \times G_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{\times}, \Gamma_{\times}, (0, A), X_{m,1} \times X_{m,2})$

Step-by-Step Questions

- 1 **Transition from $(0, A)$:** Evaluate $\Gamma_{\times}(0, A) = \Gamma_1(0) \cap \Gamma_2(A)$. Can event a fire? What is the resulting state?
- 2 **Branching Evaluation at $(1, B)$:**
 - Determine local active sets: $\Gamma_1(1)$ and $\Gamma_2(B)$.
 - Evaluate $\Gamma_{\times}(1, B) = \Gamma_1(1) \cap \Gamma_2(B)$.
 - Can event c fire? Why or why not?
- 3 **Subsequent Evolution:** From the target state, compute the next enabled event and verify if the marked state $(0, A)$ is reached.
- 4 **Nonblocking Property:** Does the reachable subgraph contain deadlocks or livelocks?

Solution: The Reachable Subgraph $Ac(G_1 \times G_2)$

Accessible Graph $Ac(G_1 \times G_2)$



Language Intersection Result

$$L(G_1 \times G_2) = \overline{(aba)^*}, \quad L_m(G_1 \times G_2) = (aba)^*$$

Notice that $aba \in L(G_1) \cap L(G_2)$, while strings containing c are strictly eliminated.

Step-by-Step Transition Analysis

- $\Gamma_{\times}(0, A) = \{a\} \cap \{a\} = \{a\} \implies (1, B)$
- At state $(1, B)$:

$$\Gamma_{\times}(1, B) = \Gamma_1(1) \cap \Gamma_2(B) = \{b\} \cap \{b, c\} = \{b\}$$

Event $c \in \Sigma_2 \setminus \Sigma_1$ is **eliminated**: G_1 cannot synchronize on c ! Target state: $(0, C)$.

- At state $(0, C)$:

$$\Gamma_{\times}(0, C) = \{a\} \cap \{a\} = \{a\} \implies (0, A)$$

System Is Nonblocking

Every accessible state can reach $(0, A) \in X_{m, \times}$.
 $Ac(G_1 \times G_2)$ is a closed periodic orbit.

Parallel Composition: Formal Definition

Mathematical Structure

Given two automata $G_1 = (X_1, \Sigma_1, f_1, \Gamma_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, f_2, \Gamma_2, x_{0,2}, X_{m,2})$, their **parallel composition** is:

$$G_1 \parallel G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{\parallel}, \Gamma_{\parallel}, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

Where for any compound state $(x_1, x_2) \in X_1 \times X_2$ and event $\sigma \in \Sigma_1 \cup \Sigma_2$:

$$f_{\parallel}((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2 \quad (G_1 \text{ moves, } G_2 \text{ idle}) \\ (x_1, f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_2(x_2) \setminus \Sigma_1 \quad (G_2 \text{ moves, } G_1 \text{ idle}) \\ (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \quad (\text{Joint lock-step}) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\Gamma_{\parallel}(x_1, x_2) = [\Gamma_1(x_1) \setminus \Sigma_2] \cup [\Gamma_2(x_2) \setminus \Sigma_1] \cup [\Gamma_1(x_1) \cap \Gamma_2(x_2)]$$

Parallel Composition: Language Properties

Natural Projection Operator

Let $P_i : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$ denote the natural projection filtering out events not in Σ_i :

- The string-level projection $P_i(s)$ erases external actions executed by the partner component.
- The inverse projection $P_i^{-1}(L_i)$ lifts local behavior to the combined alphabet $\Sigma_1 \cup \Sigma_2$, allowing arbitrary interleaving of external events.

Language Intersection Theorem

The languages of the parallel composition are given by lifted set intersection:

$$L(G_1 \parallel G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2)) \quad L_m(G_1 \parallel G_2) = P_1^{-1}(L_m(G_1)) \cap P_2^{-1}(L_m(G_2))$$

Boundary Cases

- **Identical Alphabets** ($\Sigma_1 = \Sigma_2$): $P_1 = P_2 = \text{id} \implies G_1 \parallel G_2 \equiv G_1 \times G_2$.
- **Disjoint Alphabets** ($\Sigma_1 \cap \Sigma_2 = \emptyset$): $L(G_1 \parallel G_2) = L(G_1) \parallel L(G_2)$ (pure asynchronous interleaving).

Parallel Synthesis: Nominal Space vs. Accessible Part

Concurrency vs. Combinatorial Explosion

Nominal state cardinality remains identical to the product: $|X_{\parallel}| = |X_1| \cdot |X_2|$

- Due to asynchronous interleaving, parallel composition typically reaches a **substantially larger subset** of $X_1 \times X_2$ than the synchronous product.
- Asynchronous branches create hypercube / diamond structures in the state space.

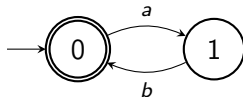
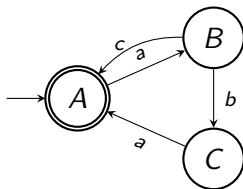
Algorithmic Computation: $\text{Ac}(G_1 \parallel G_2)$

Synthesis algorithms never enumerate unreachable elements of $X_1 \times X_2$:

$$G_{1\parallel 2,ac} := \text{Ac}(G_1 \parallel G_2) = (X_{ac}, \Sigma_1 \cup \Sigma_2, f_{\parallel}|_{X_{ac}}, \Gamma_{\parallel}, (x_{0,1}, x_{0,2}), X_m \cap X_{ac})$$

- **On-the-Fly Forward Exploration:** Rooted at $(x_{0,1}, x_{0,2})$ using BFS or DFS.
- **Successor Generation:** For each explored state, successors are queued via: (i) active private events of G_1 , (ii) active private events of G_2 , (iii) enabled rendezvous events $\sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$.
- **Structural Diagnostics:** After computing $\text{Ac}(G_1 \parallel G_2)$, nonblocking verification (CoAc test) is strictly required to detect deadlock or livelock states introduced by asynchronous races.

Parallel Composition Benchmark: $G_1 \parallel G_2$

Automaton G_1 Automaton G_2 

System Specifications

- $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{a, b, c\}$
- Shared Alphabet: $\Sigma_1 \cap \Sigma_2 = \{a, b\}$ (require rendezvous)
- Exclusive Event: $\Sigma_2 \setminus \Sigma_1 = \{c\}$ (G_2 moves alone!)
- Initial state: $(0, A) \in X_1 \times X_2$
- Marked state: $(0, A) \in X_{m,1} \times X_{m,2}$

Key Contrast with Product

In $G_1 \times G_2$, event c was strictly forbidden because $c \notin \Sigma_1$. In $G_1 \parallel G_2$, rule (ii) applies:

$$c \in \Gamma_2(x_2) \setminus \Sigma_1 \implies (x_1, x_2) \xrightarrow{c} (x_1, f_2(x_2, c))$$

Example: Computing the Accessible Part $\text{Ac}(G_1 \parallel G_2)$

Task for students

Compute the accessible automaton $\text{Ac}(G_1 \parallel G_2)$ starting at initial state $(0, A)$:

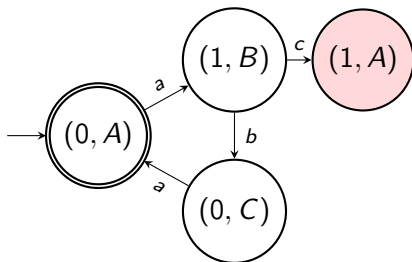
$$G_1 \parallel G_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{\parallel}, \Gamma_{\parallel}, (0, A), X_{m,1} \times X_{m,2})$$

Analytical Questions

- 1 **Transition from $(0, A)$:** Since a is shared, does $f_{\parallel}((0, A), a)$ behave identically to the product?
- 2 **Branching at State $(1, B)$:**
 - Evaluate $\Gamma_{\parallel}(1, B)$ using the 3-case rule. Which events are enabled?
 - Where does the private transition on c lead?
 - Where does the synchronized transition on b lead?
- 3 **Exploration from New State $(1, A)$:**
 - At state $(1, A)$, we have $\Gamma_1(1) = \{b\}$ and $\Gamma_2(A) = \{a\}$.
 - Since both a and b are shared ($\{a, b\} \subseteq \Sigma_1 \cap \Sigma_2$), can either fire?
 - What kind of state is $(1, A)$?
- 4 **Synthesize $\text{Ac}(G_1 \parallel G_2)$:** Draw the resulting graph and verify coaccessibility.

Solution Part 1: Graph Construction & Transition Steps

Accessible Graph $\text{Ac}(G_1 \parallel G_2)$



Step-by-Step Transition Analysis

- From Root $(0, A)$ on shared event a :

$$(0, A) \xrightarrow{a} (1, B) \quad \text{via lock-step } (a \in \Sigma_1 \cap \Sigma_2)$$

- Branching at $(1, B)$ via 3-branch rule:

$$\Gamma_{\parallel}(1, B) = \underbrace{(\{b\} \cap \{b, c\})}_{\text{shared: } b} \cup \underbrace{(\{b, c\} \setminus \Sigma_1)}_{\text{private to } G_2: c} = \{b, c\}$$

- On b (synchronized): $(1, B) \xrightarrow{b} (0, C)$
- On c (G_2 moves alone): $(1, B) \xrightarrow{c} (1, A)$

- Orbit Closure from $(0, C)$:

$$\Gamma_{\parallel}(0, C) = \{a\} \cap \{a\} = \{a\} \implies (0, A)$$

Solution Part 2: Deadlock Discovery & Structural Comparison

Discovery: A Deadlock State at $(1, A)$

State $(1, A)$ has $\Gamma_{\parallel}(1, A) = \emptyset$, meaning it is a sink state (deadlock):

$$\Gamma_1(1) \cap \Gamma_2(A) = \{b\} \cap \{a\} = \emptyset$$

- Both components require shared events, but neither enables the other's request.
- Since neither system offers an autonomous private event, no transition can fire.
- **Core Diagnostic:** State $(1, A)$ is accessible but not coaccessible $\implies G_1 \parallel G_2$ is **blocking**.

Structural Comparison: $G_1 \times G_2$ vs. $G_1 \parallel G_2$

- **Synchronous Product** ($G_1 \times G_2$): 3 accessible states. Nonblocking (periodic orbit). The strict lock-step rule artificially pruned the private branch on event c .
- **Parallel Composition** ($G_1 \parallel G_2$): 4 accessible states. **Blocking** due to the autonomous branch c leading into deadlock state $(1, A)$.
- **Marked Languages:** Both generate $L_m = (aba)^*$, but parallel composition admits the deadlocking prefix trace $ac \notin \overline{L_m}$.

Direct Comparison: $G_1 \times G_2$ vs. $G_1 \parallel G_2$

Dimension	Synchronous Product ($G_1 \times G_2$)	Parallel Composition ($G_1 \parallel G_2$)
Core Semantics	Total lock-step synchronization	Rendezvous on shared; interleaving on private
Active Set $\Gamma(x_1, x_2)$	$\Gamma_1(x_1) \cap \Gamma_2(x_2)$	$(\Gamma_1 \setminus \Sigma_2) \cup (\Gamma_2 \setminus \Sigma_1) \cup (\Gamma_1 \cap \Gamma_2)$
Private Events	Permanently disabled in product	Executed autonomously by local subsystem
Generated Language	$L(G_1) \cap L(G_2) \subseteq (\Sigma_1 \cap \Sigma_2)^*$	$P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2)) \subseteq (\Sigma_1 \cup \Sigma_2)^*$
State Space Size	Typically smaller: $ X_{ac} \leq X_1 \cdot X_2 $	Typically larger: explores concurrency paths

Key Takeaways for System Modeling

- **Consistency Check:** When alphabets coincide ($\Sigma_1 = \Sigma_2$), the two operations collapse to the same automaton: $G_1 \parallel G_2 \equiv G_1 \times G_2$.
- **The Concurrency Trade-Off:** Parallel composition faithfully captures autonomous subsystem dynamics, but private branching can introduce unexpected coordination deadlocks (as demonstrated by trace *ac*).

Part 2: Key Takeaways

Structural Pruning: Ac, CoAc, and Trim

- $\text{Ac}(G)$: States reachable from x_0 (discards unreachable states).
- $\text{CoAc}(G)$: States that can reach X_m (discards deadlocks and livelocks).
- $\text{Trim}(G) = \text{Ac}(G) \cap \text{CoAc}(G)$: System is nonblocking $\iff \text{Ac}(G) = \text{Trim}(G)$ (i.e., $\overline{L_m(G)} = L(G)$).

System Interconnection: $\times$ vs. $\parallel$

- **Product** ($G_1 \times G_2$): Lock-step on all events ($\Gamma_1 \cap \Gamma_2$). Private events are deleted. Yields language intersection: $L_1 \cap L_2$.
- **Parallel** ($G_1 \parallel G_2$): Synchronous on shared events ($\Sigma_1 \cap \Sigma_2$), asynchronous interleaving on private events. Yields lifted intersection: $P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$.

The Core Lesson

Even if G_1 and G_2 are both trim, their composition ($G_1 \times G_2$ or $G_1 \parallel G_2$) may **lose coaccessibility**. We must always compute Ac on-the-fly and test for Trim to detect blocking.

What Comes Next? Part 3: Supervisory Control

From Open-Loop Modeling to Closed-Loop Control

Compositions can generate blocking states. We need an external agent to restrict plant behavior (G) and guarantee safety and liveness: the **Supervisor** (S)

$$S/G := G \parallel S$$

Core Challenges in Part 3

In realistic systems, the supervisor has physical limitations:

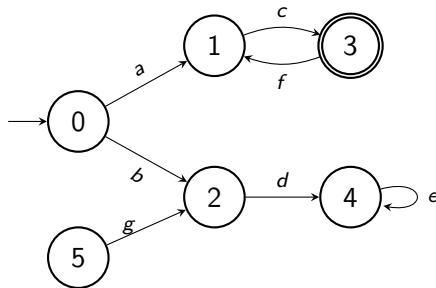
- **Actuator Limits (Uncontrollability):** $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. The supervisor cannot prevent events in Σ_{uc} from occurring (e.g., sensor readings or failures).
- **Sensor Limits (Unobservability):** $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. The supervisor cannot see events in Σ_{uo} and must estimate state trajectories (e.g., failures).
- **Synthesis Goal:** Find a supervisor that enforces specifications, avoids blocking, and is **minimally restrictive** (supC).

Part 2: Theoretical Review Questions

- 1 **Accessible and Coaccessible Parts:** Define formally the operators $Ac(G)$ and $CoAc(G)$. In terms of state-space search algorithms (forward vs. backward), how are they computed?
- 2 **Trim Automata and Nonblocking Condition:** Define a Trim automaton. Formally state the condition under which an automaton G is nonblocking, both in terms of generated/marked languages and using the $CoAc$ operator.
- 3 **Deadlock vs. Livelock Diagnostics:** Explain the structural difference between a deadlock state and a livelock cycle in an accessible automaton. Why do both configurations violate the condition $Ac(G) = Trim(G)$?
- 4 **Synchronous Product Execution Mechanism:** Define $G_1 \times G_2$ and its active event set $\Gamma_{\times}(x_1, x_2)$. Why are private events $(\Sigma_i \setminus \Sigma_j)$ permanently suppressed in the synchronous product?
- 5 **Parallel Composition via Projections:** Define the transition function $f_{\parallel}$ of $G_1 \parallel G_2$ across its three distinct cases. How is the generated language $L(G_1 \parallel G_2)$ expressed via the natural projection operators P_1 and P_2 ?
- 6 **Preservation of Structural Properties under Composition:** If G_1 and G_2 are both trim and nonblocking, is $G_1 \parallel G_2$ guaranteed to be nonblocking? Justify your answer referring to reachable shared vs. private event mismatches.

Practice Exercise 1: Structural Pruning & Nonblocking Test

Automaton G



System Formal Data

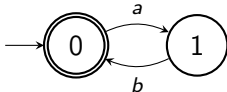
- $\Sigma = \{a, b, c, d, e, f, g\}$, $X = \{0, 1, 2, 3, 4, 5\}$
- Initial state: $x_0 = 0$, Marked states: $X_m = \{3\}$

Questions

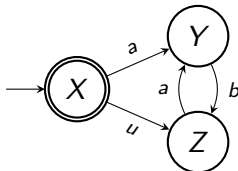
- 1 Accessible Part:** Compute $\text{Ac}(G)$. Which state(s) in X are pruned and why?
- 2 Coaccessible Part:** Compute $\text{CoAc}(G)$. Identify any deadlock states or livelock loops.
- 3 Trim Extraction:** Construct $\text{Trim}(G) = \text{Ac}(G) \cap \text{CoAc}(G)$.
- 4 Nonblocking Verdict:** Is G nonblocking? Formally justify using the conjecture $\overline{L_m(G)} \stackrel{?}{=} L(G)$.

Practice Exercise 2: Synchronous Product Computation

Automaton G_1



Automaton G_2



System Alphabets & States

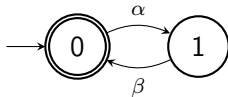
- $\Sigma_1 = \{a, b\}$, $X_1 = \{0, 1\}$, $X_{m,1} = \{0\}$
- $\Sigma_2 = \{a, b, u\}$, $X_2 = \{X, Y, Z\}$, $X_{m,2} = \{X\}$
- Initial compound state: $(0, X)$
- Marked compound set: $X_{m,\times} = \{(0, X)\}$

Questions

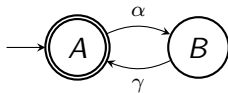
- 1 Active Event Sets:** What is the status of event $u \in \Sigma_2 \setminus \Sigma_1$ in the synchronous product $G_1 \times G_2$? Can it ever fire?
- 2 Accessible Subgraph:** Compute $Ac(G_1 \times G_2)$ by forward reachability from $(0, X)$. Draw the resulting transition graph.
- 3 Language Analysis:** Explicitly write $L_m(G_1 \times G_2)$ and confirm that $L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$.

Practice Exercise 3: Parallel Composition & Coordination Hazard

Automaton G_1



Automaton G_2



System Alphabets & States

- $\Sigma_1 = \{\alpha, \beta\}, \quad \Sigma_2 = \{\alpha, \gamma\}$
- Shared Alphabet: $\Sigma_1 \cap \Sigma_2 = \{\alpha\}$
- Private Alphabets: $\beta \in \Sigma_1 \setminus \Sigma_2, \quad \gamma \in \Sigma_2 \setminus \Sigma_1$
- Initial state: $(0, A), \quad$ Marked set: $\{(0, A)\}$

Questions

- 1 **Transition Evaluation:** From state $(1, B)$, evaluate $\Gamma_{\parallel}(1, B)$ using the 3-case rule. Can events β and γ fire independently?
- 2 **Forward State Exploration:** Determine all states belonging to $\text{Ac}(G_1 \parallel G_2)$ and draw the complete accessible graph.
- 3 **Liveness & Blocking Analysis:** Is $\text{Ac}(G_1 \parallel G_2)$ nonblocking? If blocking occurs, identify the critical state and describe the synchronization conflict.