
MODELING

Complex systems are built from **components** that are **connected** in some way

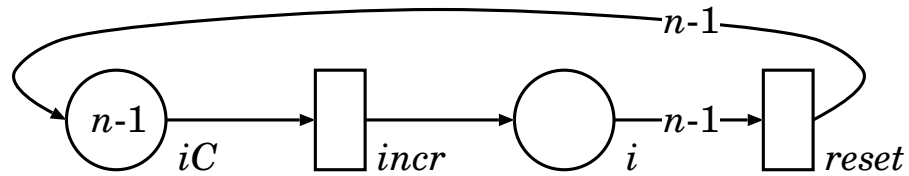
Analogously, their models are built **compositionally** using **synchronization mechanisms**

The simplest way to compose two automata A_1 and A_2 is to build the **product automaton**

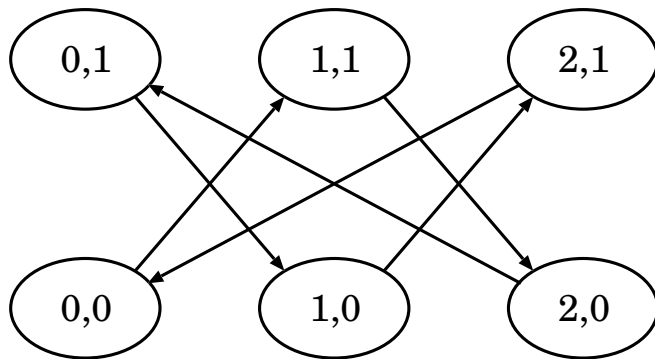
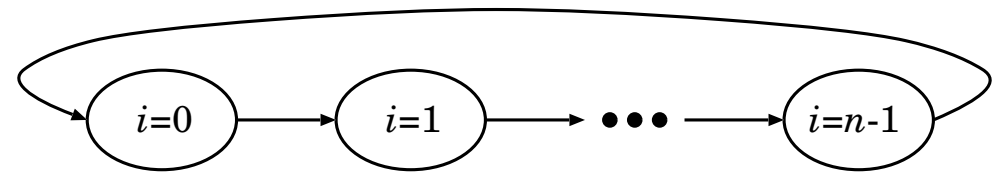
- **synchronous product automaton** $A_{synchron}$:
 - each move of $A_{synchron}$ corresponds to one move of A_1 **and** one move of A_2
 - to allow A_1 to “remain still” in state s_1 while A_2 progresses, add a transition $s_1 \rightarrow s_1$ to A_1
- **asynchronous product automaton** $A_{asynchron}$:
 - each move of $A_{asynchron}$ corresponds to **either** one move of A_1 **or** one move of A_2
- If A_1 has n_1 and A_2 has n_2 states, how many states do $A_{synchron}$ and $A_{asynchron}$ have?

Example: product of modulo- n counters

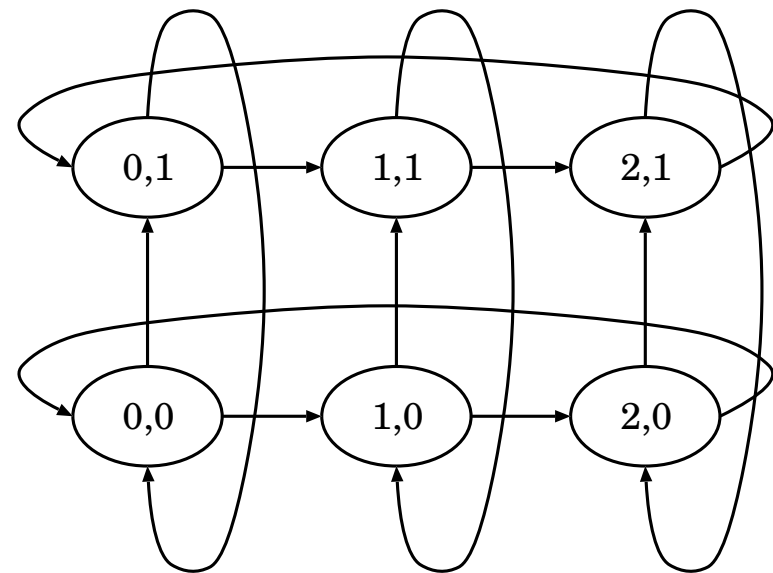
A Petri net modeling a modulo- n counter



and its automaton



A_{synch}



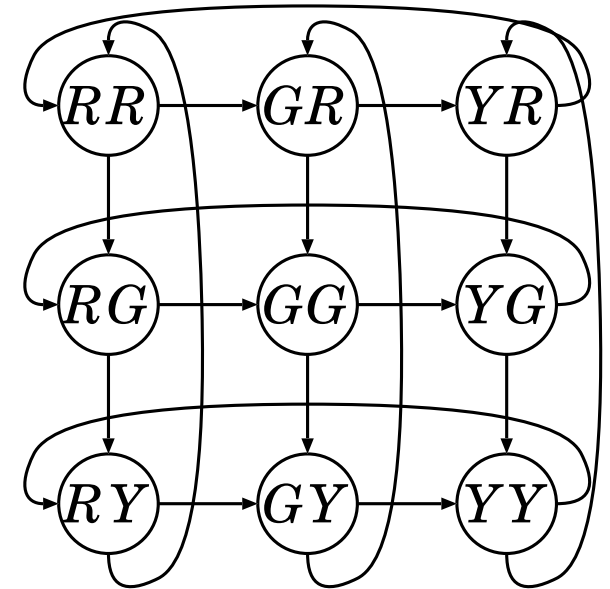
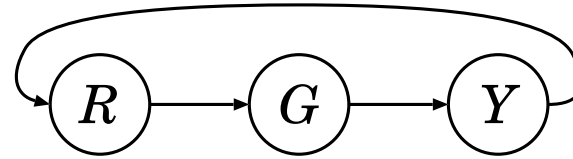
A_{asynch}

If A_1 is a modulo-3 counter and A_2 is a modulo-2 counter, both A_{synch} and A_{asynch} have 6 states

What if A_1 is a modulo-4 counter and A_2 is a modulo-2 counter?

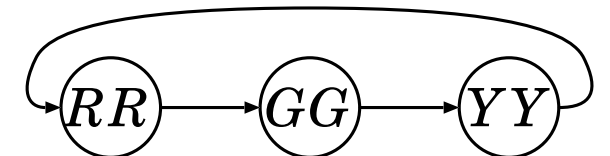
General parallel composition of automata

A traffic light A iteratively cycles between R , G , and Y



The asynchronous product of two traffic lights is not very interesting

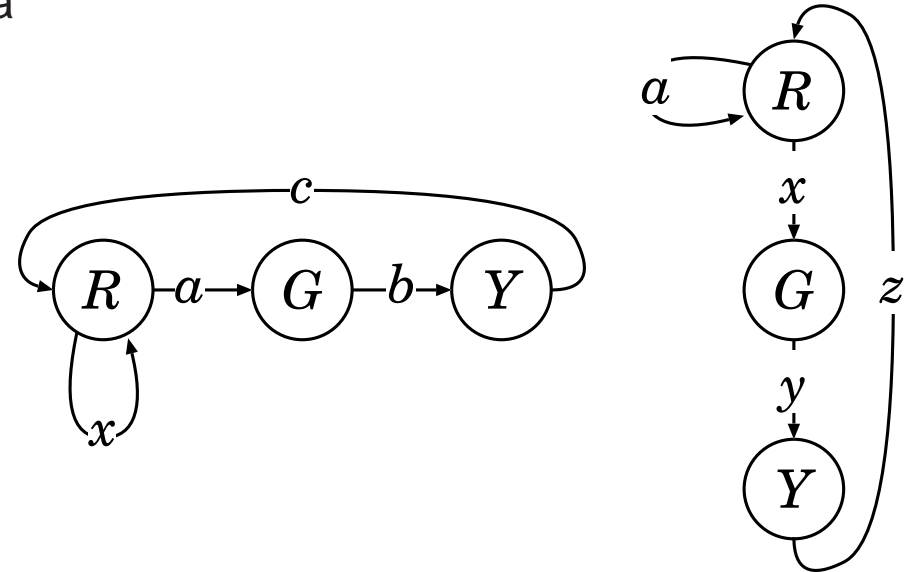
The synchronous product of two traffic lights is even less interesting



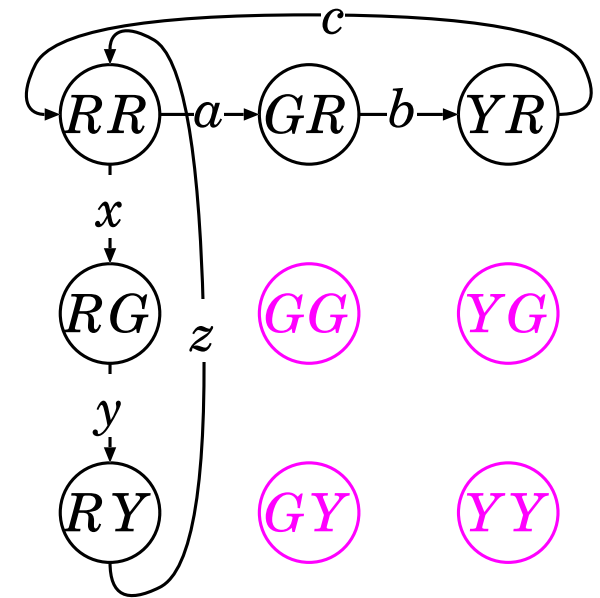
General parallel composition of automata (cont.)

What if the two traffic light are related, e.g., north-south and east-west directions at an intersection?

We need a more flexible way to compose their automata



Attach **labels** to the transitions (events) of the two lights



Perform **parallel composition** synchronizing on common labels (a and x)

Given L automata $A_k = (\mathcal{X}_k, \mathcal{E}_k, \{\mathcal{N}_{k,\alpha} : \alpha \in \mathcal{E}_k\}, \mathcal{X}_{k,init})$ for $k = L, \dots, 1$ where

- \mathcal{X}_k is the (potential) state space of A_k
- \mathcal{E}_k is the set of events for A_k
- $\mathcal{N}_{k,\alpha} : \mathcal{X}_k \rightarrow 2^{\mathcal{X}_k}$ is the next-state function for event $\alpha \in \mathcal{E}_k$ in A_k
- $\mathcal{X}_{k,init} \subseteq \mathcal{X}_k$ is the set of initial states for A_k

their **parallel composition** is $A = (\mathcal{X}, \mathcal{E}, \{\mathcal{N}_\alpha : \alpha \in \mathcal{E}\}, \mathcal{X}_{init}) = A_L || \dots || A_1$ where

- $\mathcal{X} = \mathcal{X}_L \times \dots \times \mathcal{X}_1$ is the (potential) state space of A
- $\mathcal{E} = \mathcal{E}_L \cup \dots \cup \mathcal{E}_1$ is the set of events for A_k
- $\mathcal{N}_\alpha : \mathcal{X} \rightarrow 2^{\mathcal{X}}$ is the next-state function for event $\alpha \in \mathcal{E}$ in A
- $\mathcal{X}_{init} = \mathcal{X}_{L,init} \times \dots \times \mathcal{X}_{1,init}$ is the set of initial states for A_k

and

$$\forall (i_L, \dots, i_1) \in \mathcal{X}, (i'_L, \dots, i'_1) \in \mathcal{N}_\alpha(i_L, \dots, i_1)$$

$$\Leftrightarrow$$

$$\forall k \in \{L, \dots, 1\}, \alpha \in \mathcal{E}_k \Rightarrow i'_k \in \mathcal{N}_{k,\alpha}(i_k) \quad \wedge \quad \alpha \notin \mathcal{E}_k \Rightarrow i'_k = i_k$$

Informally, the parallel composition automaton is obtained by performing

- synchronous products on events shared by multiple A_k 's
- asynchronous products events local to a single A_k

This assumes that the event names **names** in the A_k have global meaning

A more flexible approach is to publish **automata interfaces**, for example

- $A_1(\alpha, \beta, \gamma)$ means that A_1 can synchronize on events α , β , and γ
- Only α , β , and γ are **visible**, all its other events are **hidden** or **internal**
- Parallel composition of A_1 and A_2 where we synchronize event α of A_1 with event λ of A_2 and event γ of A_1 with event μ of A_2 :

$$A = A_1(\alpha, \beta, \gamma) \parallel_{\alpha \equiv \lambda, \gamma \equiv \mu} A_2(\lambda, \mu)$$

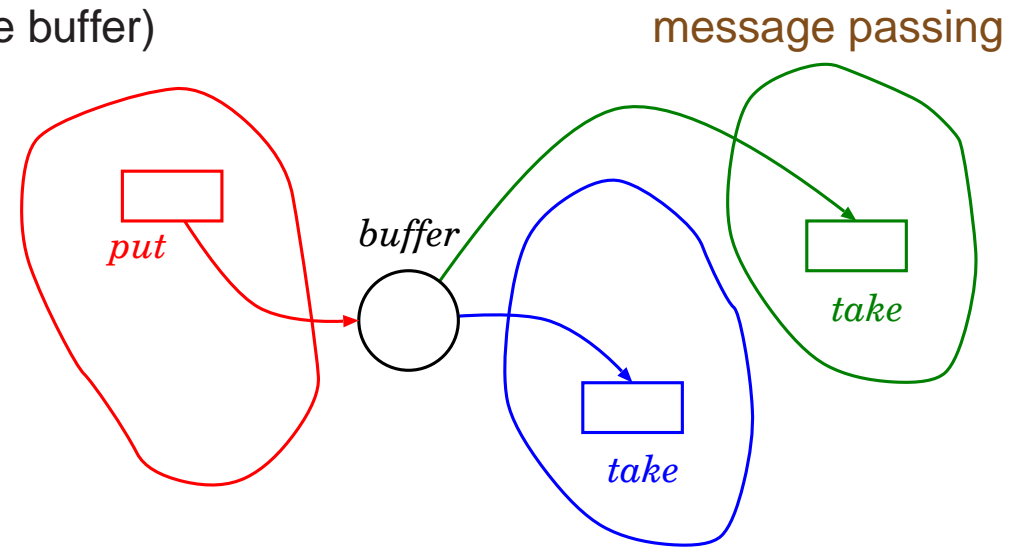
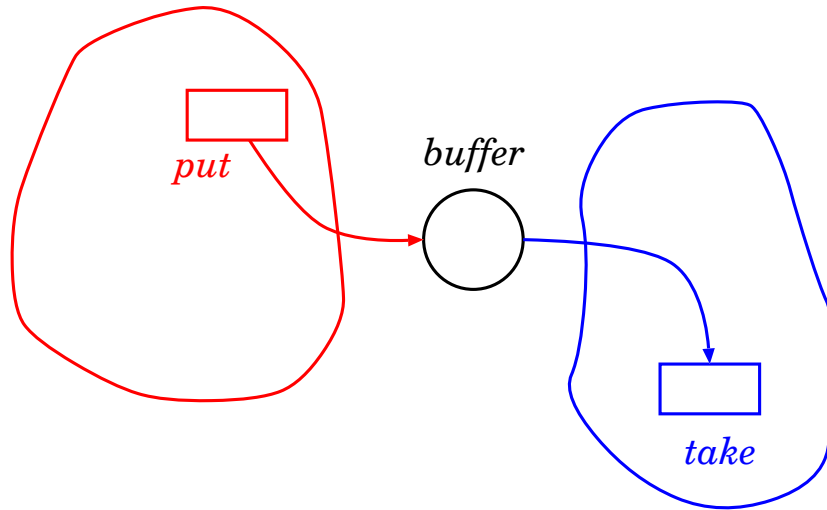
- The interface of A can be an arbitrary subset of the three events $\alpha \equiv \lambda$, β , and $\gamma \equiv \mu$
- It might be convenient to **rename** events published in the interface:

$$A(x : \alpha \equiv \lambda, y : \beta, z : \gamma \equiv \mu) = A_1(\alpha, \beta, \gamma) \parallel_{\alpha \equiv \lambda, \gamma \equiv \mu} A_2(\lambda, \mu)$$

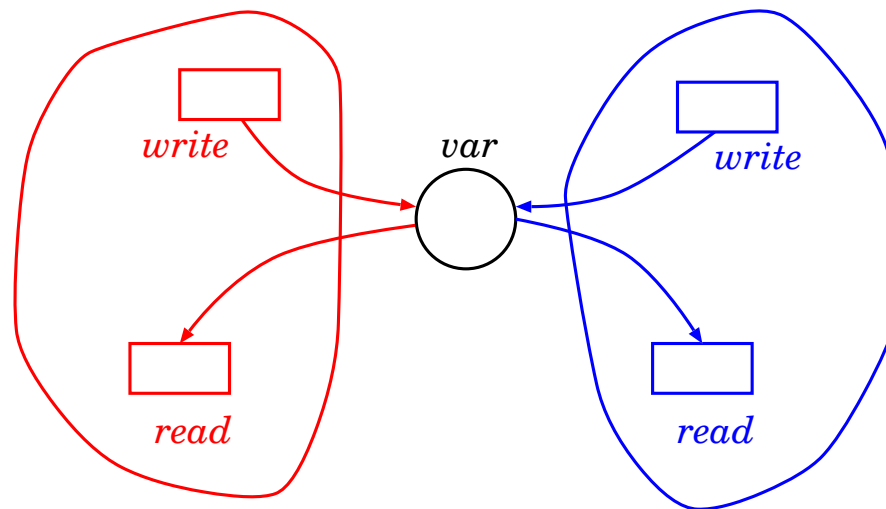
Modeling communications or synchronizations between m sequential software processes

- Asynchronous communication:

buffers (processes either **put** in or **take** from the buffer)

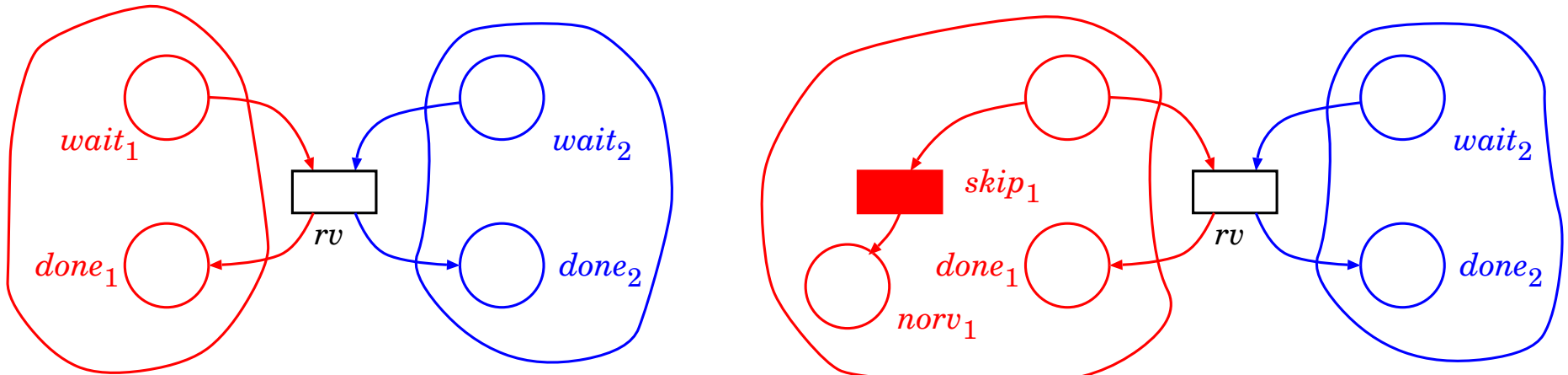


shared variables (enforce reader-writer and writer-writer mutual exclusion)

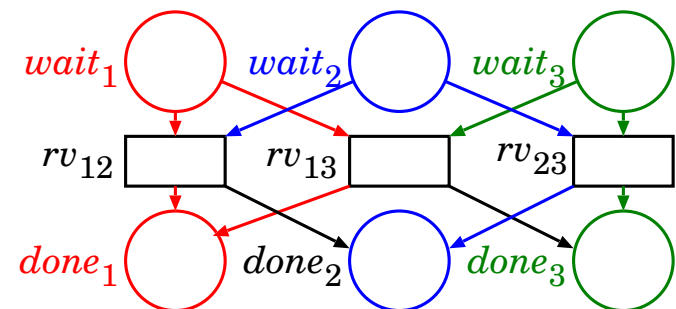
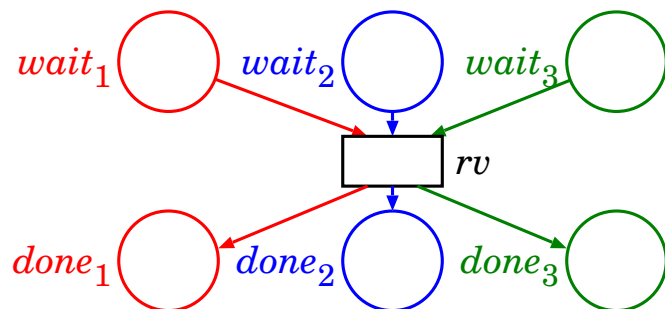


shared memory

- Synchronous communication: pairwise rendezvous
 - blocking: process A_1 arrives, waits for process A_2 if it is not there waiting
 - non-blocking: process A_1 arrives, does not wait for process A_2 if it is not there waiting
 - timeout: process A_1 arrives, waits no more than τ time for process A_2 if it is not there waiting



- Synchronous communication: synchronization barrier (generalization of rendezvous)
 - all m existing processes must synchronize \Rightarrow one rendezvous transition
 - k -out-of- m existing processes must synchronize $\Rightarrow m$ -choose- k rendezvous transitions



Other systems are best described using a variable number of **threads**

An example is systems exhibiting **fork-and-join** behavior

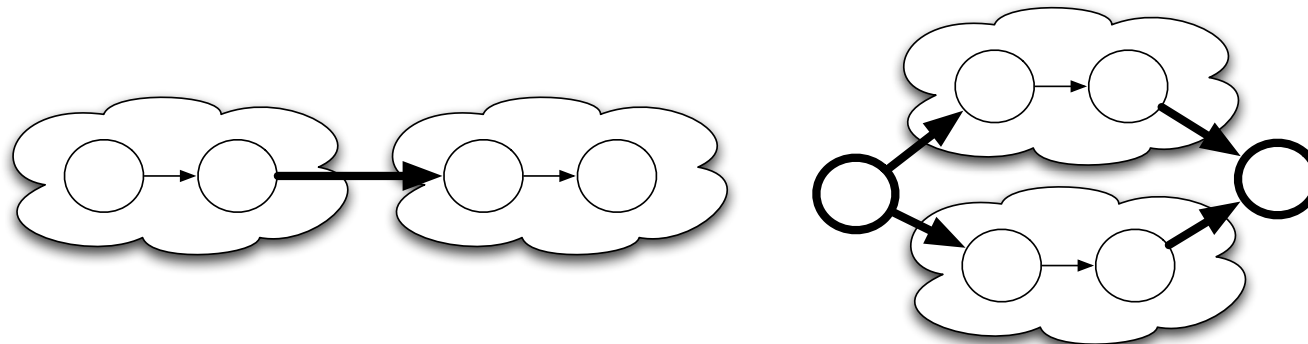
- At some point a thread can **fork** itself into n threads (not necessarily identical)
- Later on these n threads **join** back into a single thread

Such fork-and-join behavior can be modeled as a **series-parallel** system

This is a special case of **precedence graph**

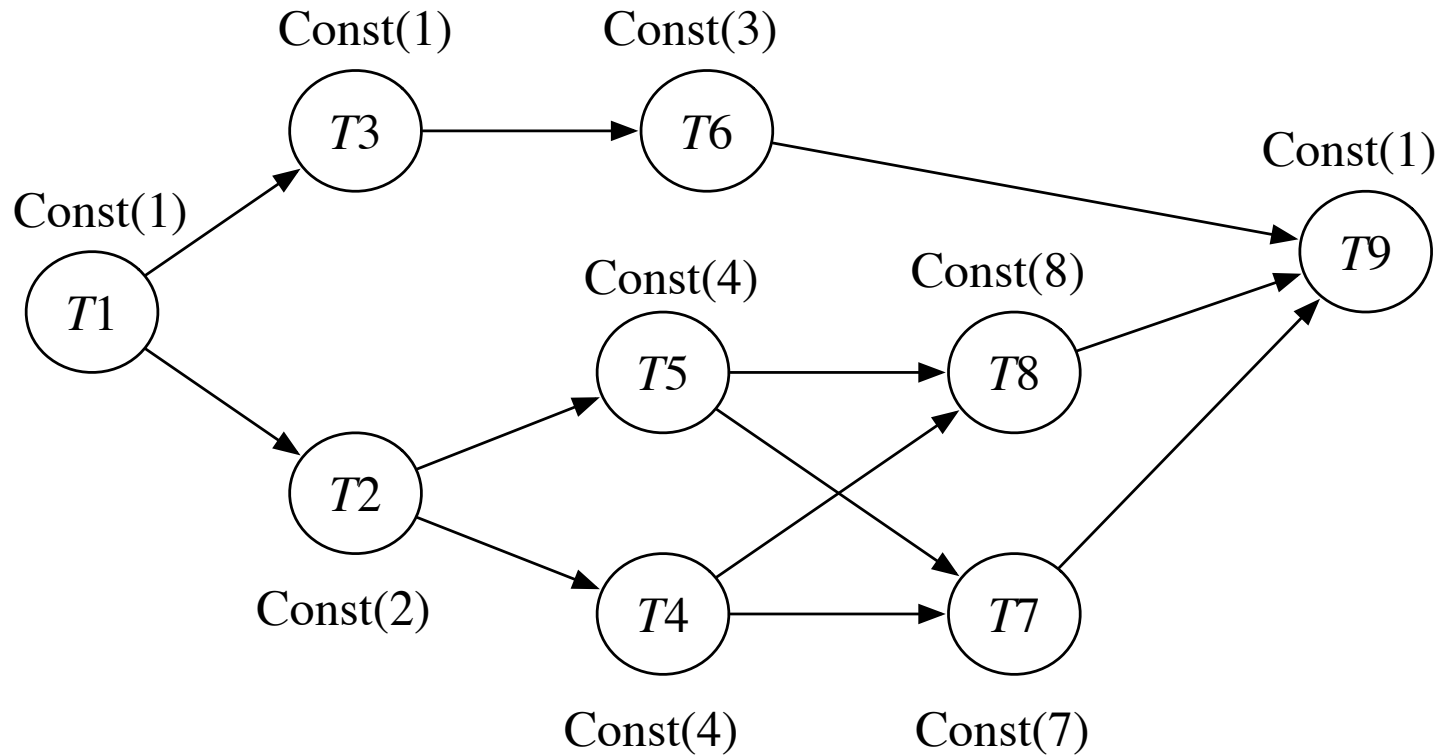
- A directed acyclic graph with a single **source** and a single **node**
- If there is an arc from a to b , then task b can start only after task a completes
- Useful to describe a set of constrained activities, possibly performed by limited resources

A S-P graph is a precedence graph built as a sequential or parallel composition of S-P graphs



A series-parallel task system (description)

A set of tasks with the following precedence constraints and timing (in seconds):

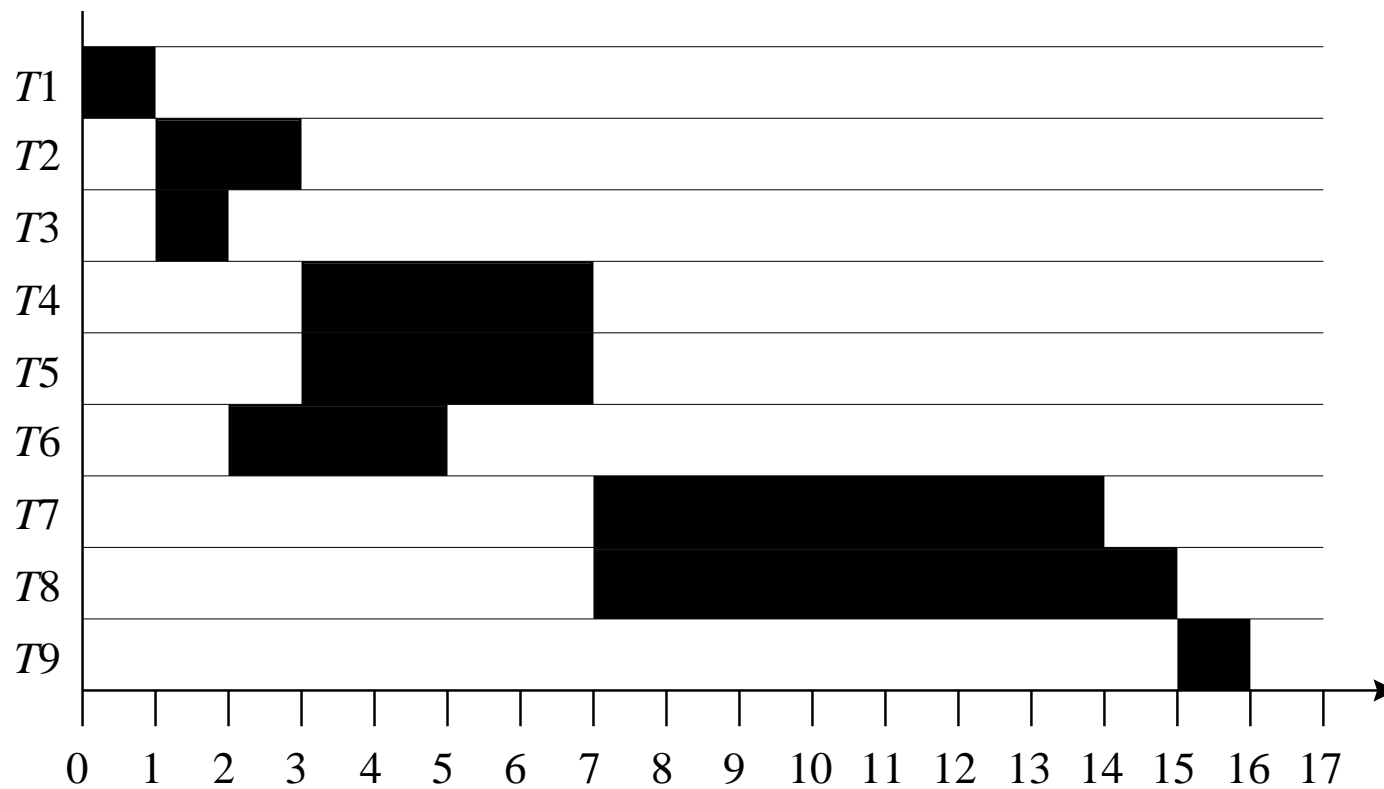


An arc from T_i to T_j means that T_j can be started only after completing T_i

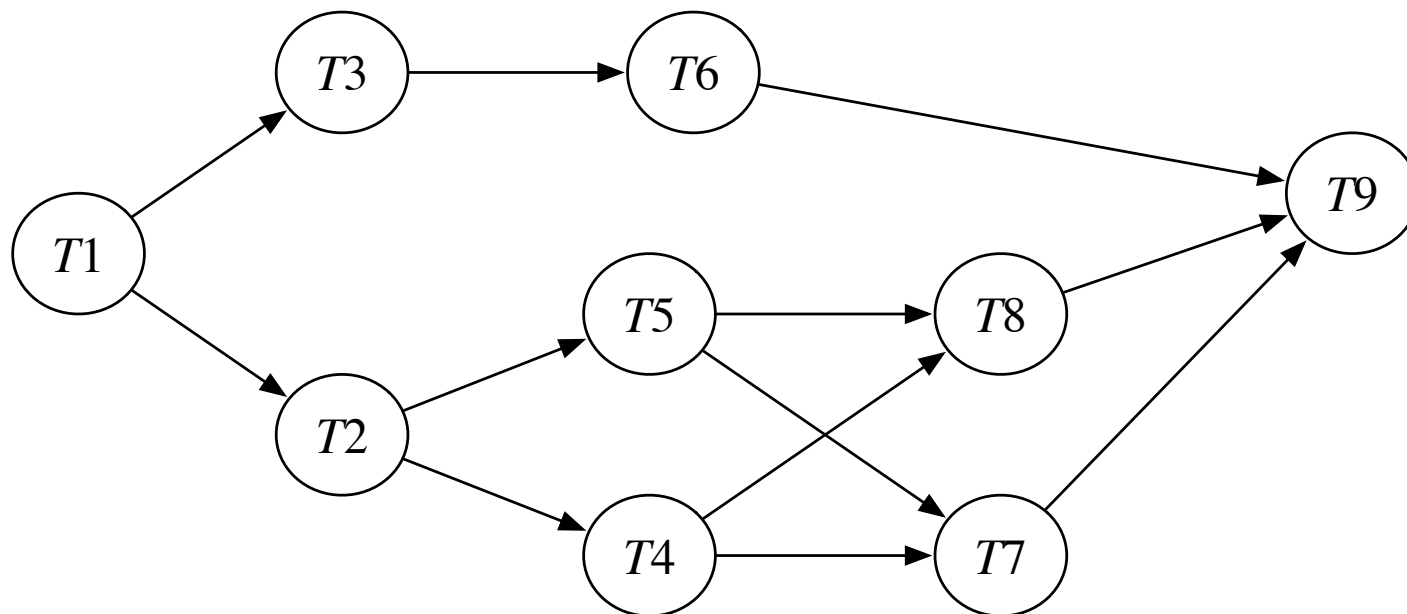
Study the response time for the task system (time from the start of T_1 to the end of T_9)

A series-parallel task system (deterministic timing)

If all durations are constant, the response time, also a constant, is 16 seconds



A series-parallel task system (arbitrary timing)

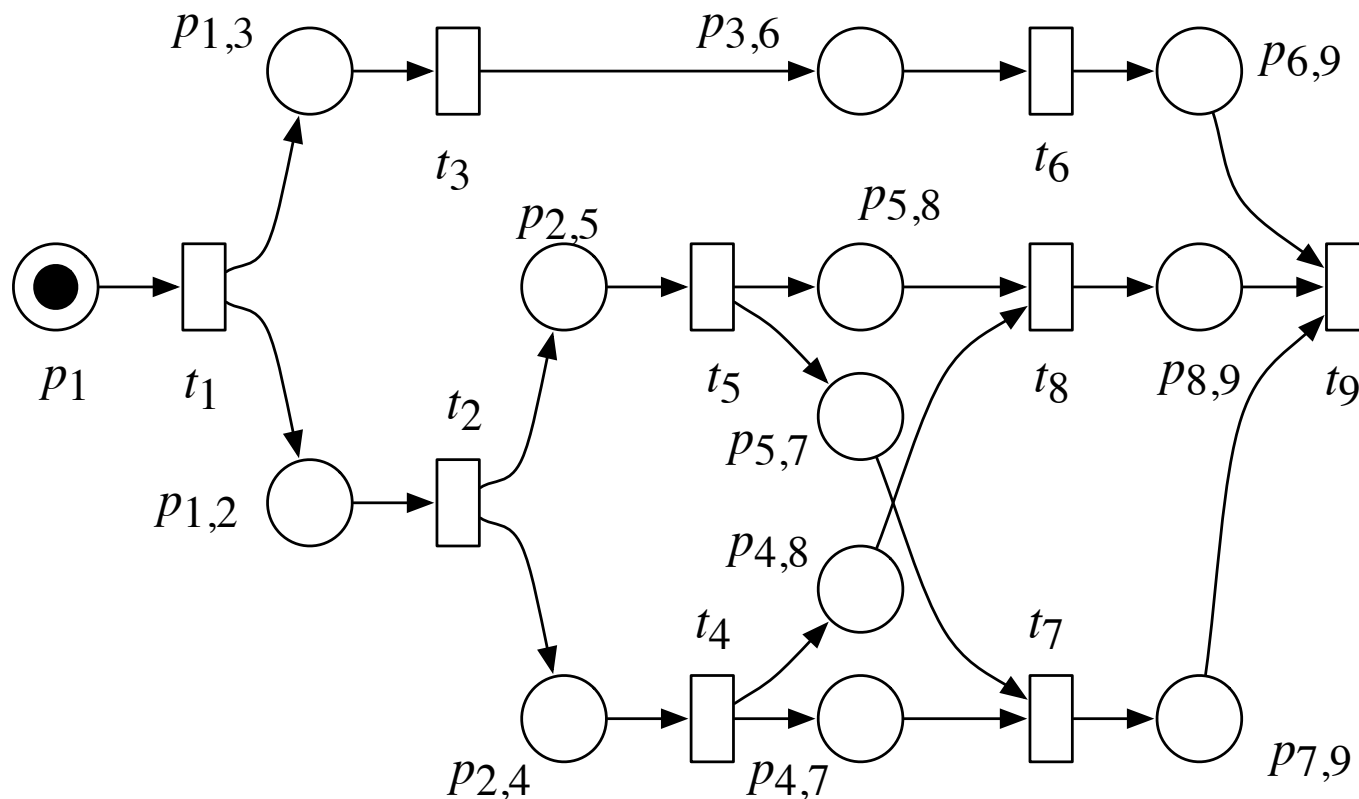


What are the possible sets of concurrently executing tasks?

- $\{T1\}$,
- $\{T2, T3\}$,
- $\{T2, T6\}$,
- $\{T3, T4, T5\}$,
- ...

A series-parallel task system (Petri net model)

- For each T_i : add event t_i
- For each T_i with no antecedents: add place p_i , with one initial token, and an arc $p_i \rightarrow t_i$
- For each $T_i \rightarrow T_j$: add place $p_{i,j}$, with no tokens in it, and arcs $t_i \rightarrow p_{i,j}$ and $p_{i,j} \rightarrow t_j$



The task system completes when all the places are empty (the only absorbing state)