Formal Methods for Dynamic Systems

殷翔

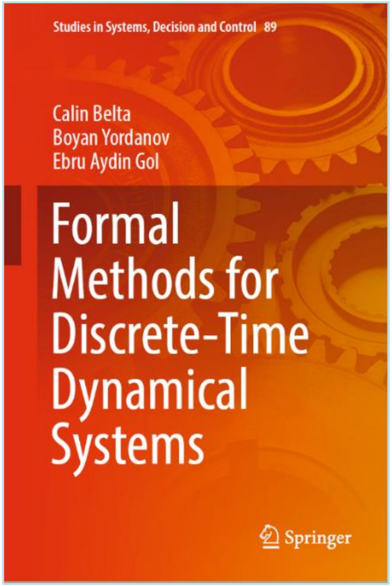
上海交通大学 自动化系

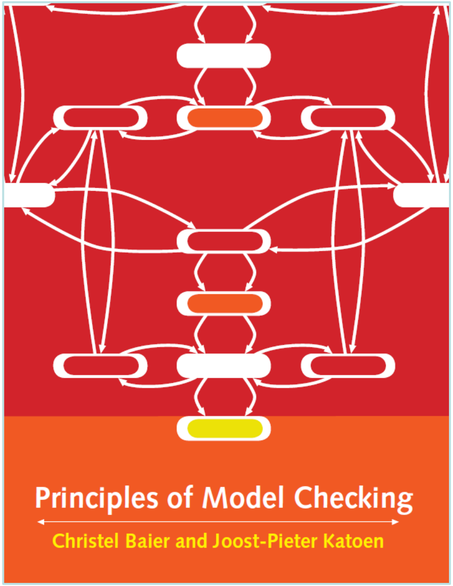
yinxiang@sjtu.edu.cn

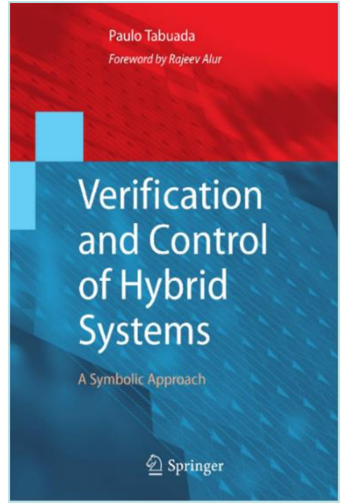
http://xiangyin.sjtu.edu.cn

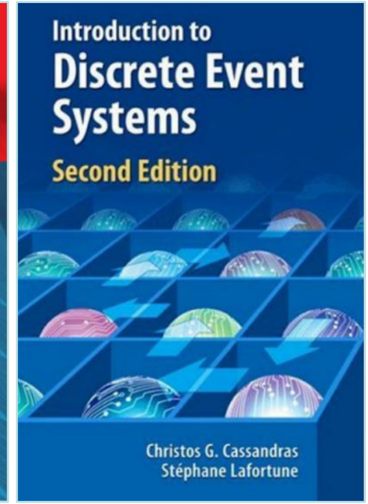
2021.07 厦门大学暑期课程

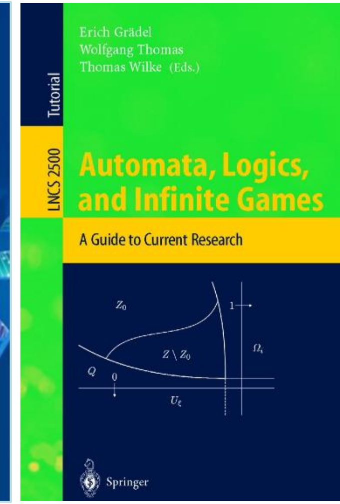
References











No required textbook, but you can read:

- **"Formal Methods for Discrete-Time Dynamical Systems", by Belta, Yordanov & Gol**
- **"Principles of Model Checking", by Baier & Katoen**
- **"Verification and Control of Hybrid Systems: A Symbolic Approach", by Tabuada**
- **"Introduction to Discrete Event Systems", by Cassandras & Lafortune**
- **"Automata, Logics and Infinite Games", by Gradel, Thomas & Wilke (Eds.)**

Introduction

What is a Dynamic System

$$\mathcal{D}: \begin{cases} x_{t+1} = f(x_t, u_t, w_t) \\ y_t = g(x_t, u_t, v_t) \end{cases}$$

discrete-time dynamic system

$$\mathcal{D}: \begin{cases} \dot{x}(t) = f(x(t), u(t), w(t)) \\ y(t) = g(x(t), u(t), v(t)) \end{cases}$$

continuous-time dynamic system

- The system has a **state-space** that contains possibly infinite states
- The evolution of states is determined by a **dynamic function**
- The dynamic can be both physical laws or logic rules

Aircrafts

Robotics

Circuits

Program

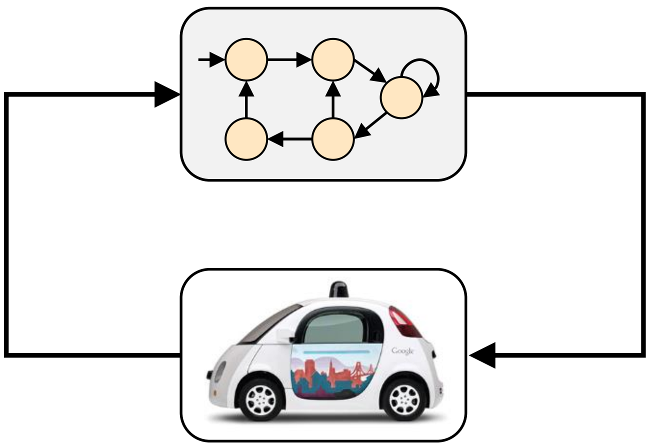
Dynamic Systems

- **Classical analysis and design objectives for dynamic systems**
  - ➢ **controllability and observability analysis**
  - ➢ **stabilization and reference tracking**
  - ➢ **optimal control**

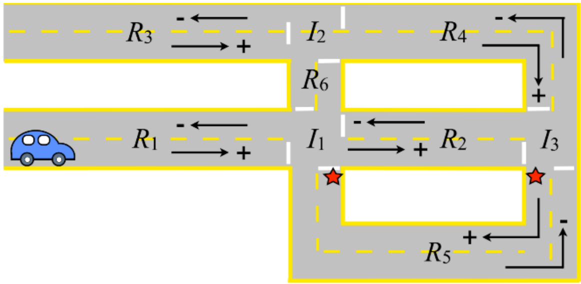
- **They are all about solving the differential equations and are about the "lower-level" dynamics of the system**

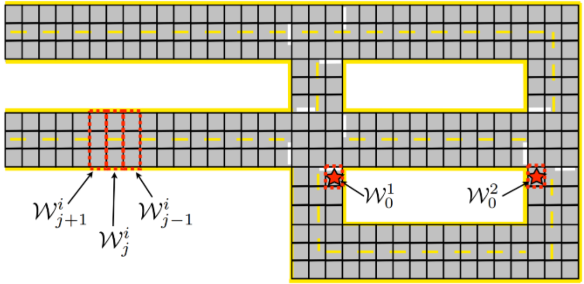
- **More dynamic systems are hybrid and the design objectives are more complicated and intelligent at the "high-level"**
  - ➢ **safety: avoid some logic error**
  - ➢ **liveness: eventually accomplish some tasks**
  - ➢ **security: some crucial information is not released**

Autonomous Vehicles



$$\dot{x} = v, \dot{v} = a - kv^2$$





- There are many low-level point-to-point navigation/control algorithms
- Autonomous vehicles are also facing high-level decision/planning problems
- Go to region A before visiting region B
- Visit region A infinitely often without reaching region B
- React to dynamic environment and uncertainties: if see A, then do B

Picture From: Wongpiromsarn,Topcu & Murray, IEEE TAC, 2012

Cyber-Physical Systems (CPS)

Cyber-physical systems (CPS) integrate sensing, computation, control and networking into physical objects and infrastructure, connecting them to the internet and to each other. ---(NSF US)


Big Data


Cloud Computing


Internet of Things

Cyber-Layer：
Perception,
Control, Decision,
Computation

Cyber World + Physical World


Industry Process


Energy Systems


Transportation

Physical-Layer：
Physic Objects,
ODE Dynamics

Need Formal Methods

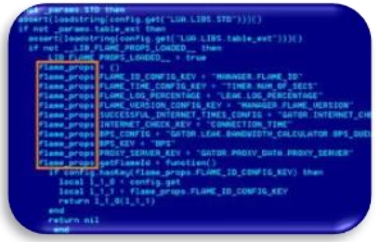
Design Challenges in Control of CPS

- **Cyber-Physical Systems: Safety-Critical Infrastructures**
  - **Safety: physical safety, functional safety, information security…**
  - **Critical: provide 100% guarantee!**

- **Cyber Controllers are Computational**
  - **Logical and high level behaviors**

- **CPS are very Complicated**
  - **Fully automated design: Correct-by-Construction**

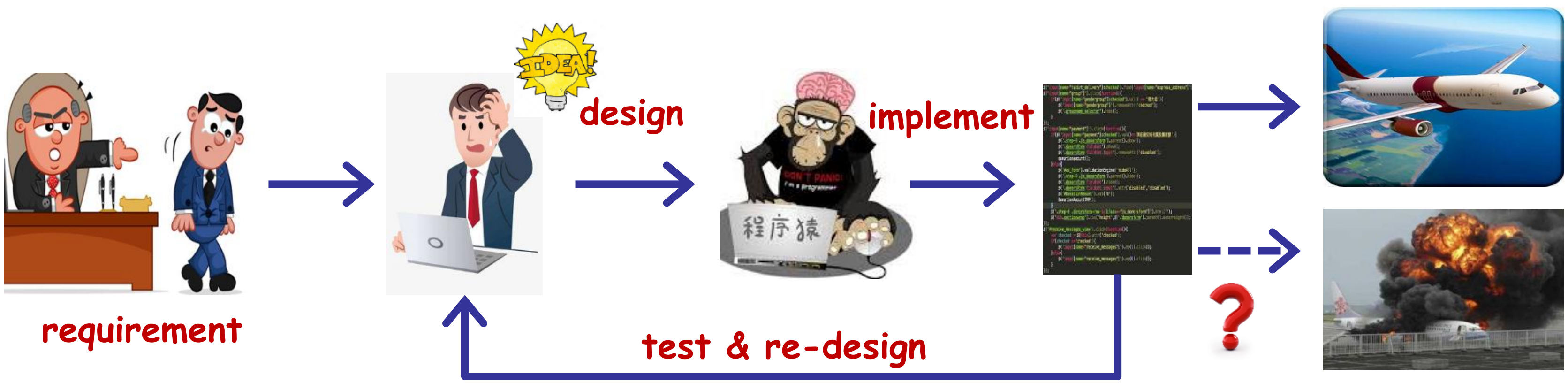


Current Design Process

Current Control Design Process for CPS

- given some spec by natural languages
- use engineering intuition/experience to come up with a solution
- extensive testing and iteration



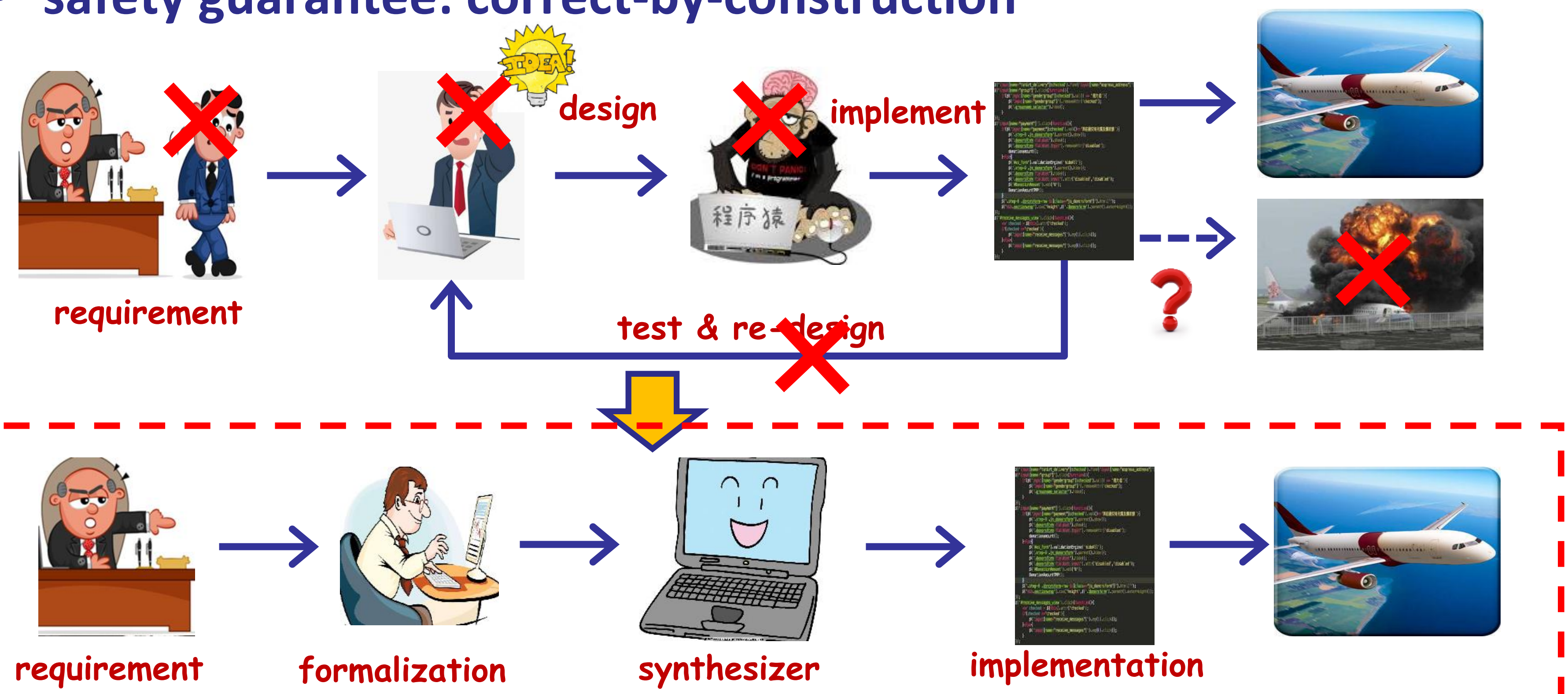
Problems

- not rigorous: Ad hoc approaches + lists of “if-then-else” rules
- little or no formal guarantees on correctness
- test & re-design: design period is very long

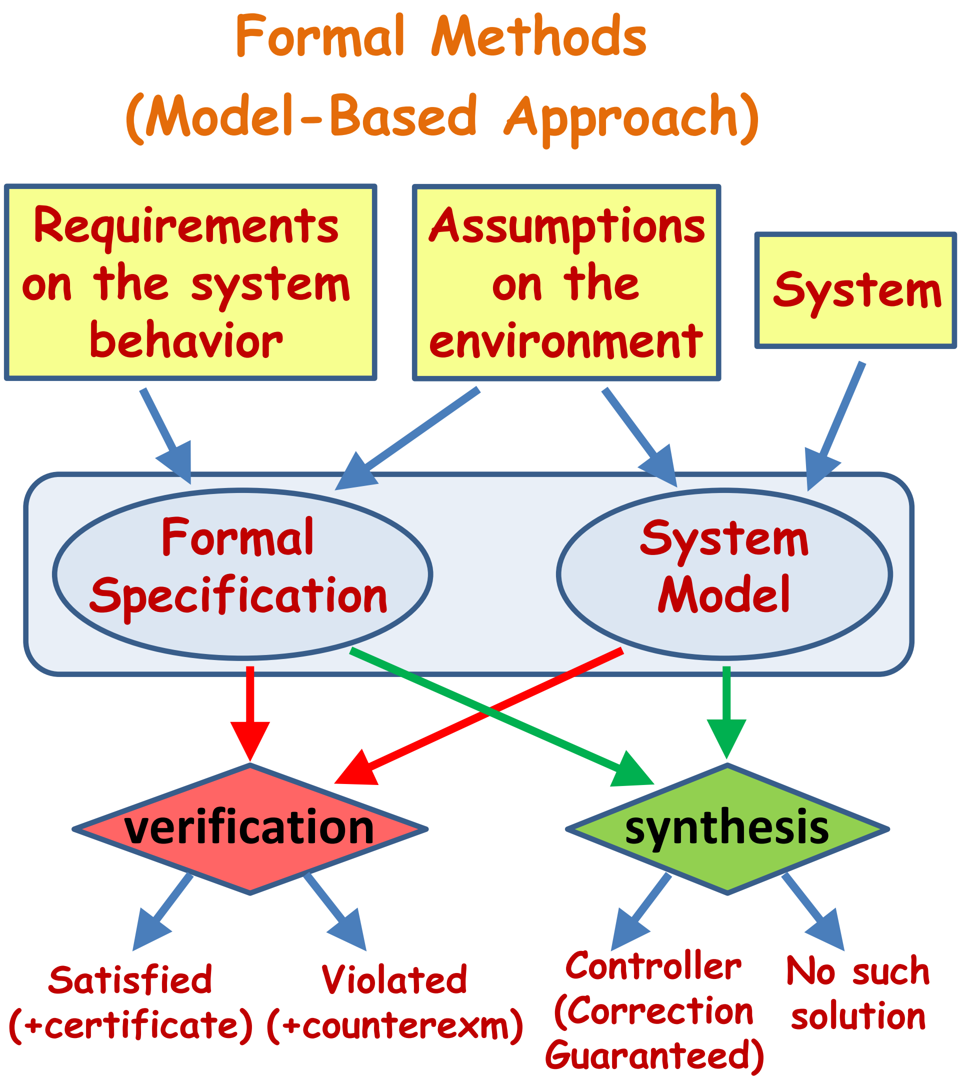
Ideal Design Process

Future Control Design Process for CPS (Hopefully!)

- rigorous requirement: no ambiguity
- fully automated: algorithmic process
- sequential design: no iteration
- safety guarantee: correct-by-construction



Formal Methods: Two Basic Paradigms



Formal Modeling

- Model: Transition Systems
- Specification: Formal Languages

Verification (Analysis)

- Formal guarantee for spec

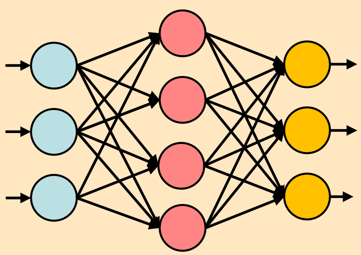
Synthesis (Control Design)

- Reactive to environment, e.g., controllability & observability
- Correct-by-construction!
  (No need to verify)

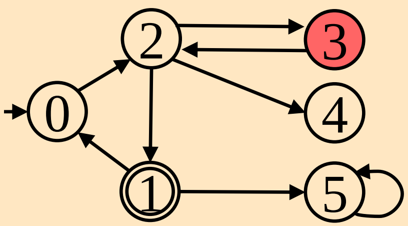
Relationship with AI

Formal reasoning & logic is the Foundation of AI

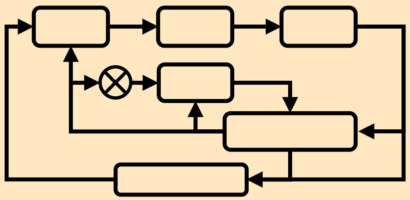
Connectionism



Symbolicism



Actionism



- AI is not just learning and neural networks
- Logics and formal reasoning are also important parts of AI
- Formal methods is closely related to dynamic systems

You Will Learn in This Course

- How to describe dynamic systems using formal models
  - labeled transition systems
  - bisimulation and quotient-based abstraction
- How to describe formal specifications/requirements
  - linear-time properties
  - linear-temporal logics, computation tree logics (briefly)
- How to formally verify whether a model satisfies a specification
  - automata-based LTL model checking
  - finite-state automata, Büchi automata, Rabin automata
- How to synthesize a reactive controller to enforce a specification
  - game-based LTL controller synthesis
  - safety game, reachability game, Büchi game, Rabin game

Basic Notations

Set Theory

- “belongs to” $\in$; “subset” $\subseteq, \subset$; “union” $\cup$; “intersection” $\cap$
- cartesian product: $A \times B = \{(a, b): a \in A, b \in B\}$, e.g., $\{1\} \times \{a, b\} = \{(1, a), (1, b)\}$
- powerset: $2^A = \{x: x \subseteq A\}$, e.g., $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
- cardinality: $|A|$ =number of elements in A, e.g., $|\{a, b, c\}| = 3$

Propositional & Predicate Logics

- “negation” $\neg$; “and” $\wedge$; “or” $\vee$; “implies” $\rightarrow$ (note: $a \rightarrow b \equiv \neg a \vee b$)
- quantifiers: “for all” $\forall$; “there exists” $\exists$
- $(\exists x)(\forall y)[x \text{ loves } y] \rightarrow (\forall y)(\exists x)[x \text{ loves } y]$; this formula is always true
- $(\forall x)(\exists y)[x \text{ loves } y] \rightarrow (\exists y)(\forall x)[x \text{ loves } y]$; this formula is not always true

Labeled Transition Systems

Labeled Transition Systems

A labeled transition system (LTS) is a tuple

$$T = (X, U, \rightarrow, X_0, AP, L)$$

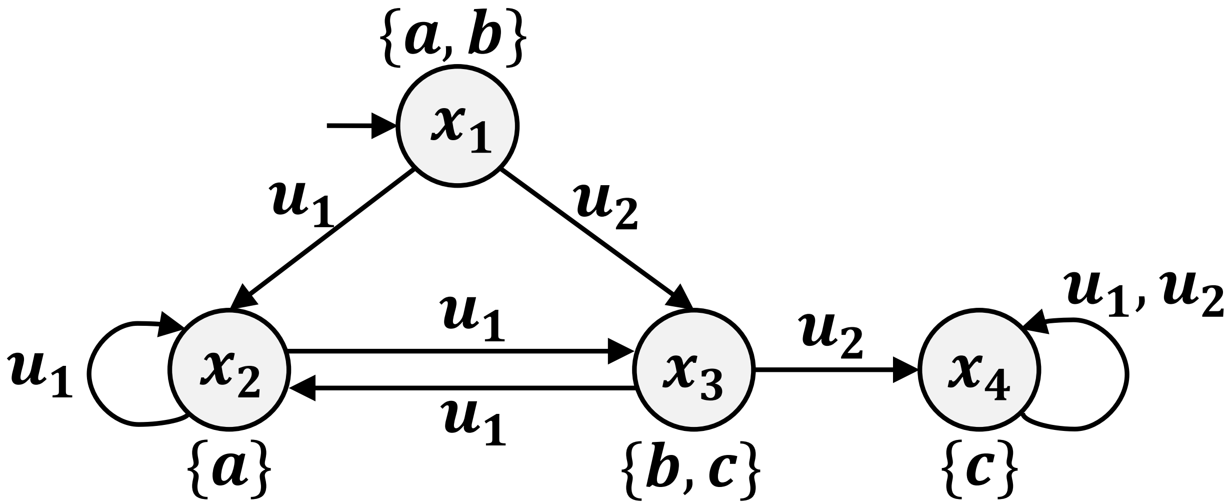
- X is a set of states
- U is a set of inputs (controls or actions)
- $\rightarrow \subseteq X \times U \times X$ is a transition relation
- $X_0 \subseteq X$ is a set of initial states
- AP is a set of atomic propositions
- $L: X \rightarrow 2^{AP}$ is a labeling function

➢ For any $(x, u, x') \in \rightarrow$, we also write it as $x \xrightarrow{u} x'$

➢ $\rightarrow$ can also be considered as a partial function from $X \times U$ to 2^X

Graph Representation of LTS

- $X = \{x_1, x_2, x_3, x_4\}$
- $U = \{u_1, u_2\}$
- $\rightarrow = \left\{ \begin{matrix} (x_1, u_1, x_2), (x_1, u_2, x_3), (x_2, u_1, x_2), (x_2, u_1, x_3), \\ (x_3, u_1, x_2), (x_3, u_2 x_4), (x_4, u_1, x_4), (x_4, u_2, x_4) \end{matrix} \right\} \subseteq X \times U \times X$
- $X_0 = \{x_1\} \subseteq X$
- $AP = \{a, b, c\}$
- $L(x_1) = \{a, b\},\ L(x_2) = \{a\},\ L(x_3) = \{b, c\},\ L(x_4) = \{c\}$

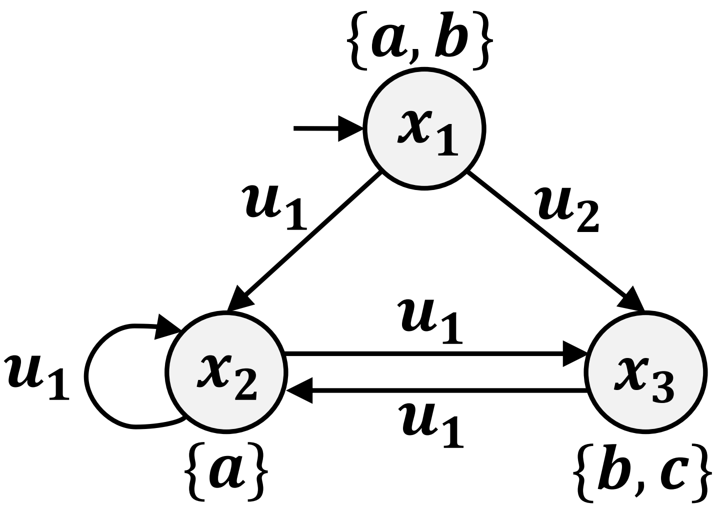


Successors & Predecessors

- **Successor states:** $Post(x,u) = \{x' \in X : x \xrightarrow{u} x'\}$
- **Predecessor states:** $Pre(x,u) = \{x' \in X : x' \xrightarrow{u} x\}$
- **The dynamic of the system is non-deterministic in general, i.e.,**

$$|X_0| > 1 \text{ or } |Post(x,u)| > 1$$

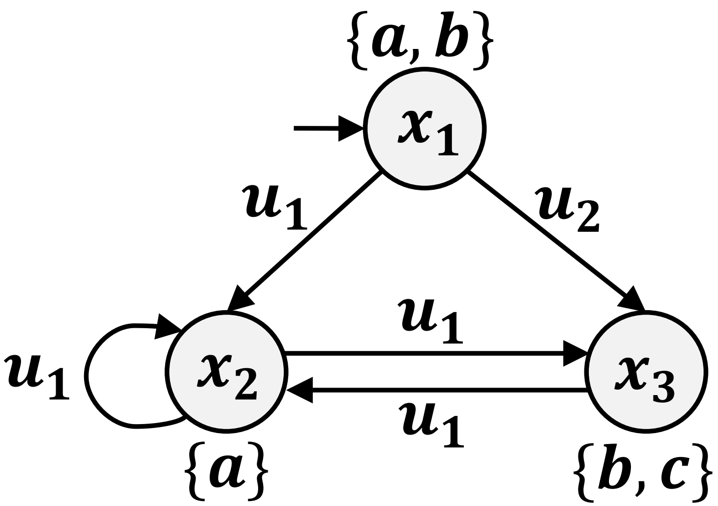
- **Non-determinism can model uncertainty or adversary**
- **LTS is said to be deterministic if** $|X_0| = 1 \wedge |Post(x,u)| = 1, \forall x \in X, u \in U$



- $Post(x_1,u_1) = \{x_2\},\ Post(x_2,u_1) = \{x_2,x_3\}$
- $Pre(x_2,u_1) = \{x_1,x_2,x_3\},\ Pre(x_2,u_3) = \{x_1\}$
- **Define** $Pre(x) = \cup_{u\in U} Pre(x,u)$; $Post(x)$ **the same**
- $Post(x_2) = \{x_2,x_3\}, Pre(x_2) = \{x_1,x_2,x_3\}$

Dynamic of LTS

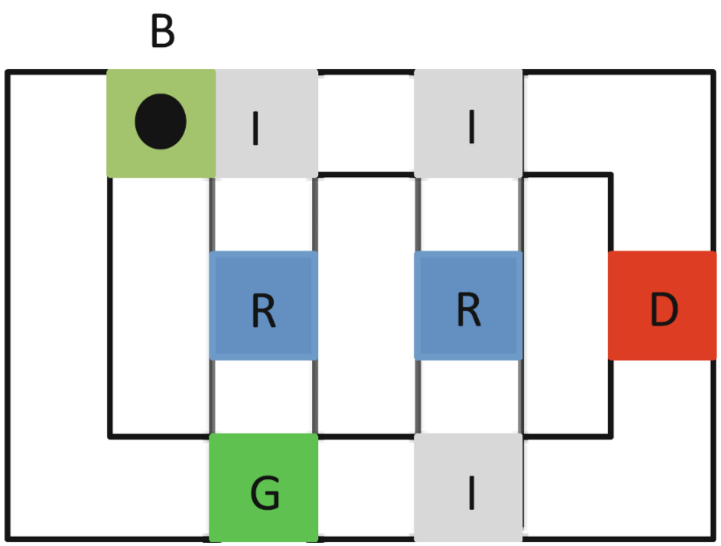
- Input **run**: $u_1 u_2 \cdots u_n (\cdots)$
- State **run**: $\rho = x_0 x_1 \cdots x_n (\cdots)$ such that $x_0 \xrightarrow{u_1} x_1 \xrightarrow{u_2} \cdots \xrightarrow{u_n} x_n$
- **Trace**: $L(\rho) = L(x_0) L(x_1) \cdots L(x_n) (\cdots)$
- All finite runs $Run^f(T)$; all infinite runs $Run(T)$
- All finite runs $Trace^f(T)$; all infinite runs $Trace(T)$
- Note: the state run or trace generated by an input run may not be unique



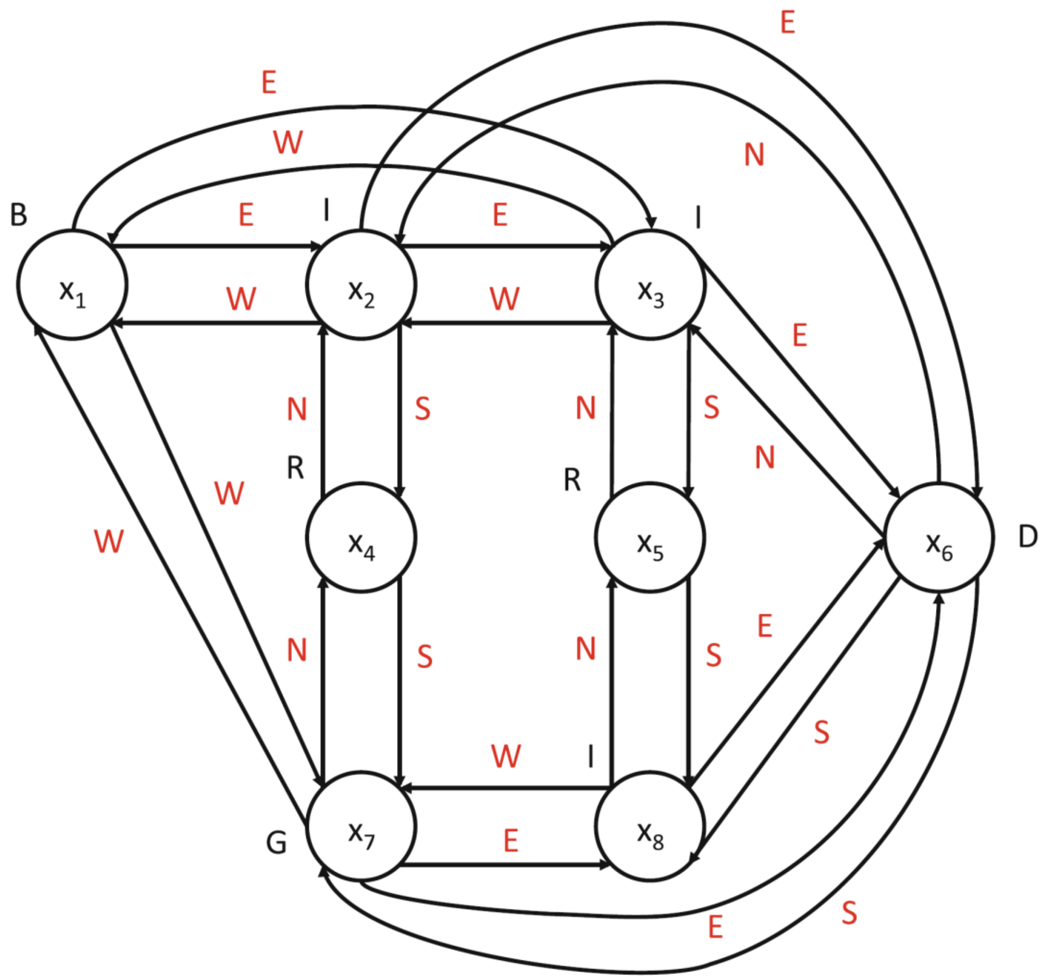
- Input: $u_2 u_1 u_1$
- Run: $x_1 \xrightarrow{u_2} x_3 \xrightarrow{u_1} x_2 \xrightarrow{u_1} x_3$ or $x_1 \xrightarrow{u_2} x_3 \xrightarrow{u_1} x_2 \xrightarrow{u_1} x_2$
- Trace: $\{a,b\}\{b,c\}\{a\}\{b,c\}$ or $\{a,b\}\{b,c\}\{a\}\{a\}$
- Infinite runs: $x_1(x_3 x_2)^\omega$ or $x_1 x_3 x_2 x_3 (x_2)^\omega$
- ω means "repeat infinite times"

LTS Example: Robot in a Grid-World

- **B:** base
- **I:** intersection
- **R:** recharge region
- **D:** dangerous region
- **G:** data gather region



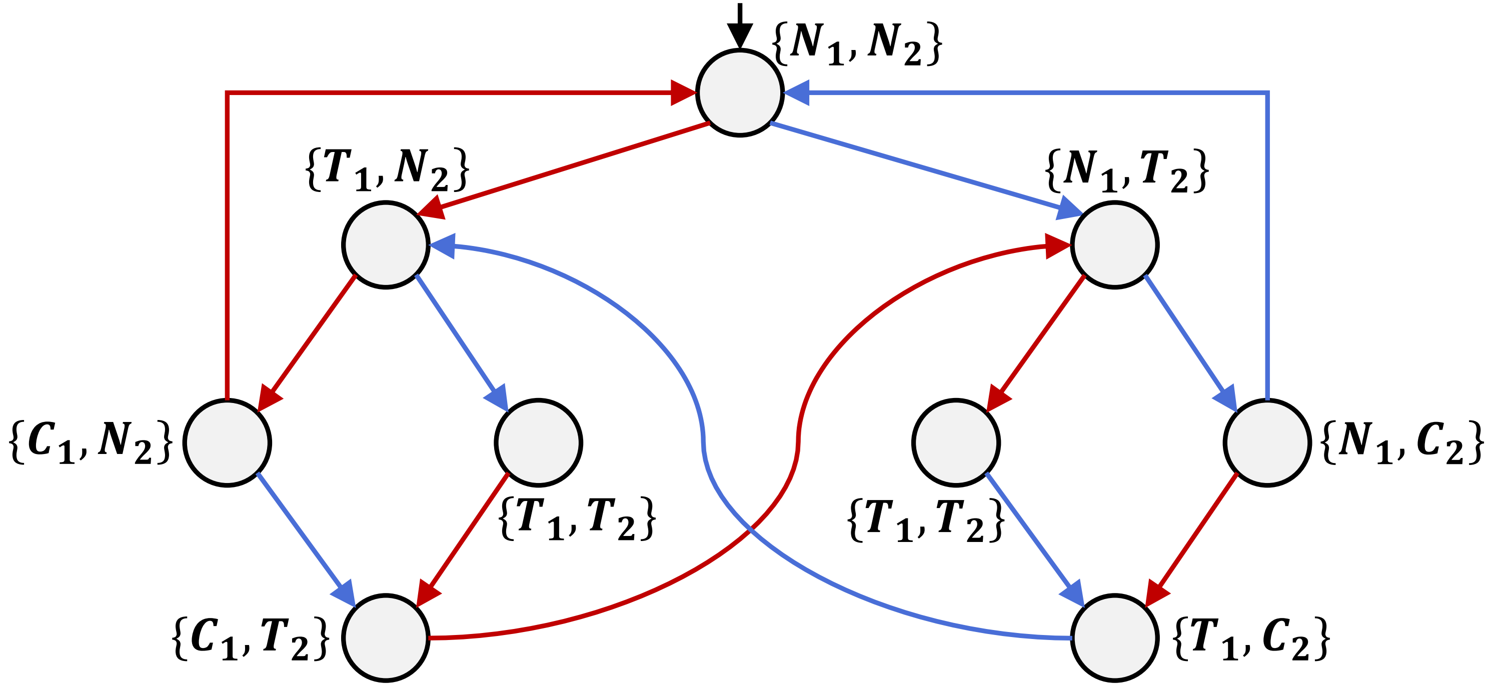
A robot in an indoor space



Corresponding LTS

LTS Example: Mutually Exclusive Processes

- two processes sharing a resource (asynchronous)
- each process has a critical section in its code
- non-critical state (N) → trying to enter critical state (T) → critical state (C)
- only one process can be in its critical section at a time (mutual exclusion)



A possible mutually exclusive protocol represented by LTS

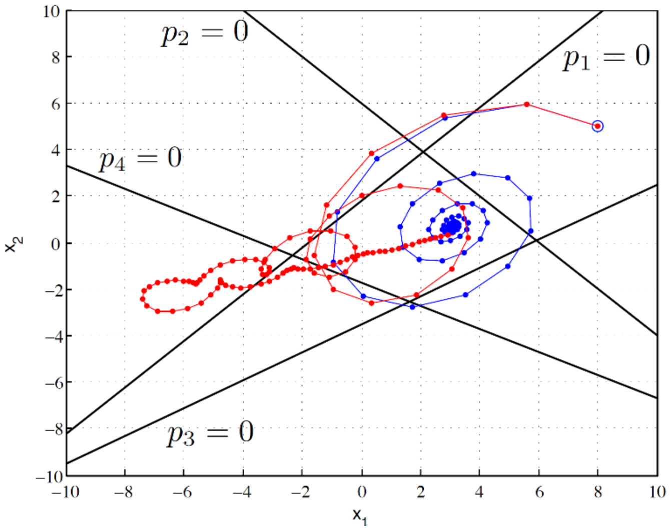
Process 1 Process 2

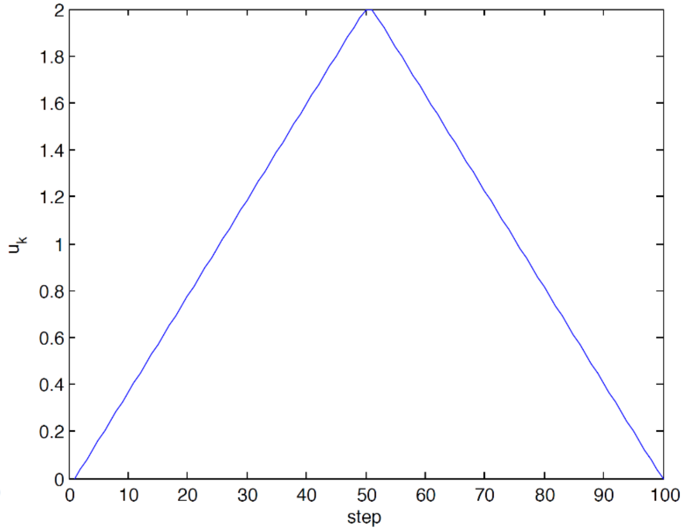
LTS Example: Discrete Time Control System

- $\mathcal{D}$: $x_{k+1} = Ax_k + Bu_k + b$, where $A = \begin{bmatrix} 0.95 & -0.5 \\ 0.5 & 0.65 \end{bmatrix}, B = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 0.5 \\ -1.3 \end{bmatrix}$

$\Updownarrow$

- One-step embedding of $\mathcal{D}$: $(X = \mathbb{R}^2, U = \mathbb{R}, \delta(x,u) = Ax_k + Bu_k + b)$





Use affine function to describe property

- p: $p_i(x) > 0$
- z: $p_i(x) = 0$
- n: $p_i(x) < 0$

- Input word: $u_0 u_1 \cdots u_{100}$ with $u_k = 0.04k$ ($k \leq 50$), $u_k = -0.04k + 4$ ($k \geq 50$)
- State run: $\begin{bmatrix} 8 \\ 5 \end{bmatrix} \begin{bmatrix} 5.60 \\ 5.95 \end{bmatrix} \begin{bmatrix} 2.80 \\ 5.44 \end{bmatrix} \cdots$
- Trace: $(p,p,n,p)(p,p,n,p)(n,p,n,p)\cdots$, e.g., $p_1(x) = [1 \quad -1]x + 1.8$

Composition of LTSs

- Building the LTS model directly for the entire system is very difficult
- In practice, a large system is **composed** by many components
- Components interact with each other by
  - some "private actions" can execute individually/asynchronously
  - some "common actions" need to execute synchronously
- **Monolithic Model = Local Modules + Synchronization Rules**

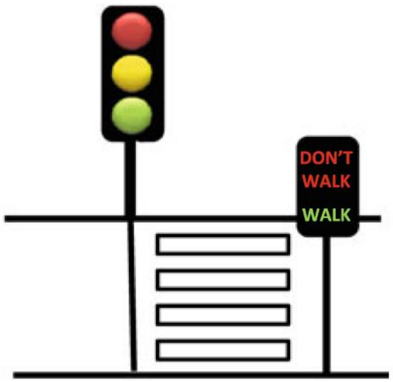
Product of LTSs

Let T_1 and T_2 be two LTSs, where $T_i = (X_i, U_i, \rightarrow_i, X_{0,i}, AP_i, L_i)$. The **product of** T_1 and T_2 is a new LTS

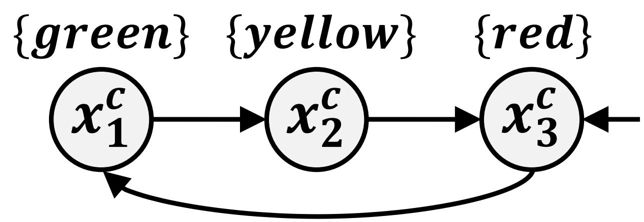
$$T_1 \otimes T_2 = (X, U, \rightarrow, X_0, AP, L)$$

- $X = X_1 \times X_2$ with $X_0 = X_{0,1} \times X_{0,2}$
- $U = U_1 \cup U_2$, $AP = AP_1 \cup AP_2$, $L(x_1, x_2) = L(x_1) \cup L(x_2)$
- $\rightarrow \subseteq X \times U \times X$ is defined by:
  - $u \in U_1 \cap U_2$: $(x_1, x_2) \xrightarrow{u} (x_1', x_2')$ iff $x_1 \underset{1}{\xrightarrow{u}} x_1'$ and $x_2 \underset{2}{\xrightarrow{u}} x_2'$
  - $u \in U_1 \setminus U_2$: $(x_1, x_2) \xrightarrow{u} (x_1', x_2)$ iff $x_1 \underset{1}{\xrightarrow{u}} x_1'$
  - $u \in U_2 \setminus U_1$: $(x_1, x_2) \xrightarrow{u} (x_1, x_2')$ iff $x_2 \underset{2}{\xrightarrow{u}} x_2'$

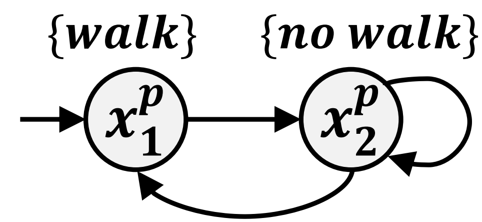
Example: Traffic Lights



A pedestrian crossing system



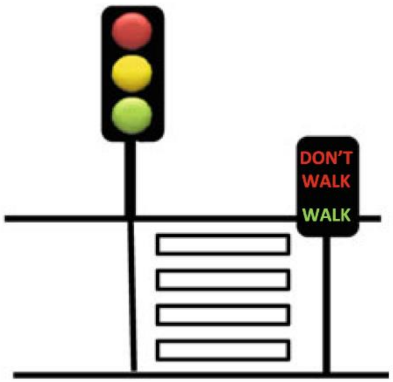
Car light system T_c



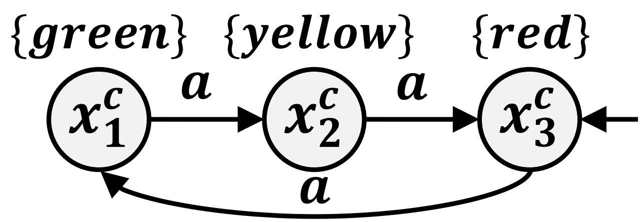
Pedestrian light system T_p

Example: Traffic Lights

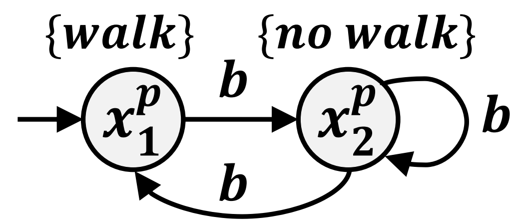
Case 1: Two lights are fully asynchronized



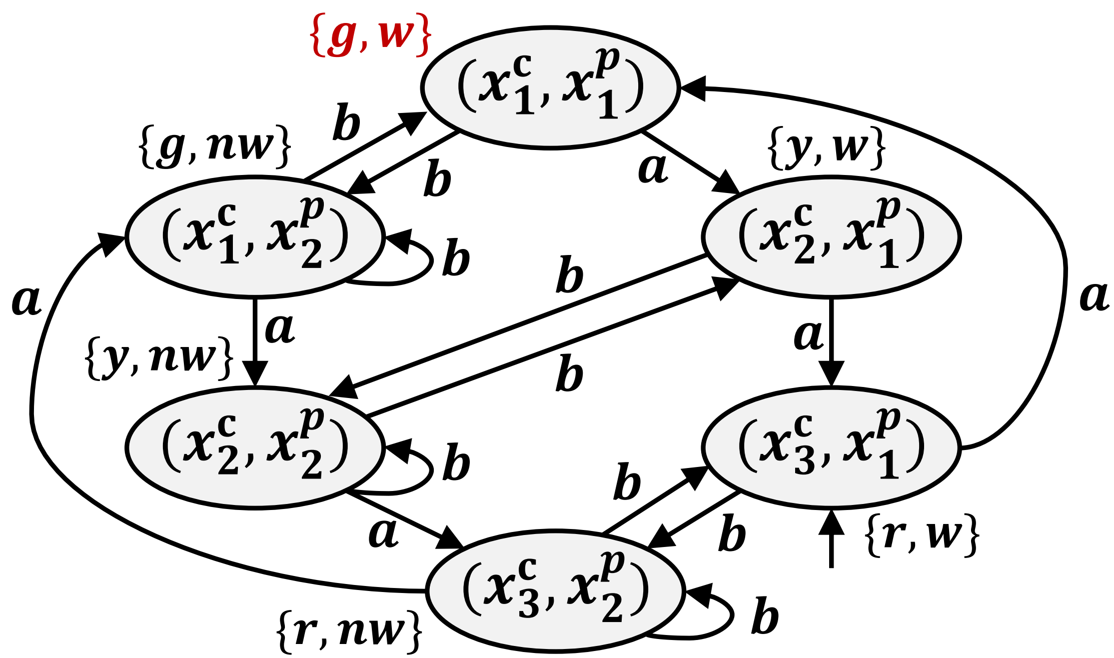
A pedestrian crossing system



Car light system T_c



Pedestrian light system T_p

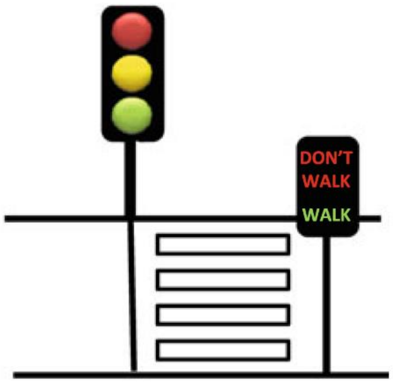


Product system $T_c \otimes T_p$

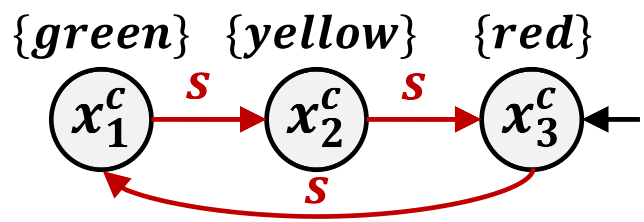
Not safe due to $\{g, w\}$!

Example: Traffic Lights

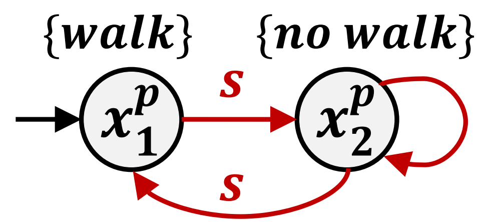
Case 2: Two lights are fully synchronized



A pedestrian crossing system



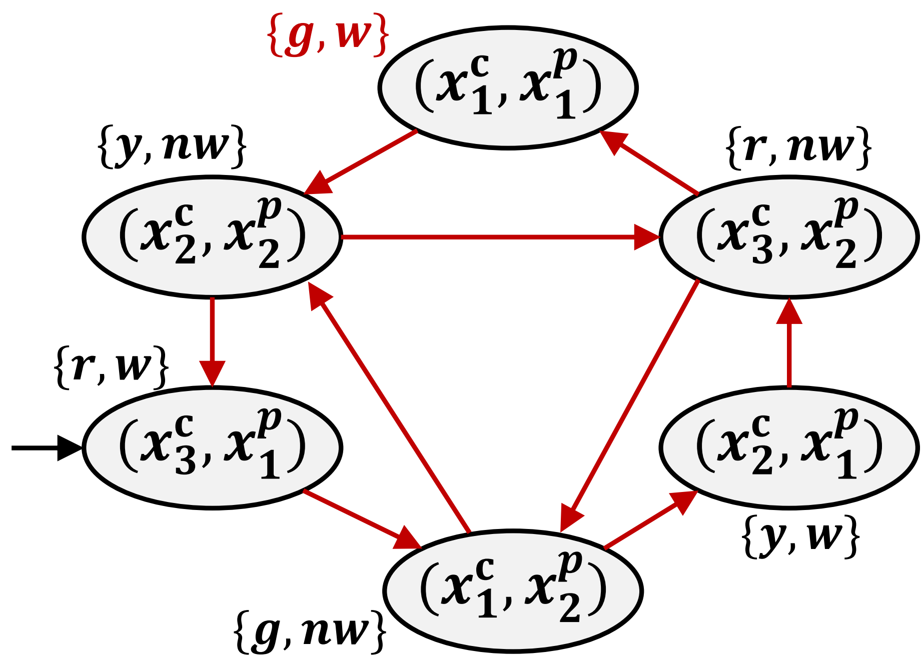
Car light system T_c



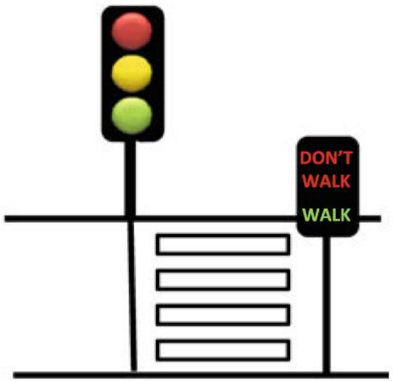
Pedestrian light system T_c

Product system $T_c \otimes T_p$

Not safe due to $\{g, w\}$!

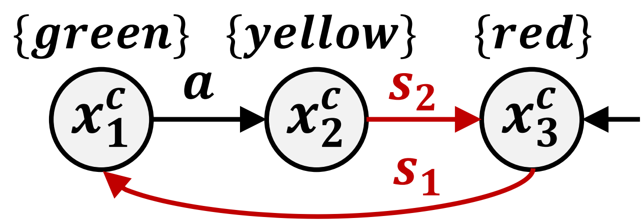


Example: Traffic Lights

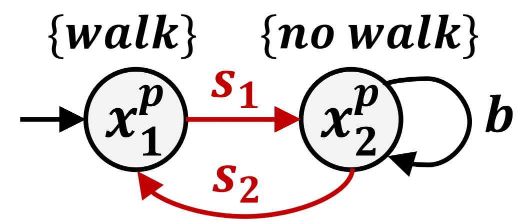


A pedestrian crossing system

Case 3: Two lights are partially synchronized



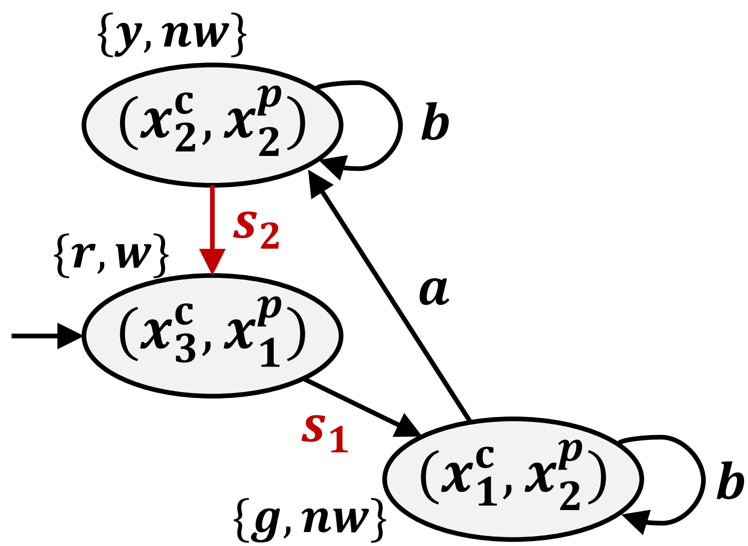
Car light system T_c



Pedestrian light system T_c

Product system $T_c \otimes T_p$

Safe!



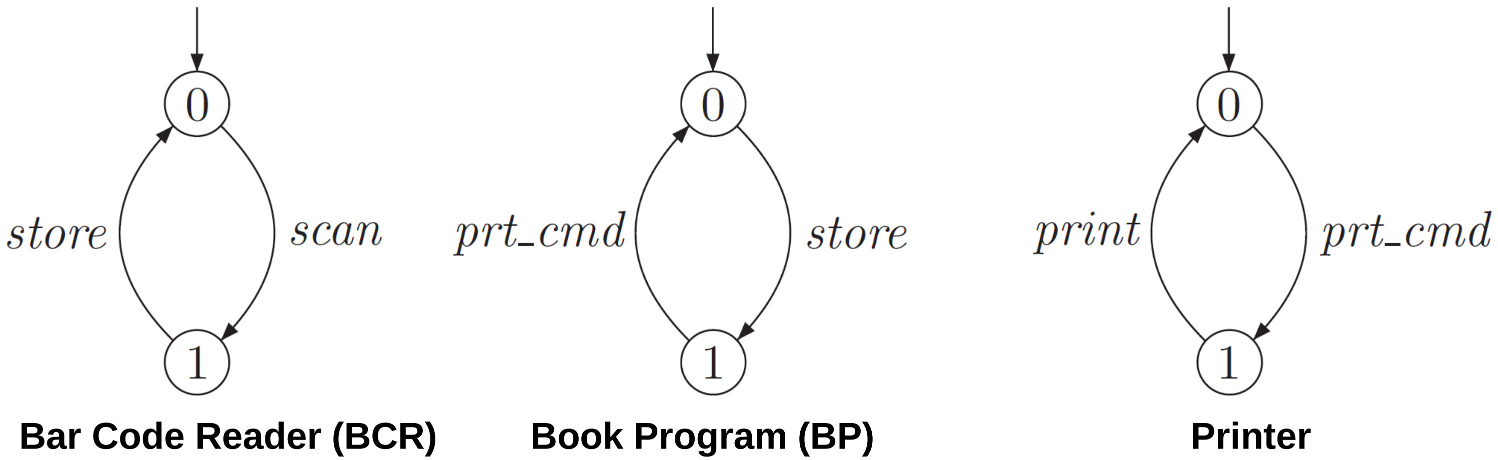
Comments on Product Composition

- Fully asynchronized system is essentially a "shuffle"
- Synchronization essentially restricts the behavior of each module
- Not all possible states are reachable in the product
- Synchronization can be physical, or by communication, or by control
- Controller can be a new component that synchronizes with the plant
- The state-space grows exponentially fast in the # of components
- "product" and "parallel" compositions are sometimes different in the literature; we do not distinguish explicitly here

Stage Summary

- Dynamic system can be represented as an LTS
- “States” in LTS can be either physical locations or logical status
- Atomic propositions represent high-level properties of interest
- Large system is usually composed by local modules by product
- Product essentially captures how systems interact with each other

Question



- The **bar code reader** reads a bar code and communicates the data of the just scanned product to the **booking program**. On receiving such data, the booking program transmits the price of the article to the **printer** that prints the article Id together with the price on the receipt.

- **Question: Build the entire booking system $BCR \otimes BP \otimes Printer = (BCR \otimes BP) \otimes Printer$**