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

2020.07 厦门大学暑期课程

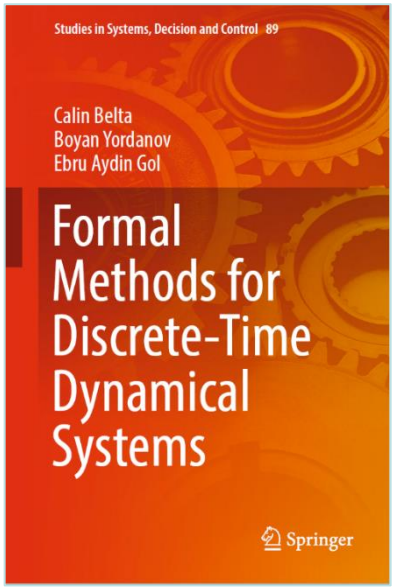
Course Information

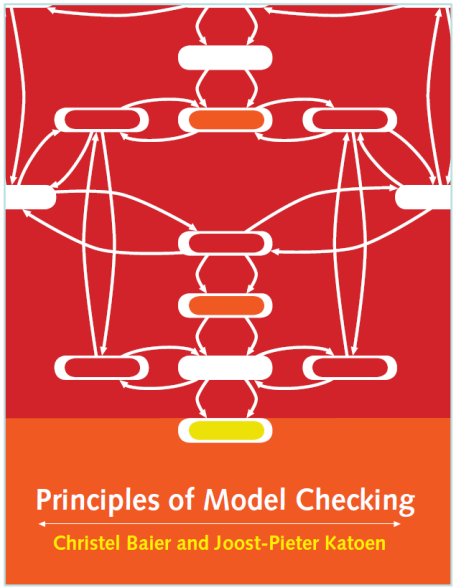
- 课程：《动态系统的形式化分析与控制》
- 教师：殷翔，副教授，博导，国家青年千人
- 课时：15 Hours (3 Classes × 5 nights)
- 7.7 (周二), 7.9 (周四), 7.11 (周六), 7.14 (周二), 7.16 (周四)
- 助教邮箱：kedixie@163.com
- 考核方式：考勤40%+作业40%+测验20%

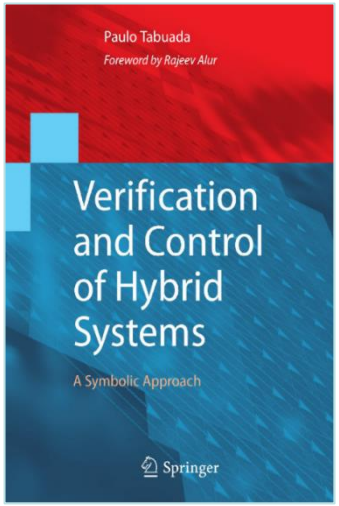
动态系统的形式分析与...

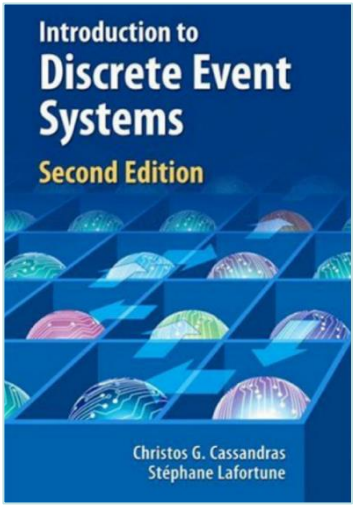
扫一扫二维码，加入群聊。

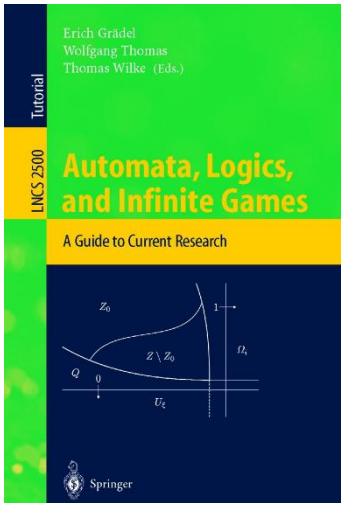
References











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

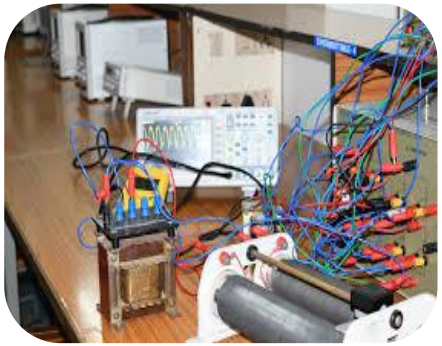
$$\mathcal{D}: \begin{cases} \dot{x}(t) = f(x(t), u(t), w(t)) \\ y(t) = g(x(t), u(t), v(t)) \end{cases}$$

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



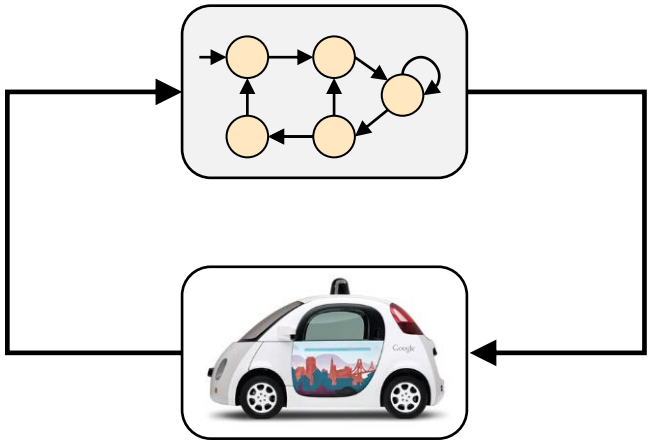
Circuits

Program

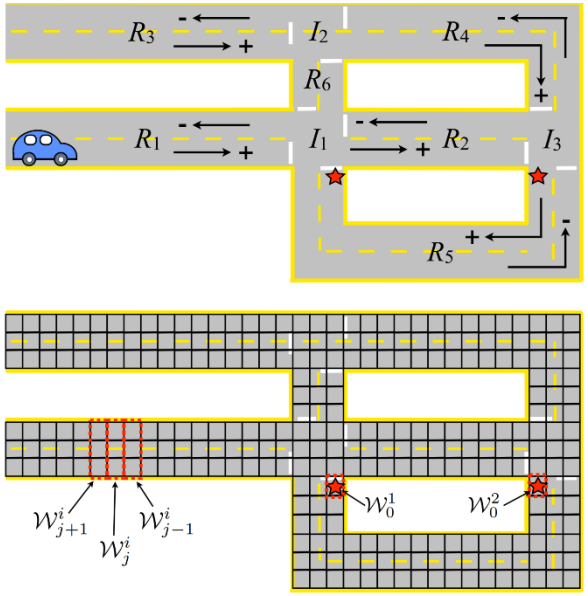
Dynamic Systems

- 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

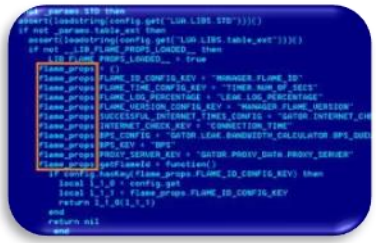
Physical-Layer: Physic Objects, ODE Dynamics

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**

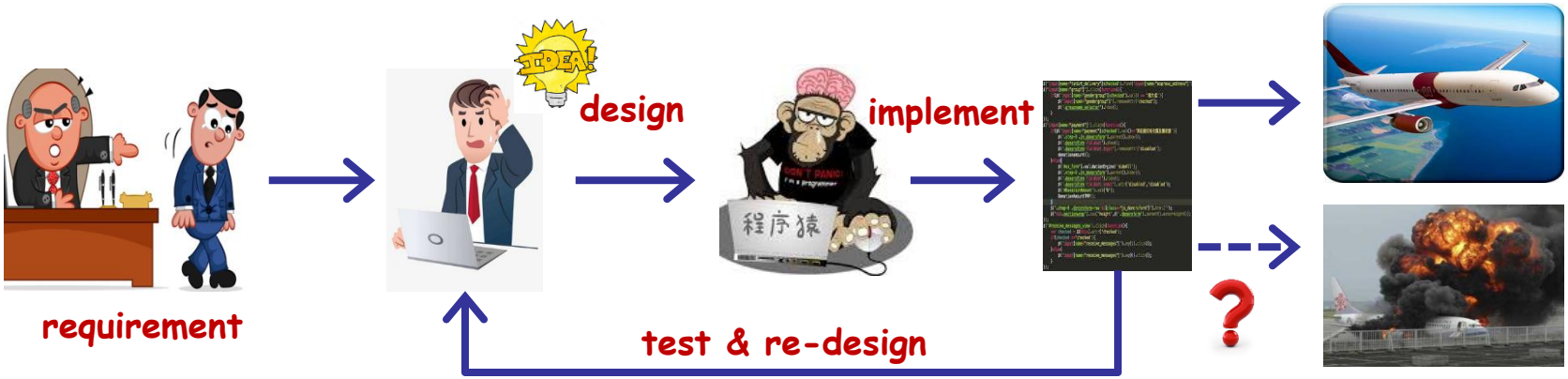


Current Design Process

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



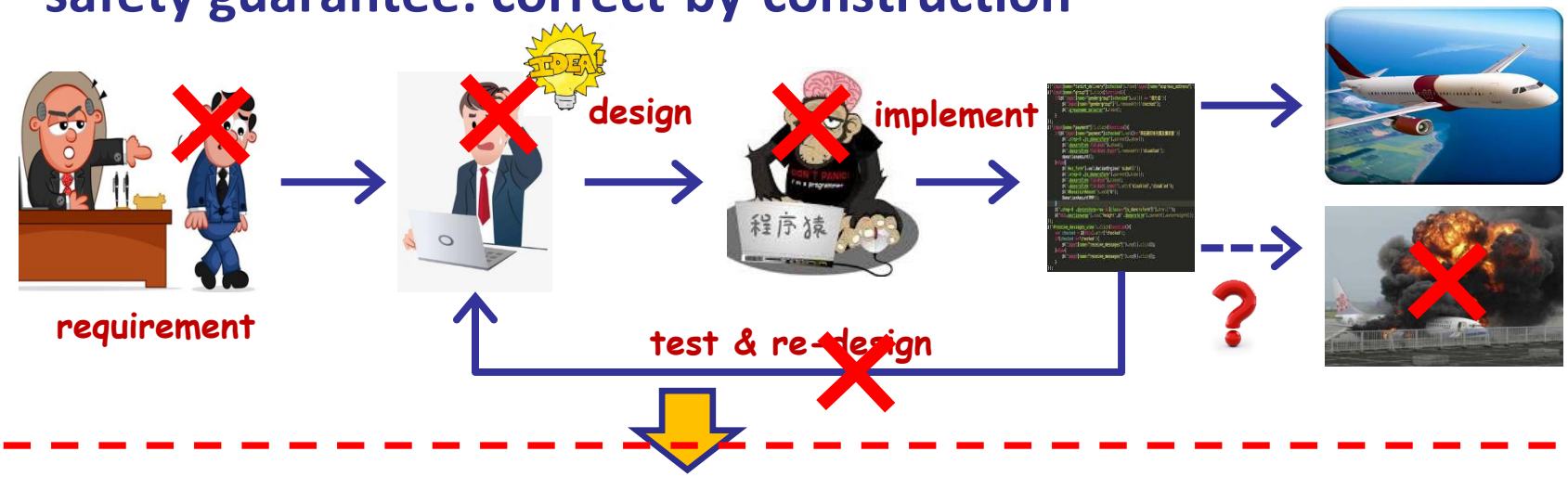
Problems

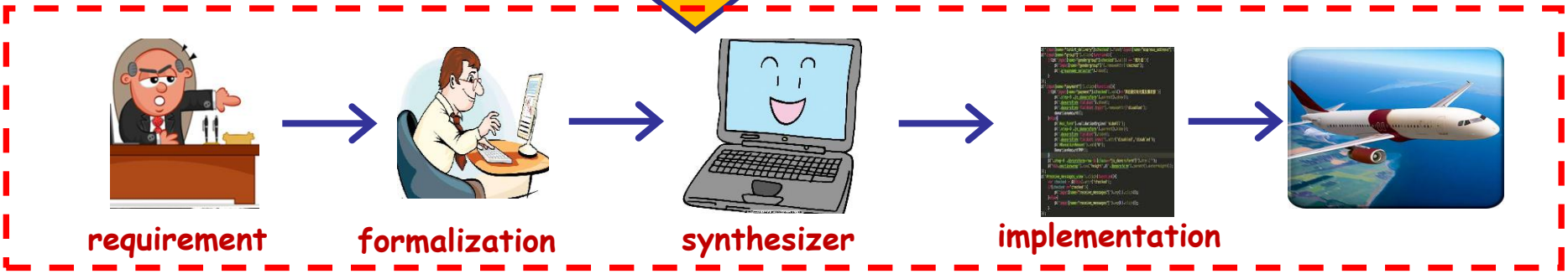
- not rigorous: Ad hoc approaches + lists of “if-then-else” rules
- little or no formal guarantees on correctness
- test & re-design: design period is very long

Ideal Design Process

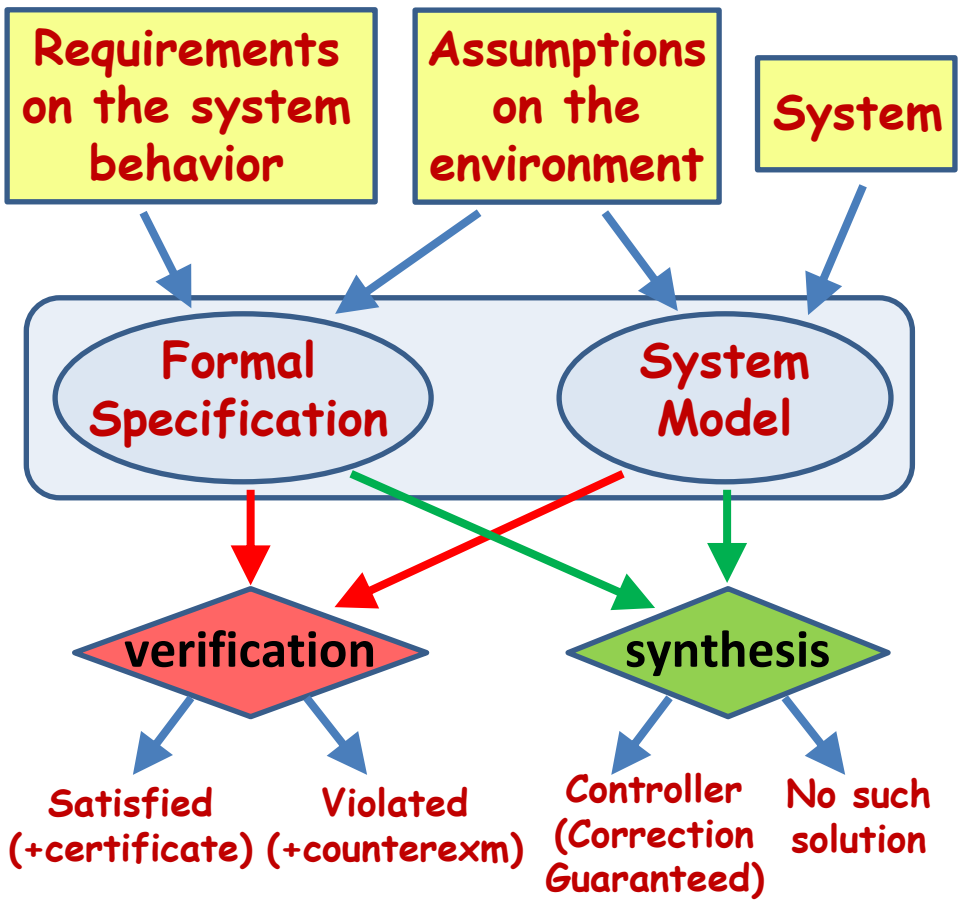
Future Control Design Process for CPS (Hopefully!)

- 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

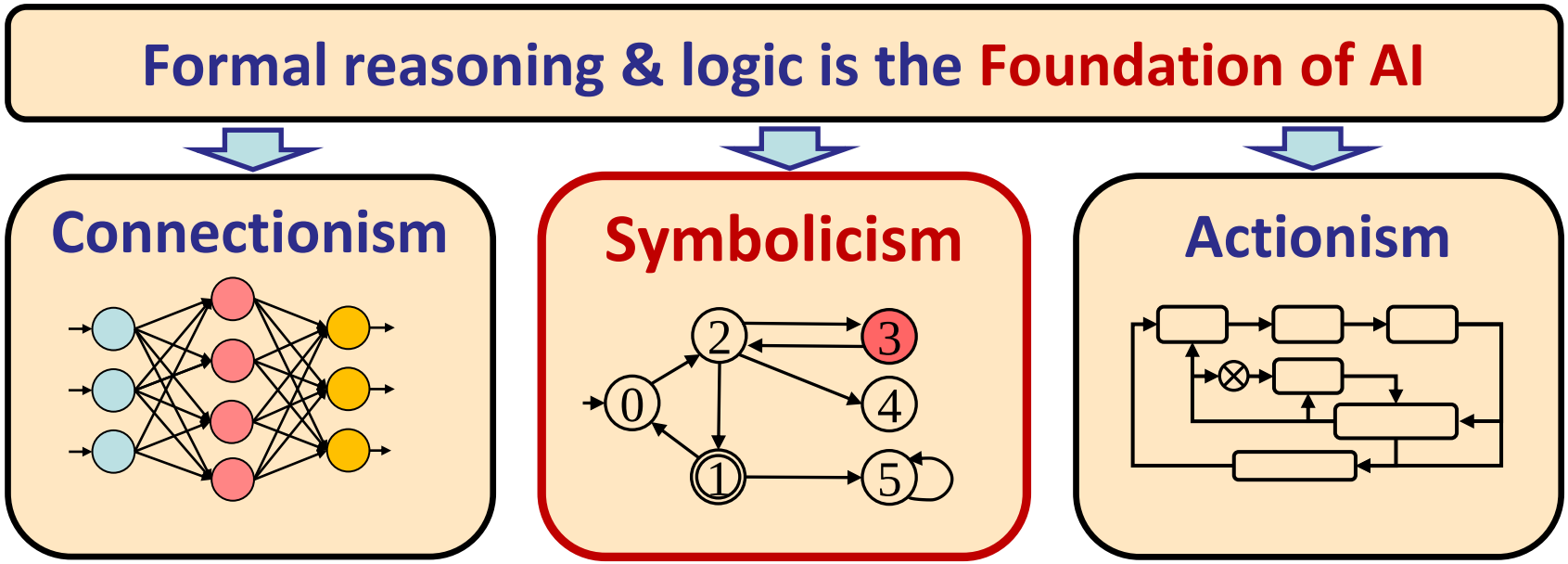
Verification (Analysis)

- Formal guarantee for spec

Synthesis (Control Design)

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

Relationship with AI



- 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: $|A|$ =# 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

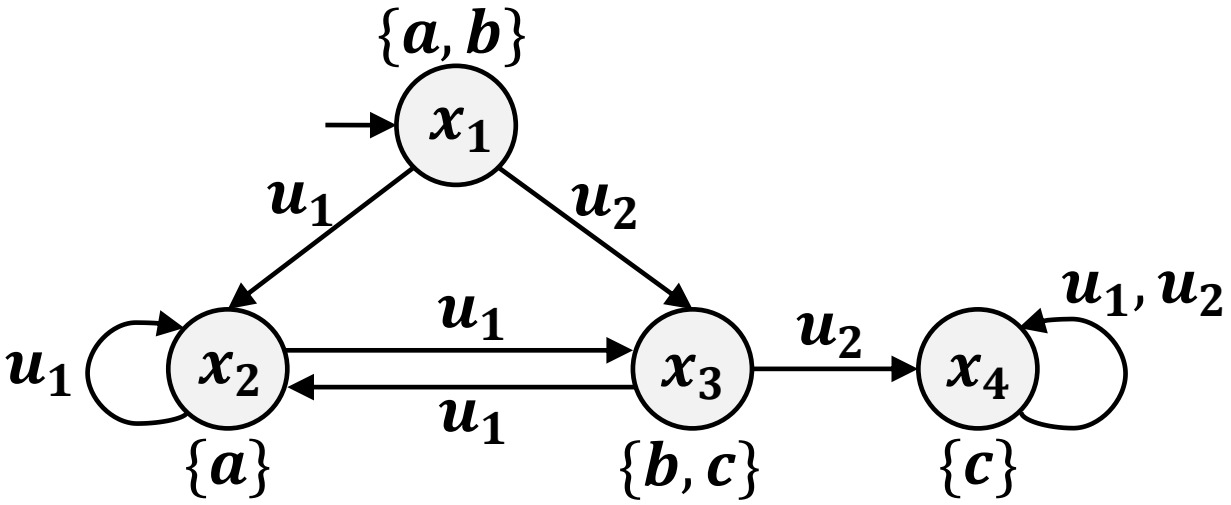
$$T = (X, U, \rightarrow, X_0, AP, L)$$

- 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

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

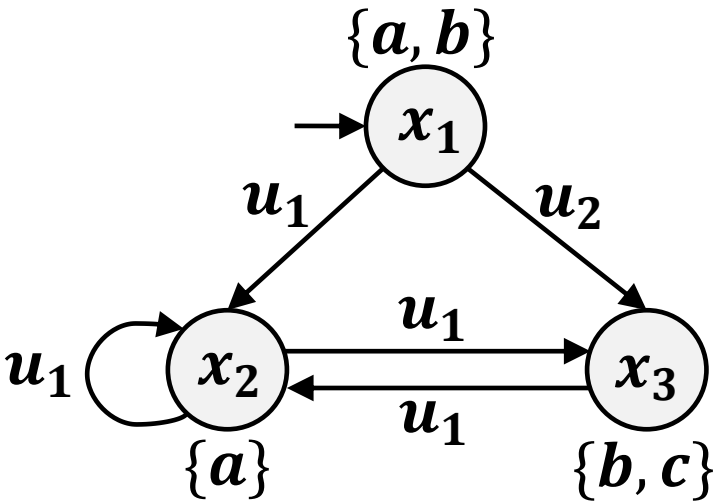


Successors & Predecessors

- **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.,

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

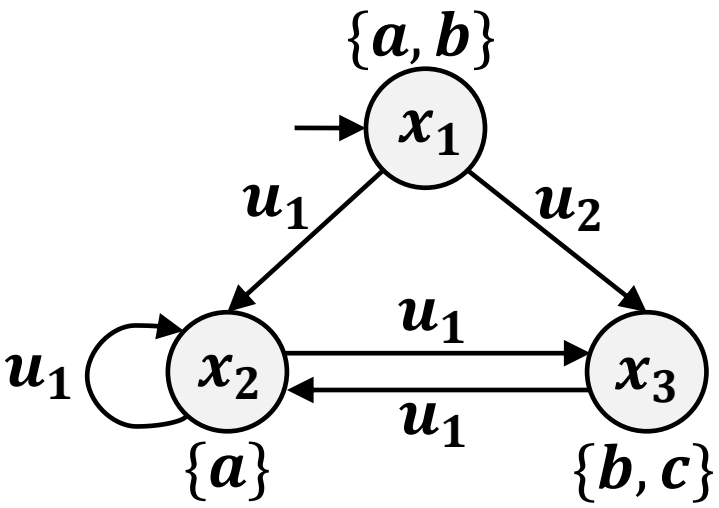
- Non-determinism can model **uncertainty** or **adversary**
- LTS is said to be **deterministic** if $|X_0| = 1 \wedge |Post(x,u)| = 1, \forall x \in X, u \in U$



- $Post(x_1,u_1) = \{x_2\},\ Post(x_2,u_1) = \{x_2,x_3\}$
- $Pre(x_2,u_1) = \{x_1,x_2,x_3\},\ Pre(x_2,u_3) = \{x_1\}$
- 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

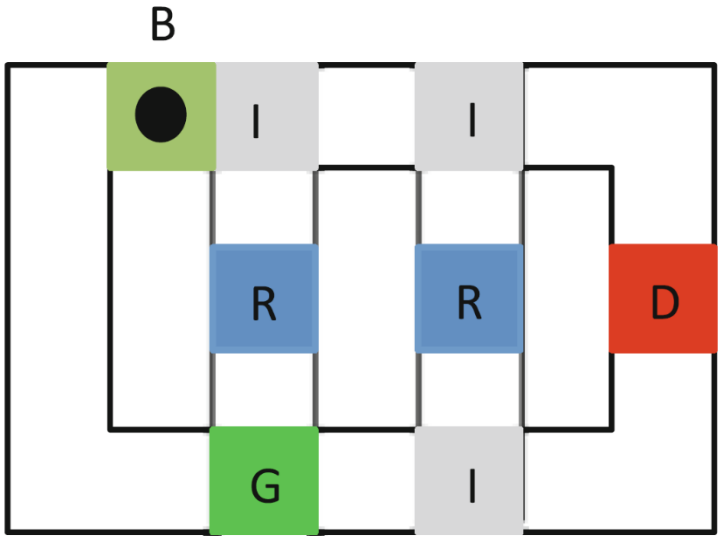
- Input run: $u_1 u_2 \cdots u_n(\cdots)$
- State run: $\rho = x_0 x_1 \cdots x_n(\cdots)$ such that $x_0 \xrightarrow{u_1} x_1 \xrightarrow{u_n} \cdots \xrightarrow{u_n} x_n$
- Trace: $L(\rho) = L(x_0)L(x_1)\cdots L(x_n)(\cdots)$
- All finite runs $Run^f(T)$; all infinite runs $Run(T)$
- All finite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



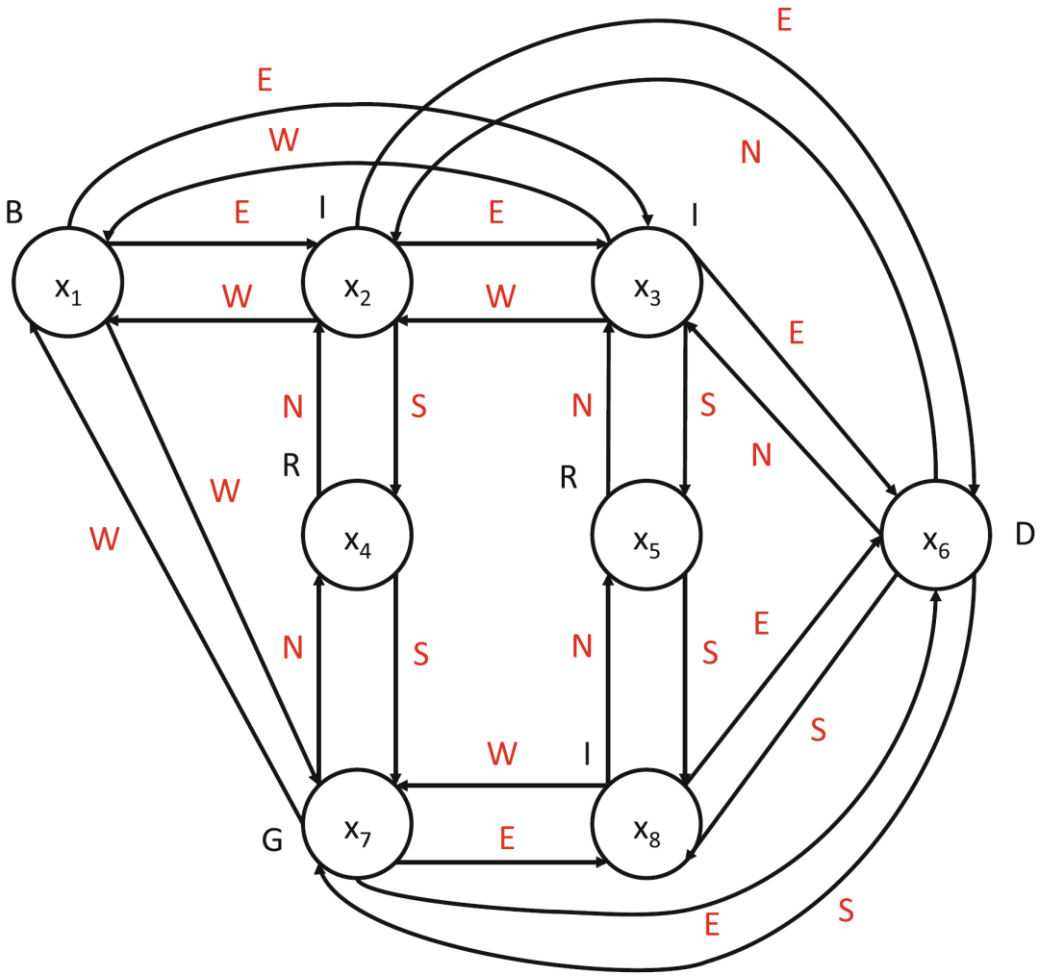
- Input: $u_2 u_1 u_1$
- Run: $x_1 \xrightarrow{u_2} x_3 \xrightarrow{u_1} x_2 \xrightarrow{u_1} x_3$ or $x_1 \xrightarrow{u_2} x_3 \xrightarrow{u_1} x_2 \xrightarrow{u_1} x_2$
- Trace: $\{a, b\}\{b, c\}\{a\}\{b, c\}$ or $\{a, b\}\{b, c\}\{a\}\{a\}$
- Infinite runs: $x_1(x_3 x_2)^\omega$ or $x_1 x_3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



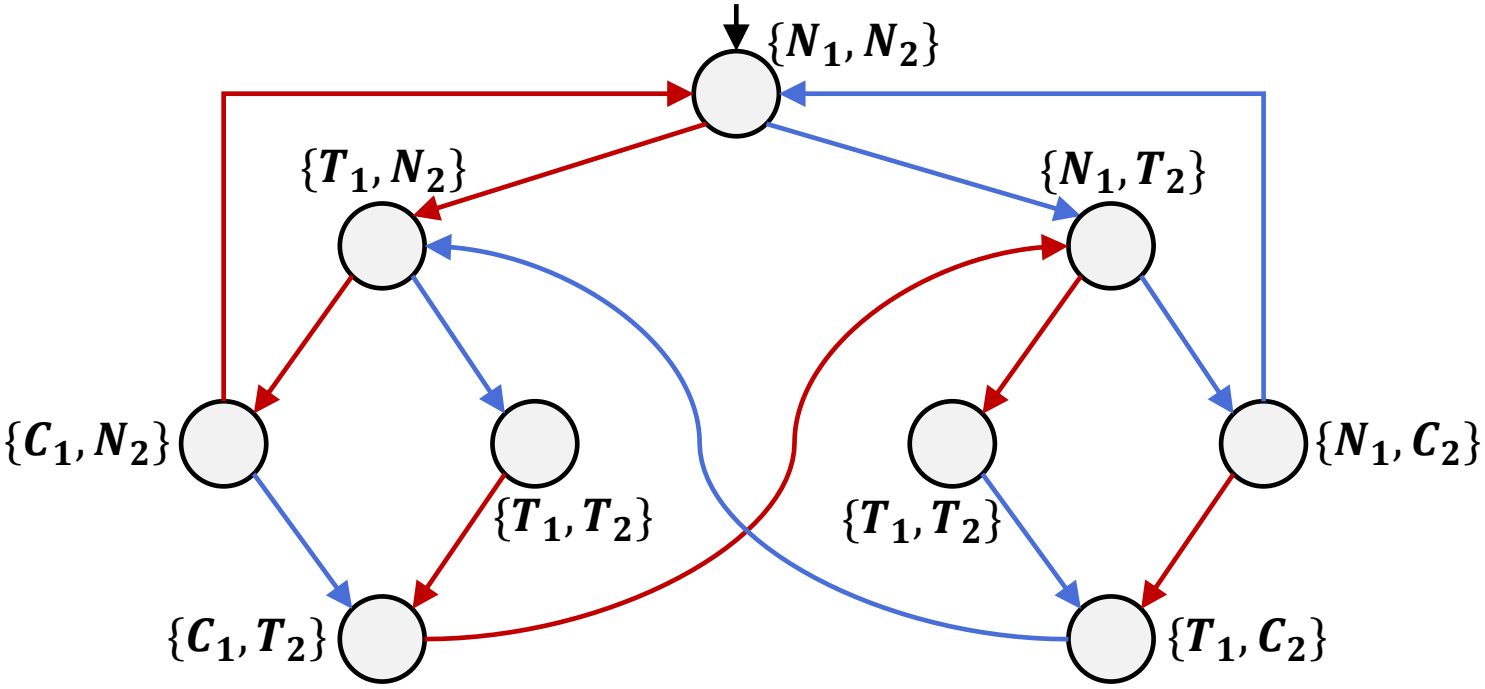
A robot in an indoor space



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)



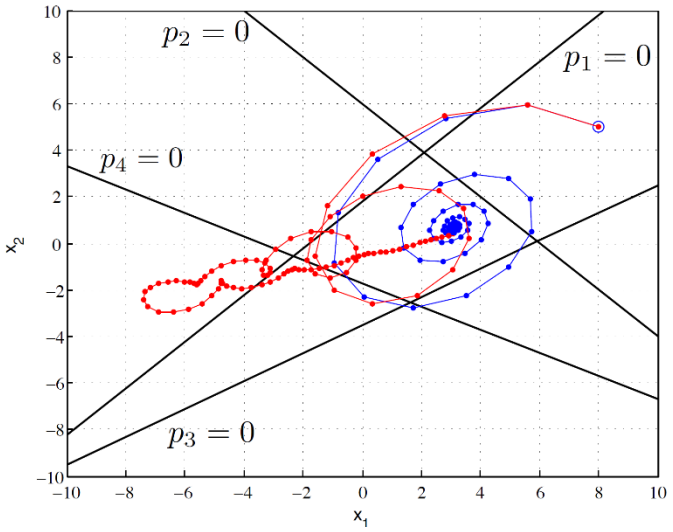
A possible mutually exclusive protocol represented by LTS

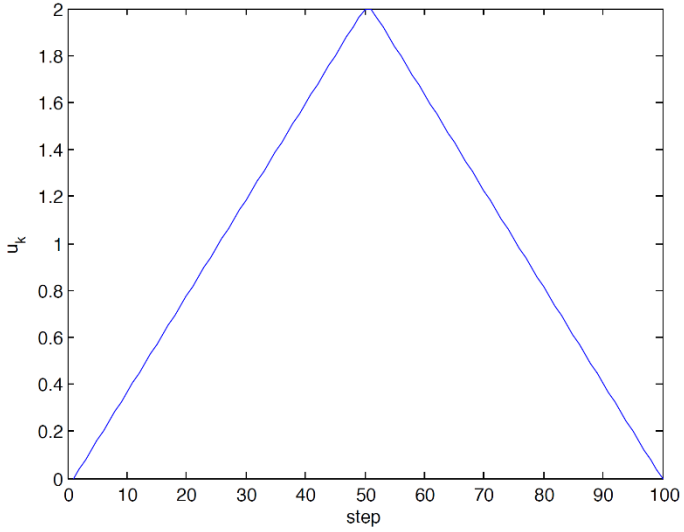
Process 1 **Process 2**

LTS Example: Discrete Time Control System

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

- **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

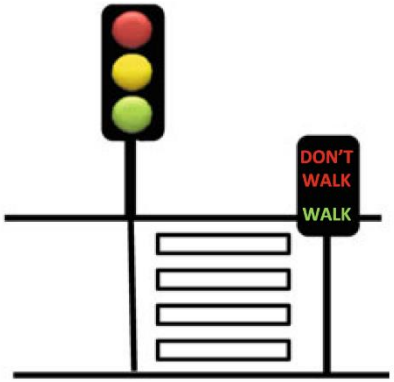
Product of LTSs

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

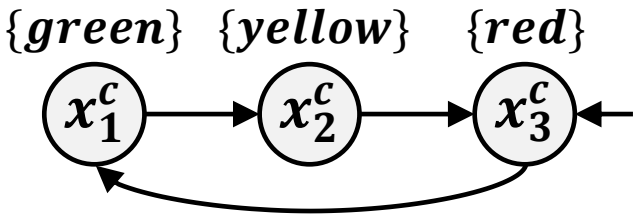
$$T_1 \otimes T_2 = (X, U, \rightarrow, X_0, AP, L)$$

- $X = X_1 \times X_2$ 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 $x_1 \xrightarrow[1]{u} x_1'$ and $x_2 \xrightarrow[2]{u} x_2'$
  - $u \in U_1 \setminus U_2$: $(x_1, x_2) \xrightarrow{u} (x_1', x_2)$ iff $x_1 \xrightarrow[1]{u} x_1'$
  - $u \in U_2 \setminus U_1$: $(x_1, x_2) \xrightarrow{u} (x_1, x_2')$ iff $x_2 \xrightarrow[2]{u} x_2'$

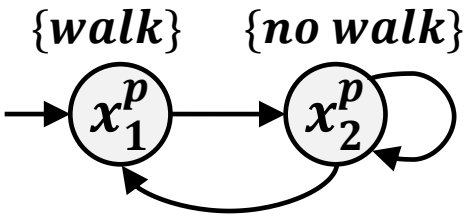
Example: Traffic Lights



A pedestrian crossing system

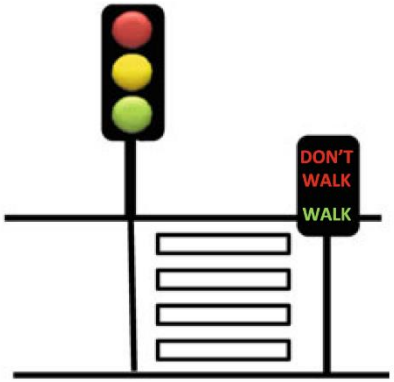


Car light system T_c



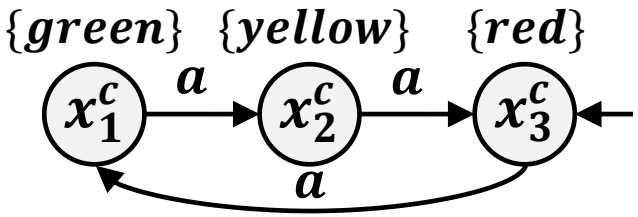
Pedestrian light system T_p

Example: Traffic Lights

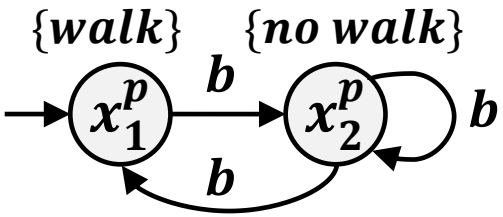


A pedestrian crossing system

Case 1: Two lights are fully asynchronized



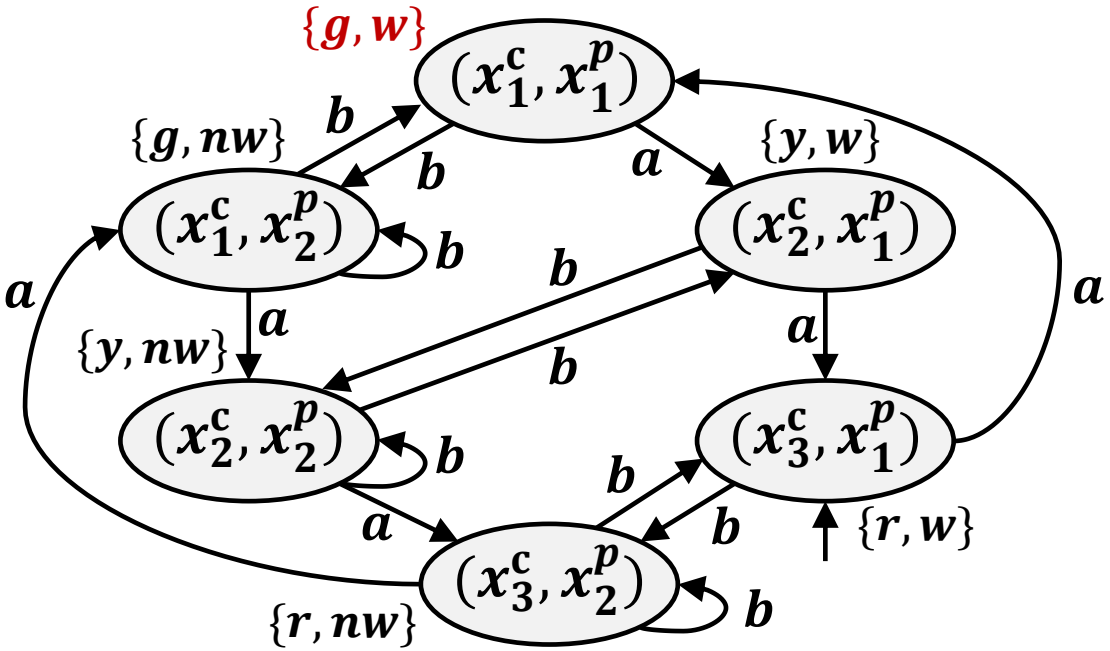
Car light system T_c



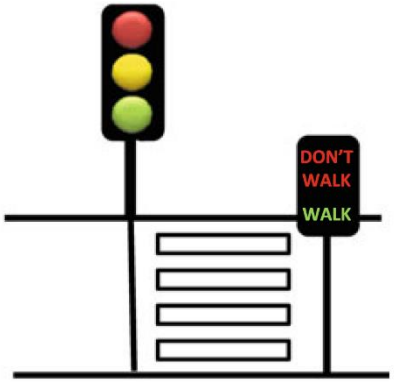
Pedestrian light system T_p

Product system $T_c \otimes T_p$

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

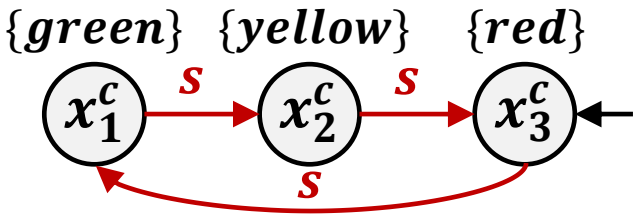


Example: Traffic Lights

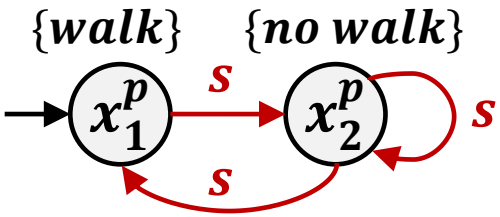


A pedestrian crossing system

Case 2: Two lights are fully synchronized



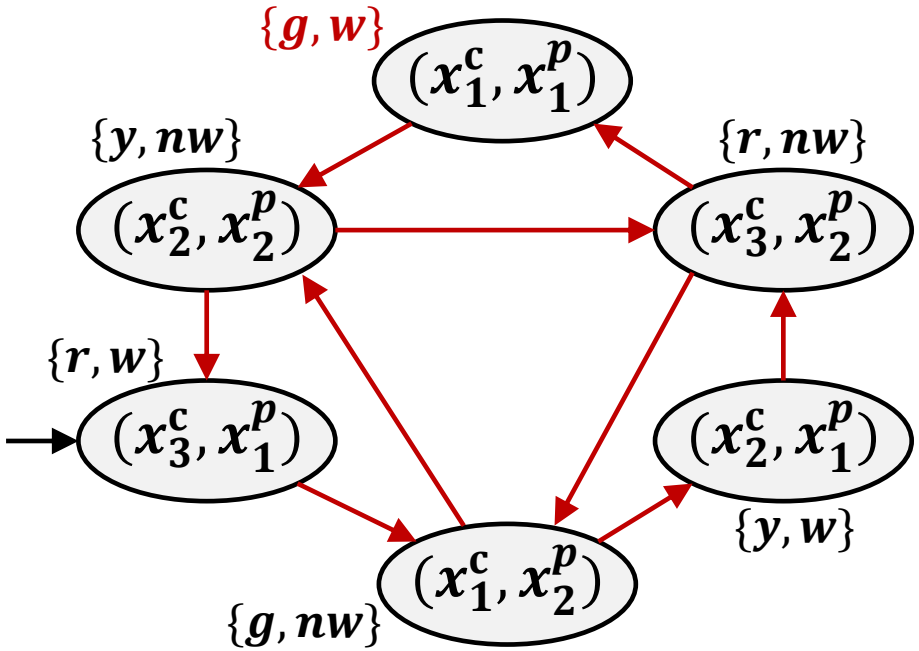
Car light system T_c



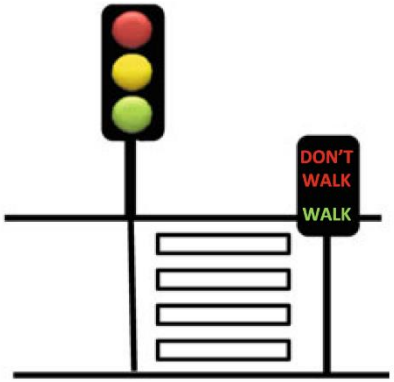
Pedestrian light system T_c

Product system $T_c \otimes T_p$

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

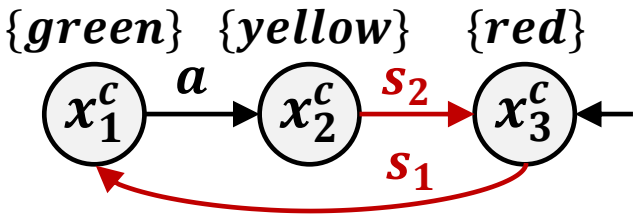


Example: Traffic Lights

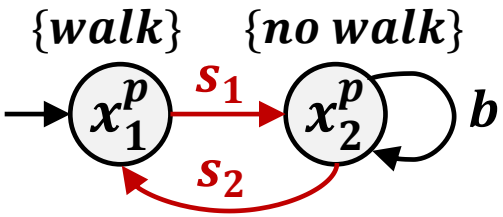


A pedestrian crossing system

Case 3: Two lights are partially synchronized



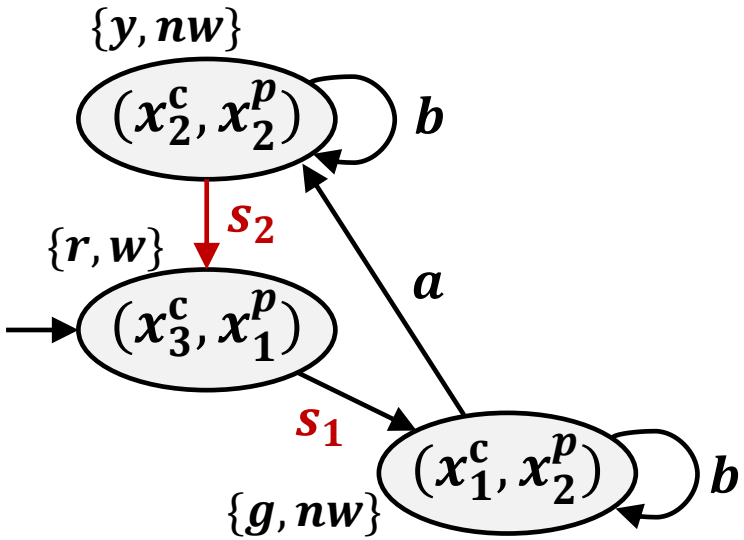
Car light system T_c



Pedestrian light system T_c

Product system $T_c \otimes T_p$

Safe!



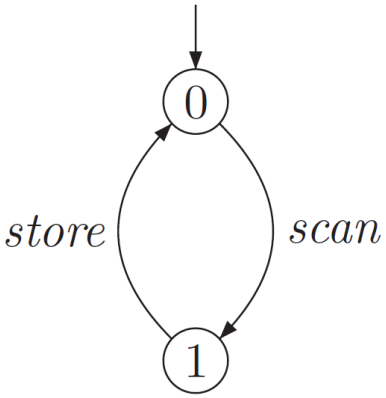
Comments on Product Composition

- 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

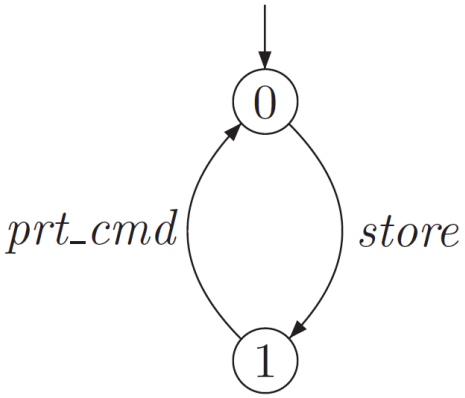
Stage Summary

- 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

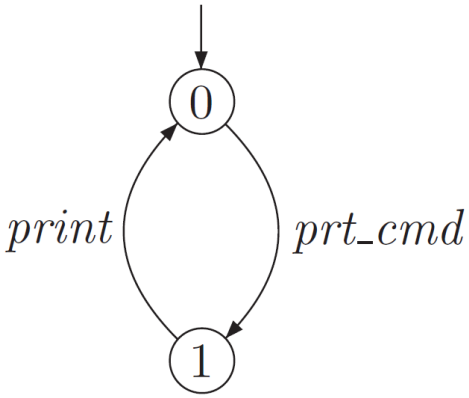
Question



Bar Code Reader (BCR)



Book Program (BP)



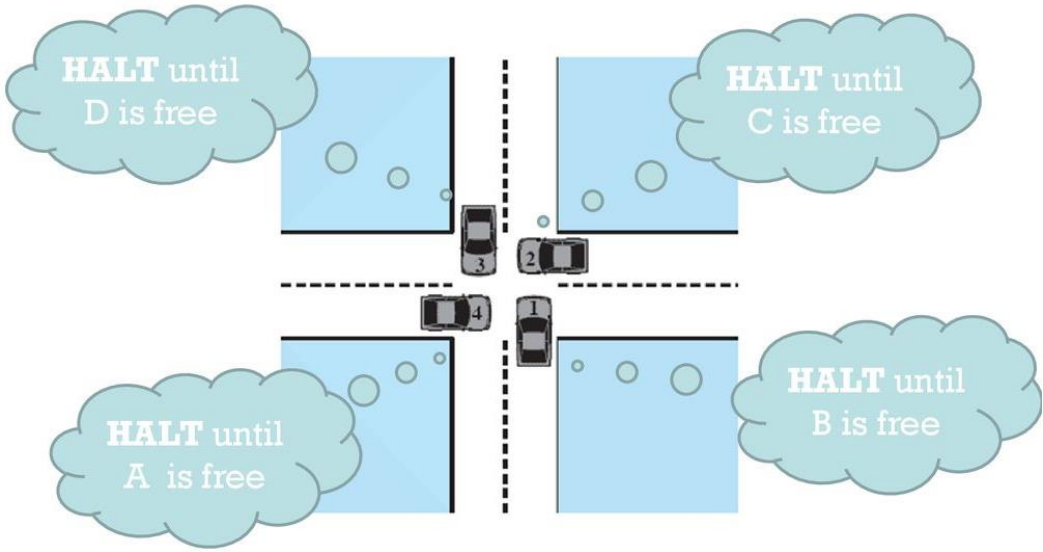
Printer

- 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 $BCR \otimes BP \otimes Printer = (BCR \otimes BP) \otimes Printer$

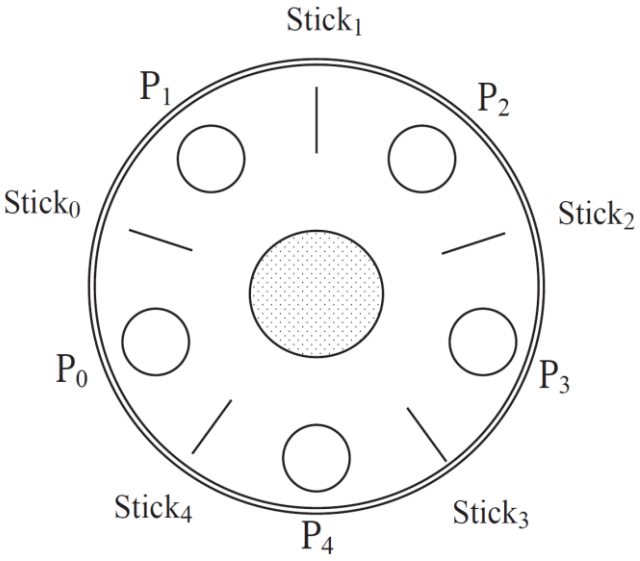
Formal Properties

Deadlock

- Sequential programs may have terminal states
- For parallel systems, however, computations typically do not terminate
- **Deadlock state**: $Post(x) = \emptyset$; a system with no deadlock is called **live**
- Therefore, deadlocks are undesirable and mostly represent a design error.
- A typical deadlock scenario occurs in the synchronization when components mutually wait for each other to progress.
- We **assume mostly the systems is live**; deadlock avoidance is another story.



Example: Dining Philosophers

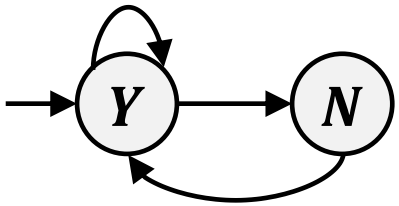


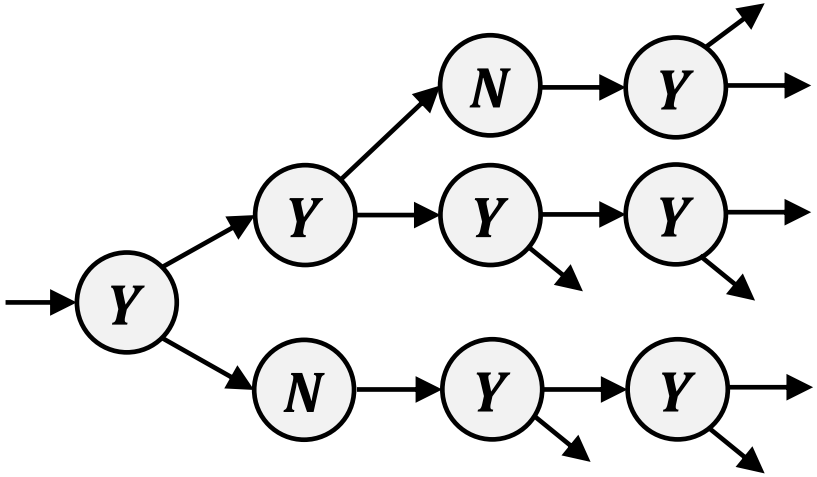
- To take food, each philosopher needs two sticks
- A deadlock occurs when all philosophers possess a single stick
- The problem is to design a protocol such that the complete system is deadlock-free, i.e., at least one philosopher can eat and think infinitely often.
- A fair solution may be required with each philosopher being able to think and eat infinitely often (freedom of individual starvation)

A possible solution is to make the sticks available for only one philosopher at a time. It can be verified that this solution is deadlock- and starvation-free.

Linear-Time Property

- Recall $Trace(T) \subseteq \left(2^{AP}\right)^{\omega}$ is the set of infinite sequence generated by T
- A linear-time property P over AP is a subset of $\left(2^{AP}\right)^{\omega}$ specifying the traces that a transition system should exhibit
- We say that system T satisfies P, denoted by $T \vDash P$, if $Trace(T) \subseteq P$
- "Linear" is the opposite of "branching" not "nonlinear"
- LT property is on a specific infinite execution





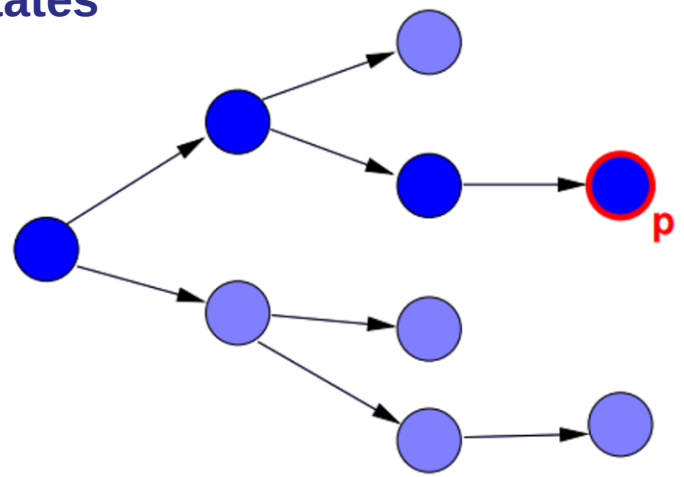
Safety & Invariant

- **Safety**: bad things never happen ⇔ always good things
- **Invariant**: some property should hold for all reachable state

An LT property P_{inv} is an **invariant** if there is a propositional logic formula Φ (called the invariant condition) over AP such that

$$P_{inv} = \{A_0 A_1 A_2 \cdots \in (2^{AP})^{\omega} : \forall i \geq 0, A_i \vDash \Phi\}$$

- Therefore, $T \vDash P_{inv}$ iff $L(x) \vDash \Phi$ for all reachable states
- Can be checked easily be a DFS or a BFS
- Mutual exclusion property: $\Phi = \neg crit_1 \vee \neg crit_2$
- Traffic light: $\Phi = \neg green \vee \neg walk$



Safety

- Invariant is essentially a state-based safety property
- In general, safety may impose requirements on finite path fragments
- Ex: Money can only be withdrawn from the ATM once a correct PIN has been provided; this is not invariant but is still safety

An LT property P_{safe} is a safety property if for all $\sigma \in (2^{AP})^{\omega} \setminus P_{safe}$ there exists a finite bad prefix $\hat{\sigma}$ of σ such

$$P_{safe} \cap \left\{\sigma' \in (2^{AP})^{\omega} : \hat{\sigma} \text{ is a finite prefix of } \sigma'\right\} = \emptyset$$

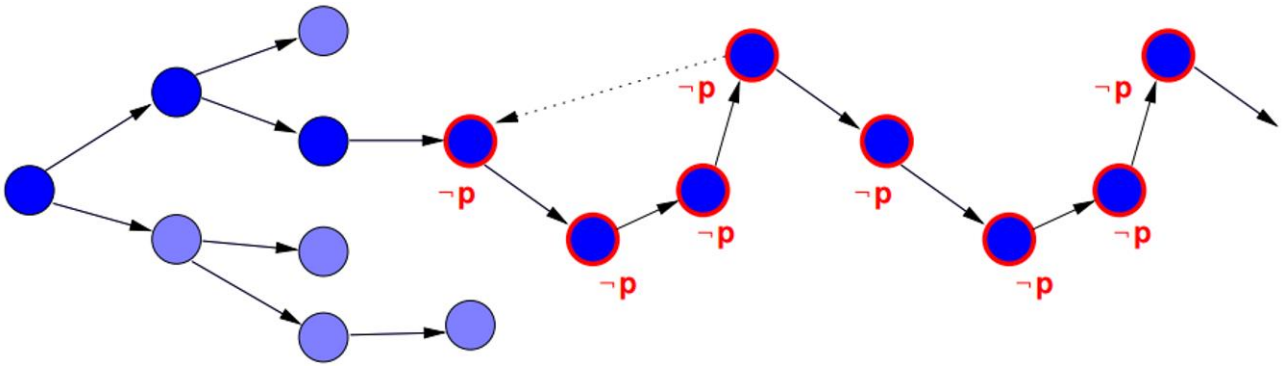
- $AP = \{red, yellow\}$
- red phase must be preceded immediately by a yellow phase
- Bad prefix: $\{yellow\}\emptyset\emptyset\{red\}$, $\{yellow\}\{yellow\}\{red\}\{red\}$

Liveness

- Safety says “something bad never happens”
- We also need liveness saying “something good will happen”
- Liveness should not constrain the finite behaviors, but require a certain condition on the infinite behaviors.
- For example, certain events occur infinitely often.

An LT property P_{live} is a **liveness property** if $pref(P_{live}) = (2^{AP})^*$

- $pref(P_{live})$ denotes the set of all finite prefix of P_{live}
- $(2^{AP})^*$ denotes the set all of finite words over 2^{AP}



Example: Liveness

- **Eventually**: each process will eventually enter its critical section:

  the set of all infinite words $A_0A_1 \cdots \in \left(2^{AP}\right)^{\omega}$ such that

  $$(\exists i \geq 0: crit_1 \in A_i) \wedge (\exists i \geq 0: crit_2 \in A_i)$$

- **Repeated eventually**: each process will enter its critical section infinitely often

  the set of all infinite words $A_0A_1 \cdots \in \left(2^{AP}\right)^{\omega}$ such that

  $$(\forall k \geq 0, \exists i \geq k: crit_1 \in A_i) \wedge (\forall k \geq, \exists i \geq k: crit_2 \in A_i)$$

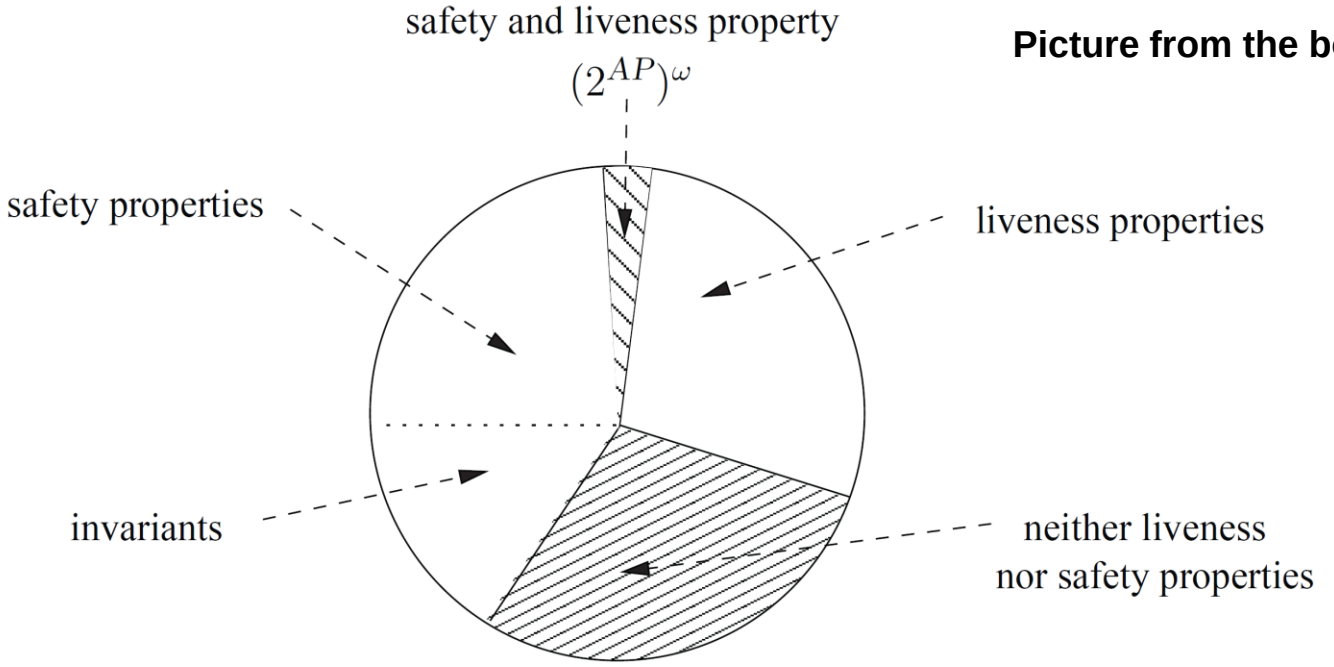
- **Starvation freedom**: each waiting process will eventually enter its critical section

  the set of all infinite words $A_0A_1 \cdots \in \left(2^{AP}\right)^{\omega}$ such that

  $$\left(\forall i \geq: wait_1 \in A_i \Rightarrow (\exists k > i: crit_1 \in A_k)\right) \wedge$$
  $$\left(\forall i \geq: wait_2 \in A_i \Rightarrow (\exists k > i: crit_2 \in A_k)\right)$$

Decomposition Theorem

Any LT property P can be decomposed as $P = P_{safe} \cap P_{live}$



Picture from the book of Baier and Katoen

- $P = (2^{AP})^{\omega}$ is the only property that is both safe and live
- In general, a property can be neither safe nor live
  - Consider $AP = \{a\}$ and $P =$ first $\emptyset$ and then $\{a\}$ infinitely often
  - It can be decomposed as $P = \emptyset(2^{AP})^{\omega} \cap \{\sigma: \{a\} \text{ infinitely often in } \sigma\}$

Stage Summary

- System having no deadlock will generate infinite sequences
- Linear-time properties evaluate infinite sequences
- Safety is a property that is violated in a finite horizon
- Liveness is a property that does not care about what have done
- In general, an LT property consists of both safety and liveness

Question

(a) If a becomes valid, afterward b stays valid ad infinitum or until c holds.

(b) Between two neighboring occurrences of a, b always holds.

(c) Between two neighboring occurrences of a, b occurs more often than c.

(d) $a \wedge \neg b$ and $b \wedge \neg a$ are valid in alternation or until c becomes valid.

Question: For each property, determine if it is a safety or liveness or both or none.

Bisimulation & Abstraction

Model Equivalence by Bisimulation

Motivations

- Different people may build different models for the same system
- Some models are complex but some are simple
- How to determine whether two models are describing the same thing?
- How to simplify a complex model to a simple but equivalent one?

Basic Ideas

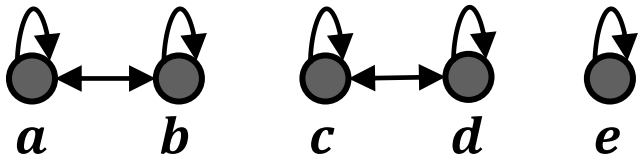
- Model equivalence is captured by "bisimulation"
- One model is more "precise" than the other if

  "no matter what you do, I can do the same thing" (simulation)
- Two models are equivalent if they can simulate each other

Equivalence Relation

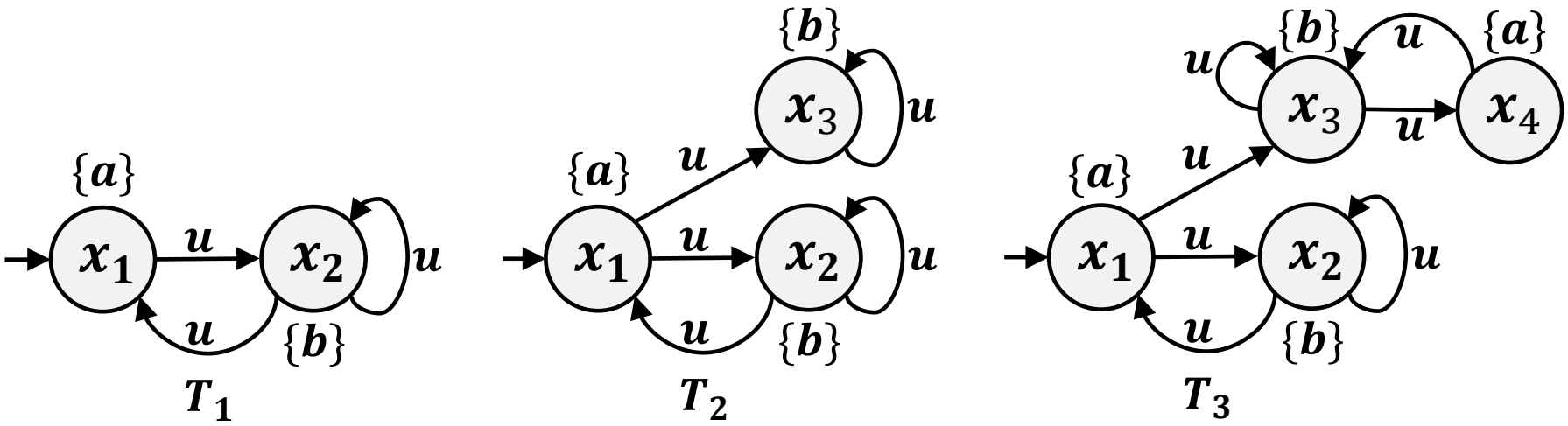
- a relation from set A to set B is a set of pairs $\sim \subseteq A \times B$
- we write $a \sim b$ if $(a, b) \in \sim$
- a relation $\sim \subseteq A \times A$ on A is an equivalence relation if it satisfies:
  - reflexivity: $\forall a \in A: a \sim a$
  - symmetry: $\forall a, b \in A$, if $a \sim b$, then $b \sim a$
  - transitivity: $\forall a, b, \in A$, if $a \sim b$ and $b \sim c$, then $a \sim c$
- an equivalent relation induces an equivalent class

$$A/_{\sim} = \{[a] \in 2^A : a \in A\}, \text{ where } [a] = \{b \in A : a \sim b\}$$



- $A = \{a, b, c, d, e\}$
- $\sim = \{(a,a), (a,b), (b,a), (b,b), (c,c), (c,d), (d,c), (d,d), (e,e)\}$
- $A/_{\sim} = \{\{a, b\}, \{c, d\}, \{e\}\}$
- $[a] = [b] = \{a, b\}, [c] = [d] = \{c, d\}, [e] = \{e\}$

Model Equivalence



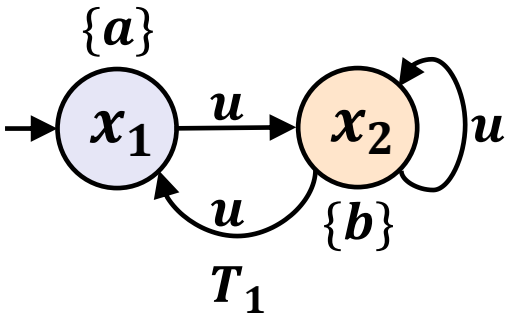
- $Trace(T_1) = Trace(T_2)$ but state x_3 is T_2 seems to be different
- $Trace(T_1) = Trace(T_3)$ but are they really equivalent?

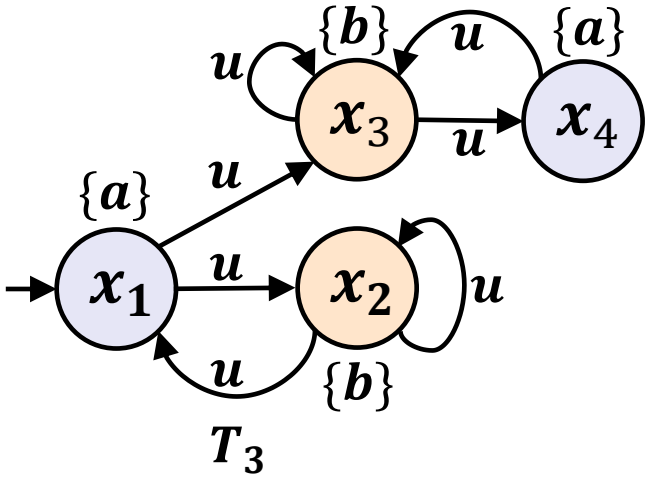
Observations

- trace equivalence is not good enough to describe model equivalence
- we needs to look at the equivalences of states

State Equivalence

- What does two states are “equivalent” mean?
  - they should have the same property (atomic propositions)
  - they should have the same future behaviors
- Two systems are equivalent if their initial states are equivalent
- For a system itself, we can aggregate equivalent states (abstraction)





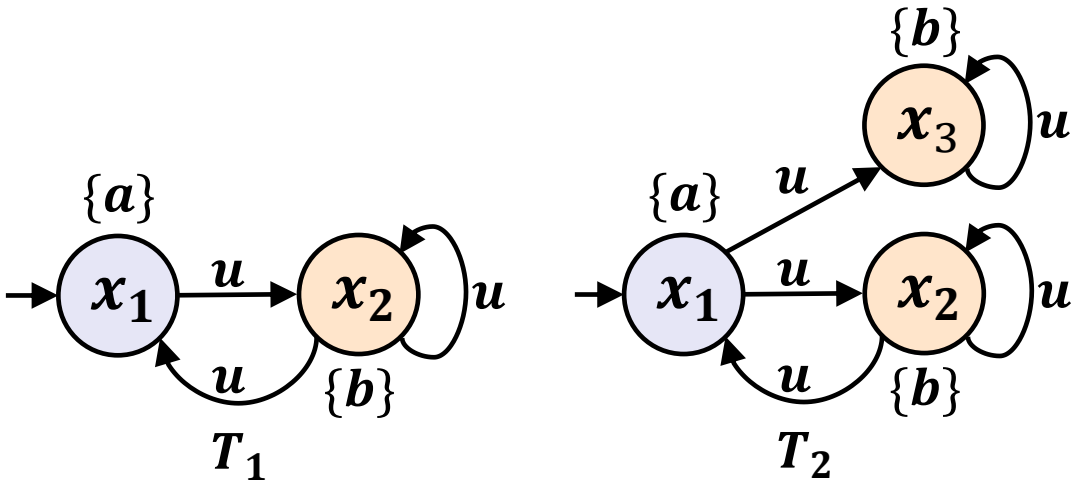
Simulation Relation

Let T_1 and T_2 be two LTSs, where $T_i = (X_i, U_i, \rightarrow_i, X_{0,i}, AP, L_i)$. Then a relation $\sim \subseteq X_1 \times X_2$ is a **simulation relation** from T_1 to T_2 if

- $\forall x_{0,1} \in X_{0,1}, \exists x_{0,2} \in X_{0,2}: x_{0,1} \sim x_{0,2}$
- for all $x_1 \sim x_2$, it holds that
  - $L_1(x_1) = L_2(x_2)$
  - If $x_1' \in Post(x_1)$ then there exists $x_2' \in Post(x_2)$ with $x_1' \sim x_2'$

We say T_1 is simulated by T_2 or T_2 simulates T_1, denoted by $T_1 \preccurlyeq T_2$ if there exists a simulation relation from T_1 to T_2

Example: Simulation Relation



- We have $T_1 \preccurlyeq T_2$.
- Consider relation $\sim = \{(x_1, x_1), (x_2, x_2), (x_3, x_3)\} \subseteq X_1 \times X_2$
- However, $T_2 \not\preccurlyeq T_2$ as x_3 cannot be related to any state in T_1

- $\forall x_{0,1} \in X_{0,1}, \exists x_{0,2} \in X_{0,2}: x_{0,1} \sim x_{0,2}$
- for all $x_1 \sim x_2$, it holds that
  - $L_1(x_1) = L_2(x_2)$
  - If $x_1' \in Post(x_1)$ then there exists $x_2' \in Post(x_2)$ with $x_1' \sim x_2'$

Bisimulation Relation

Let T_1 and T_2 be two LTSs, where $T_i = (X_i, U_i, \delta_i, X_{0,i}, AP, L_i)$. Then A relation $\sim \subseteq X_1 \times X_2$ is a **bisimulation relation** between T_1 to T_2 if

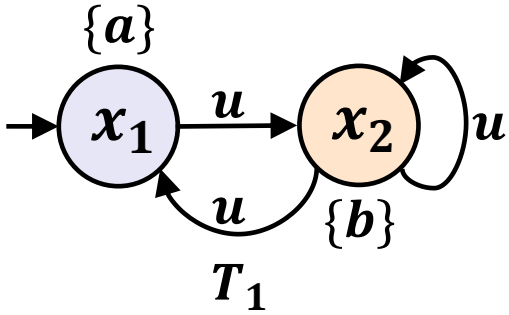
- $\forall x_{0,1} \in X_{0,1}, \exists x_{0,2} \in X_{0,2}: x_{0,1} \sim x_{0,2}$
- $\forall x_{0,2} \in X_{0,2}, \exists x_{0,1} \in X_{0,1}: x_{0,1} \sim x_{0,2}$
- for all $x_1 \sim x_2$, it holds that
  - $L_1(x_1) = L_2(x_2)$
  - if $x_1' \in Post(x_1)$ then there exists $x_2' \in Post(x_2)$ with $x_1' \sim x_2'$
  - if $x_2' \in Post(x_2)$ then there exists $x_1' \in Post(x_1)$ with $x_1' \sim x_2'$

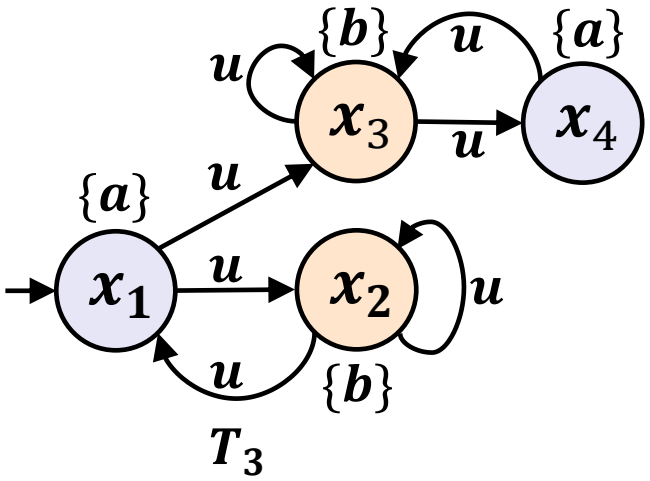
We say T_1 and T_2 are **bisimilar**, denoted by $T_1 \cong T_2$, if there exists a bisimulation relation between T_1 and T_2

Remark: bisimulation is equivalent to

- $\sim \subseteq X_1 \times X_2$ is a simulation relation from T_1 to T_2; and
- $\sim^{-1} \subseteq X_2 \times X_1$ is a simulation relation from T_2 to T_1.

Example: Bisimulation Relation





- We have $T_1 \cong T_3$
- Consider relation $\sim = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_3, x_3)\} \subseteq X_1 \times X_3$

- $\forall x_{0,1} \in X_{0,1}, \exists x_{0,2} \in X_{0,2} : x_{0,1} \sim x_{0,2}$
- $\forall x_{0,2} \in X_{0,2}, \exists x_{0,1} \in X_{0,1} : x_{0,1} \sim x_{0,2}$
- for all $x_1 \sim x_2$, it holds that
  - $L_1(x_1) = L_2(x_2)$
  - if $x_1' \in Post(x_1)$ then there exists $x_2' \in Post(x_2)$ with $x_1' \sim x_2'$
  - if $x_2' \in Post(x_2)$ then there exists $x_1' \in Post(x_1)$ with $x_1' \sim x_2'$

Fixed-Point Algorithm for Bisimulation

- Question: how to determine whether or not $T_1 \cong T_2$?
- Idea: first relate all pairs and then iterative shrink the relation

Define operator

$$F: 2^{X_1 \times X_2} \to 2^{X_1 \times X_2}$$

by: for any $R \subseteq X_1 \times X_2$, we have $(x_1, x_2) \in F(R)$ if

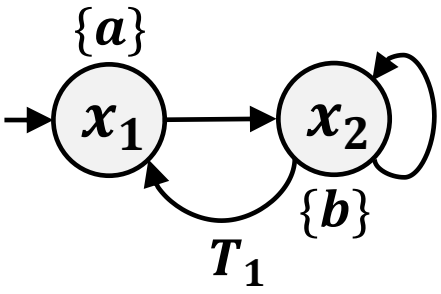
- $(x_1, x_2) \in R$
- $\forall x_1' \in Post(x_1), \exists x_2' \in Post(x_2): (x_1', x_2') \in R$
- $\forall x_2' \in Post(x_2), \exists x_1' \in Post(x_1): (x_1', x_2') \in R$

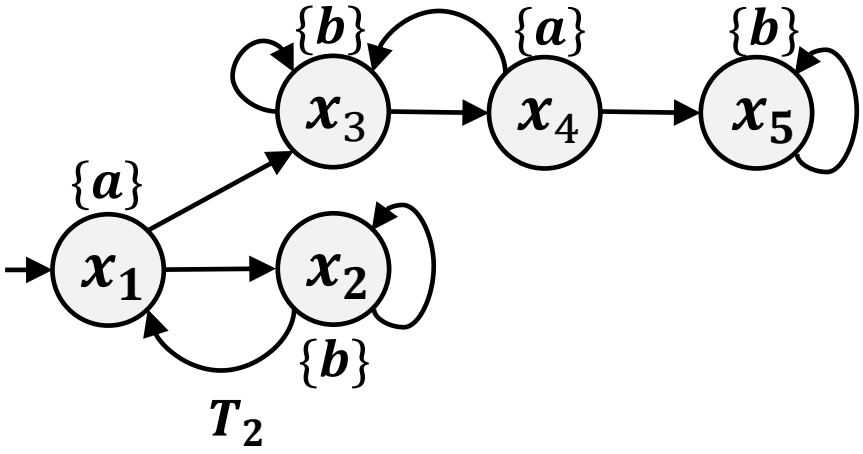
Then the fixed-point

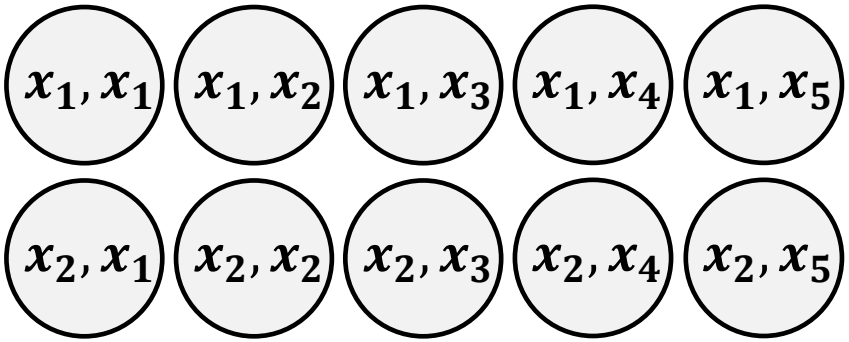
$$\lim_{k \to \infty} F^k(R_0), \text{ where } R_0 = \{(x_1, x_2): L_1(x_1) = L_2(x_2)\}$$

is the maximal bisimulation relation between T_1 and T_2.

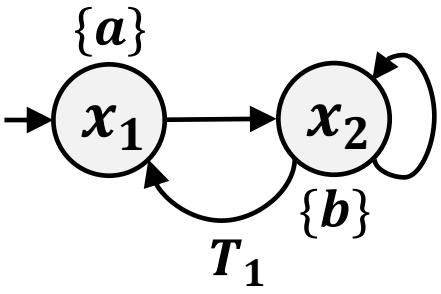
Example: Fixed-Point Iteration

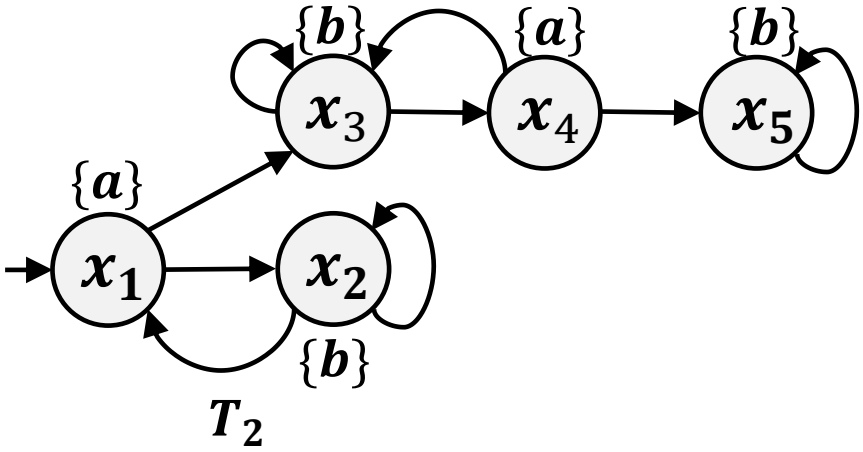


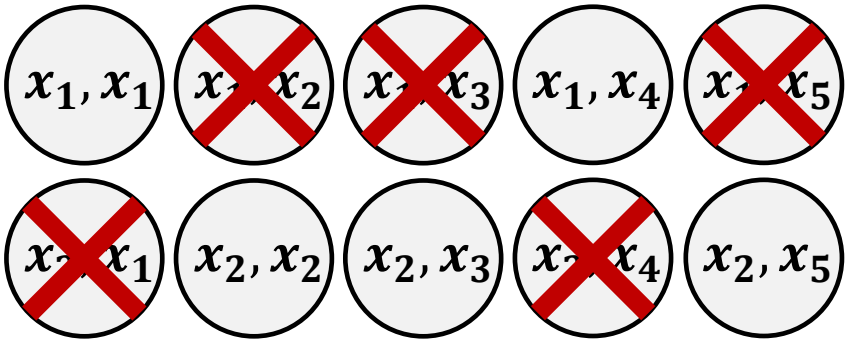




Example: Fixed-Point Iteration

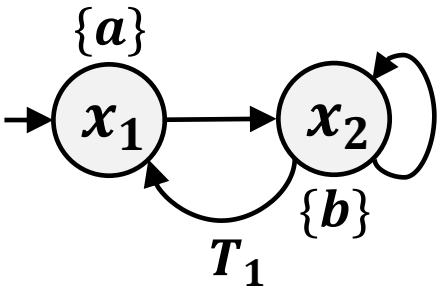


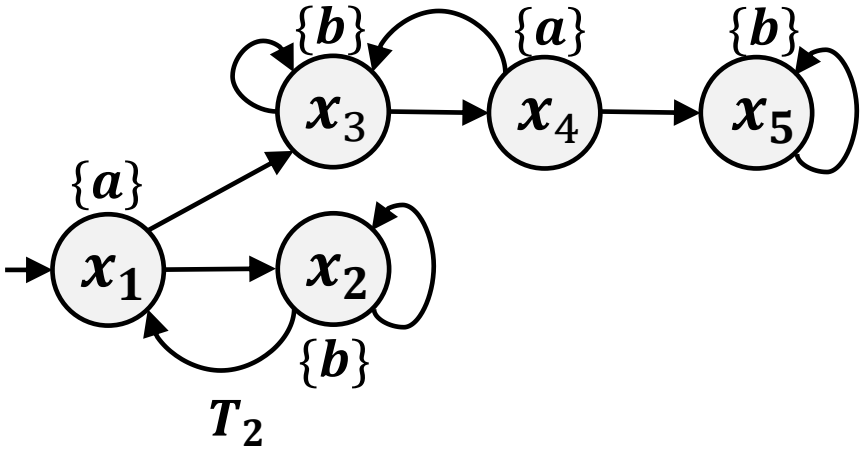


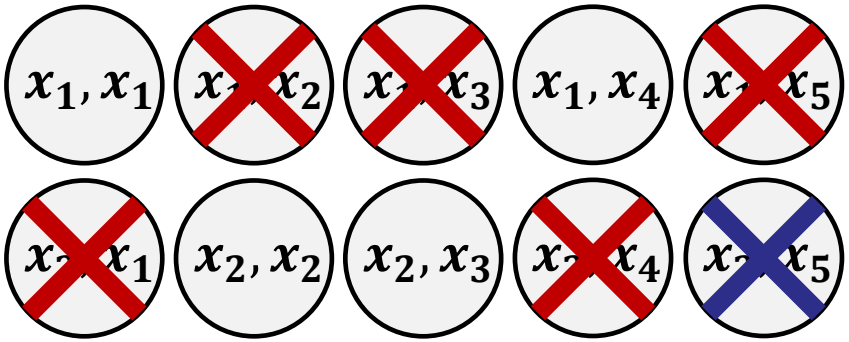


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$

Example: Fixed-Point Iteration

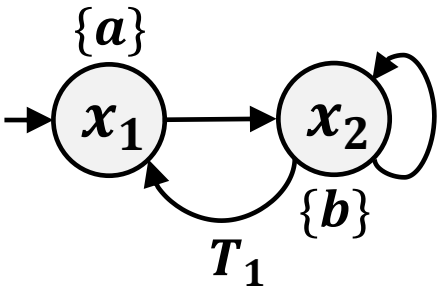


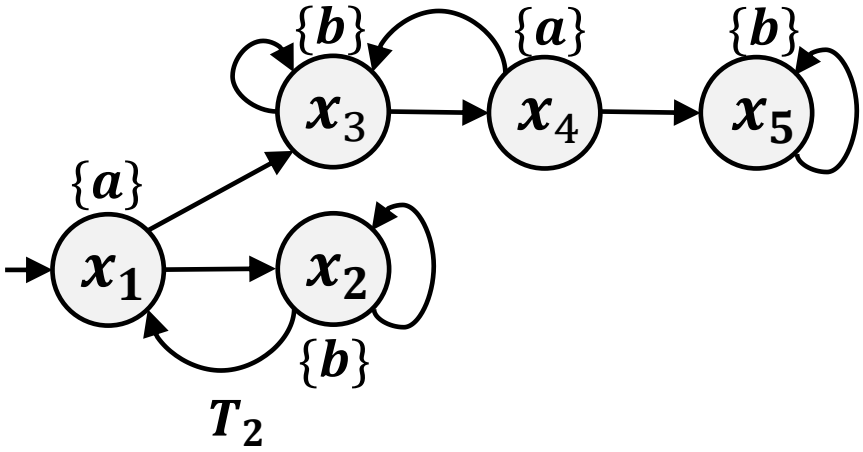


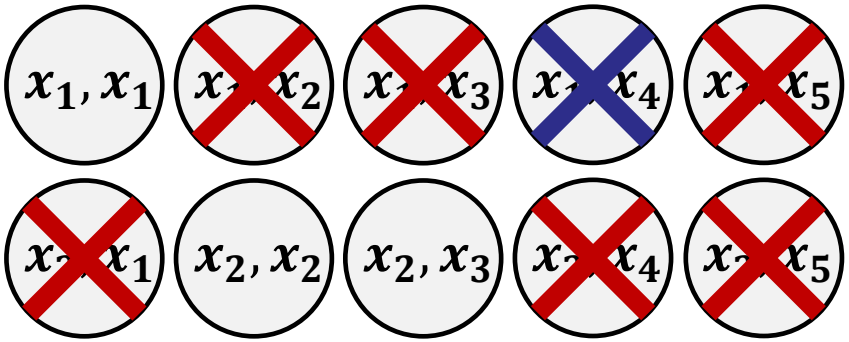


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$

Example: Fixed-Point Iteration

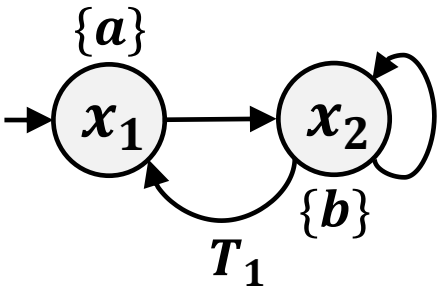


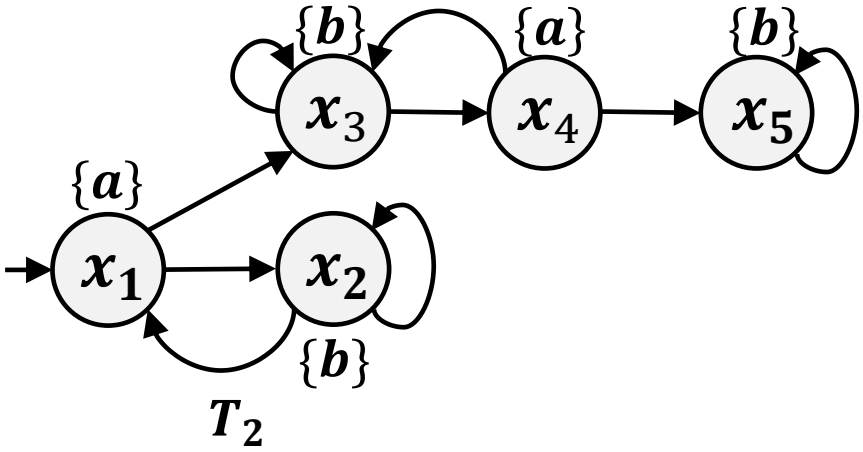


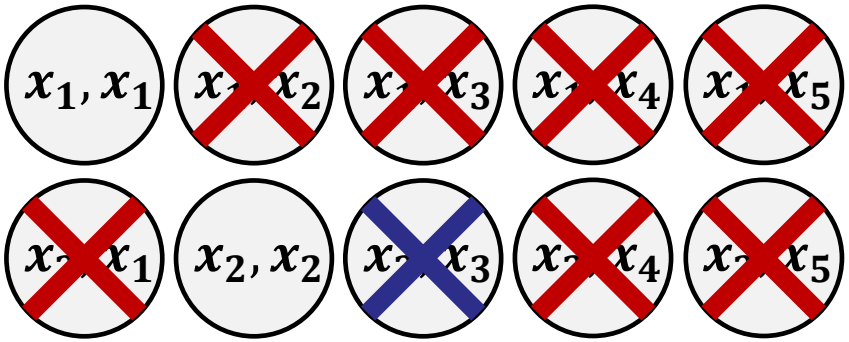


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$
- $R_2 = F(R_1) = \{(x_1, x_1), (x_2, x_2), (x_2, x_3)\}$

Example: Fixed-Point Iteration

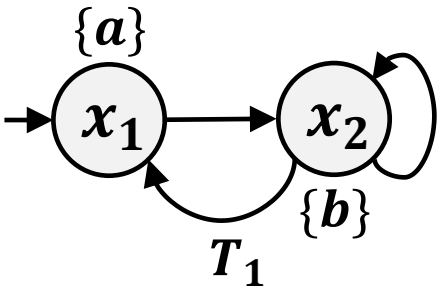


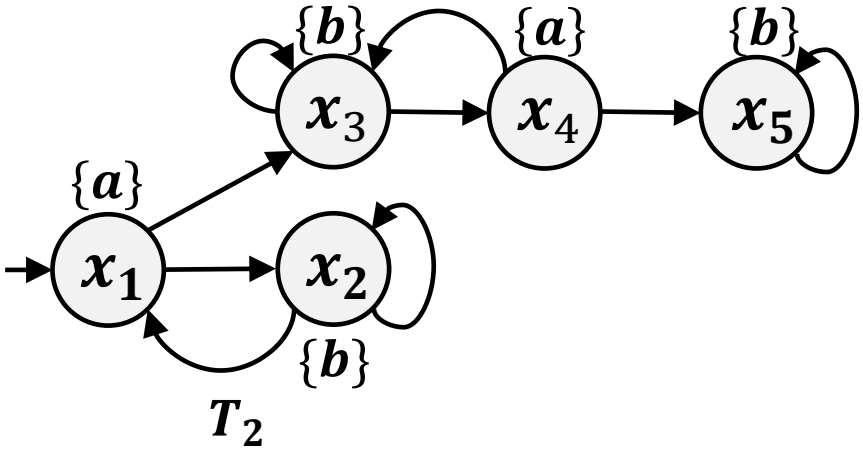


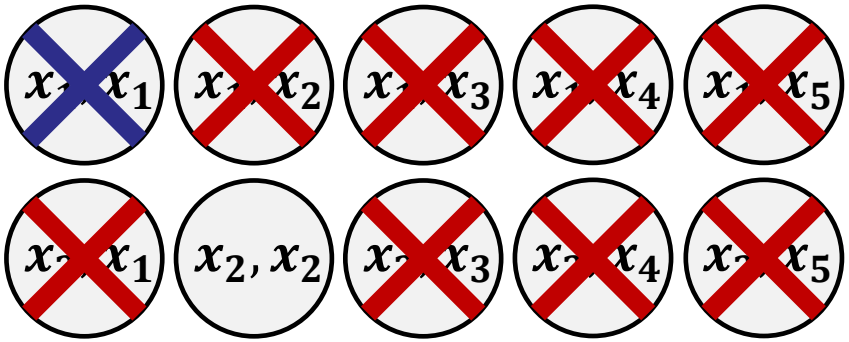


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$
- $R_2 = F(R_1) = \{(x_1, x_1), (x_2, x_2), (x_2, x_3)\}$
- $R_3 = F(R_2) = \{(x_1, x_1), (x_2, x_2)\}$

Example: Fixed-Point Iteration

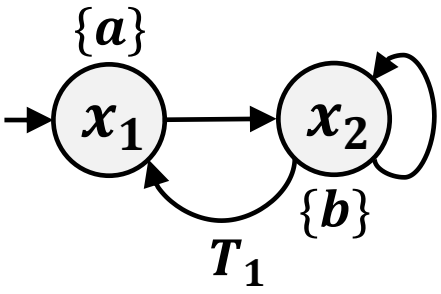


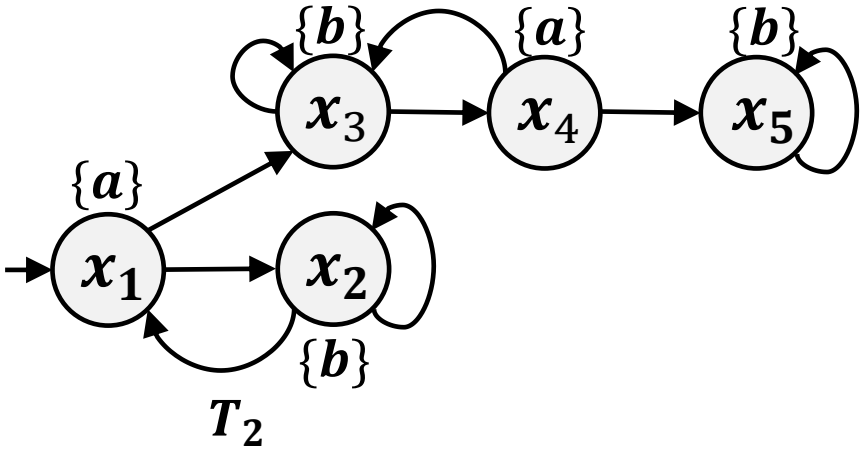


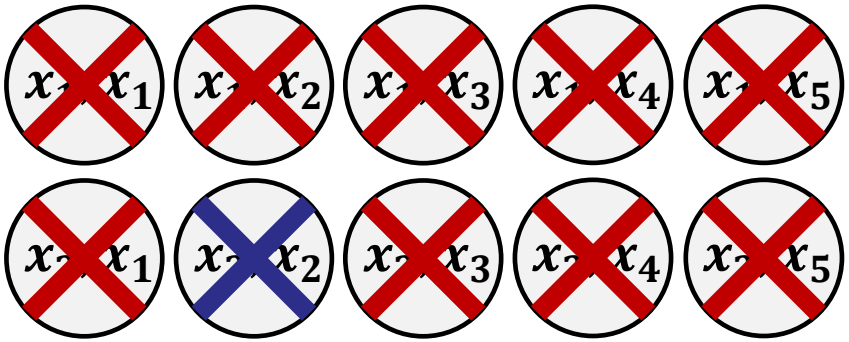


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$
- $R_2 = F(R_1) = \{(x_1, x_1), (x_2, x_2), (x_2, x_3)\}$
- $R_3 = F(R_2) = \{(x_1, x_1), (x_2, x_2)\}$
- $R_4 = F(R_3) = \{(x_2, x_2)\}$

Example: Fixed-Point Iteration

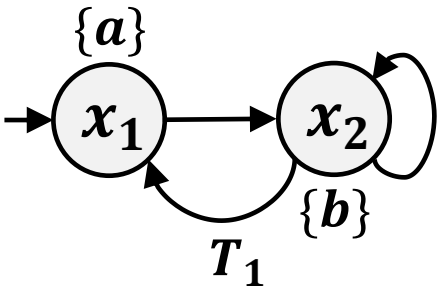


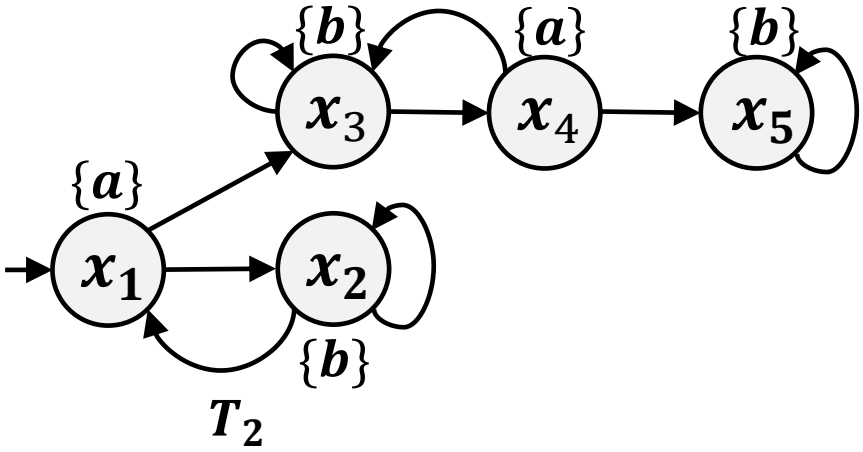


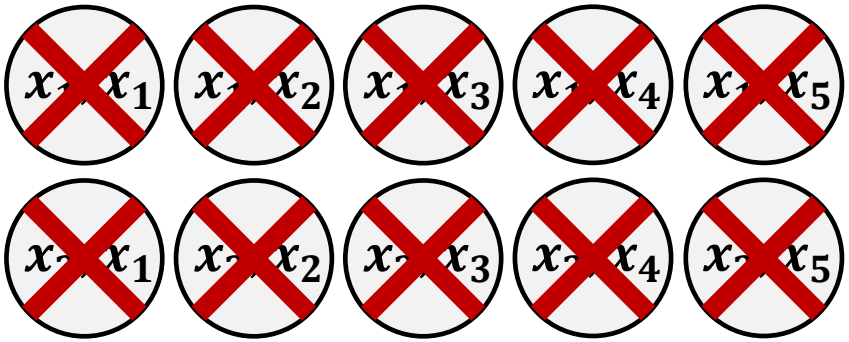


- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$
- $R_2 = F(R_1) = \{(x_1, x_1), (x_2, x_2), (x_2, x_3)\}$
- $R_3 = F(R_2) = \{(x_1, x_1), (x_2, x_2)\}$
- $R_4 = F(R_3) = \{(x_2, x_2)\}$
- $R_5 = F(R_4) = \emptyset$

Example: Fixed-Point Iteration







- $R_0 = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3), (x_2, x_5)\}$
- $R_1 = F(R_0) = \{(x_1, x_1), (x_1, x_4), (x_2, x_2), (x_2, x_3)\}$
- $R_2 = F(R_1) = \{(x_1, x_1), (x_2, x_2), (x_2, x_3)\}$
- $R_3 = F(R_2) = \{(x_1, x_1), (x_2, x_2)\}$
- $R_4 = F(R_3) = \{(x_2, x_2)\}$
- $R_5 = F(R_4) = \emptyset$

$T_1 \ncong T_2$!

Comment on Bisimulation

- Simulation implies trace inclusion, i.e.,

$$T_1 \preccurlyeq T_2 \Rightarrow Trace(T_1) \subseteq Trace(T_2)$$

- Bisimulation implies trace equivalence, i.e.,

$$T_1 \cong T_2 \Rightarrow Trace(T_1) = Trace(T_2)$$

- The vice versa is not true in general

- What if we also want to match control inputs?
  Change the definitions of the operator to

$$\forall x_1' \in Post(x_1, \boldsymbol{u}), \exists x_2' \in Post(x_2, \boldsymbol{u}) \cdots$$

Bisimulation on Itself

- For a single system T, we can compute the maximal bisimulation relation $\sim \subseteq X \times X$ between T and itself (by the fixed-point alg.)
- Note that such a relation $\sim$ is always non-empty. Why? since a state should be equivalent to itself, i.e., the identity relation is included in $\sim$
- Relation $\sim$ is in fact an equivalent relation telling which states are equivalent in terms of both the current property and the future
- Therefore, we can aggregate equivalent states and treat them as a new state (the equivalent classes)
- In this way, we are able to abstract the system model without losing any information

Quotient-Based Abstraction

Let $T = (X, U, \delta, X_0, AP, L)$ be an LTS and $\sim \subseteq X \times X$ be an equivalence relation on X. Then $\sim$ induces a quotient transition system

$$T/_\sim = (X/_\sim, U, \rightarrow_\sim, X/_{\sim,0}, AP, L_\sim)$$

- $X/_\sim$ is the quotient space (the set of all equivalence classes) with $X/_{\sim,0} = \{[x] \in 2^X : [x] \cap X_0 \neq \emptyset\}$
- for $X_1, X_2 \in X/_\sim$ and $u \in U$, we have

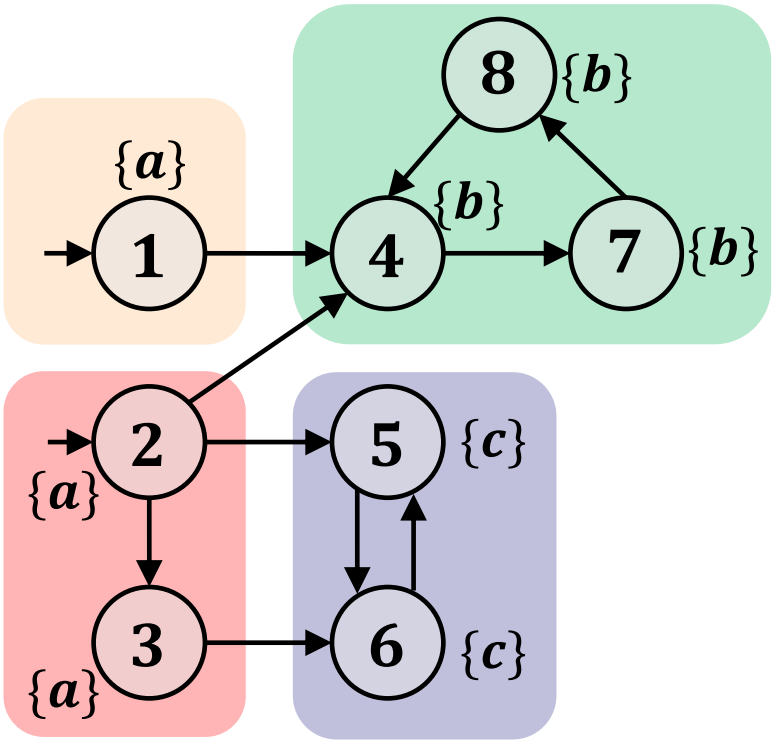
$$X_1 \xrightarrow[\sim]{u} X_2 \Leftrightarrow \exists x_1 \in X_1, x_2 \in X_2 : x_1 \xrightarrow{u} x_2$$

- $L/_\sim(x) = L(x)$ for all $x \in X$

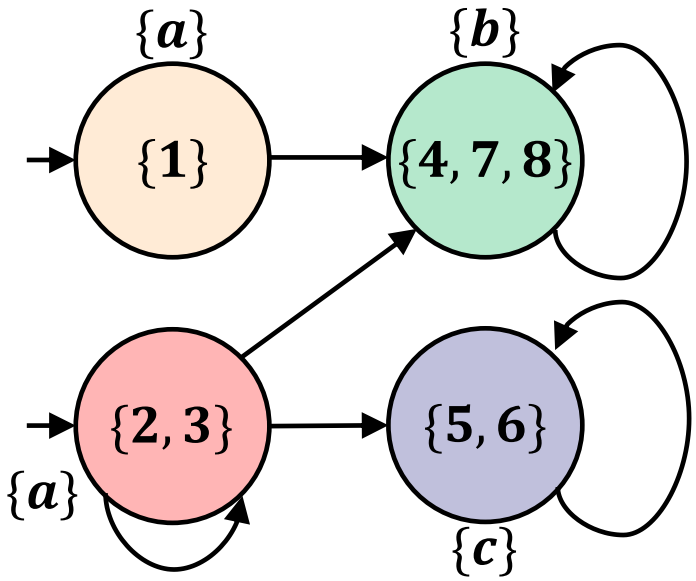
Theorem

- for any $\sim \subseteq X \times X$, we have $T \preccurlyeq T/_\sim$
- if $\sim \subseteq X \times X$ is a bisimulation relation for T, then $T \cong T/_\sim$

Example: Quotient System



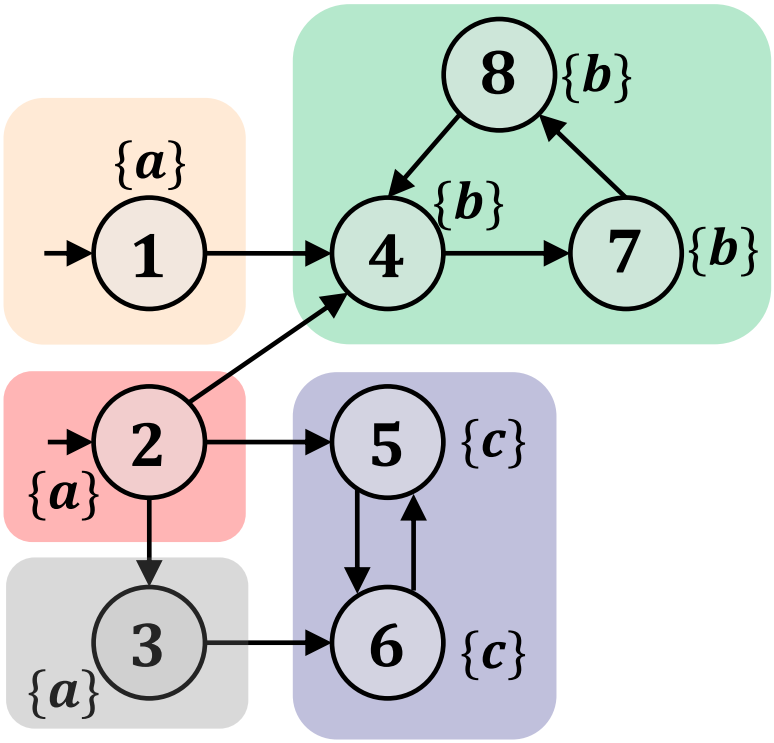
original system T



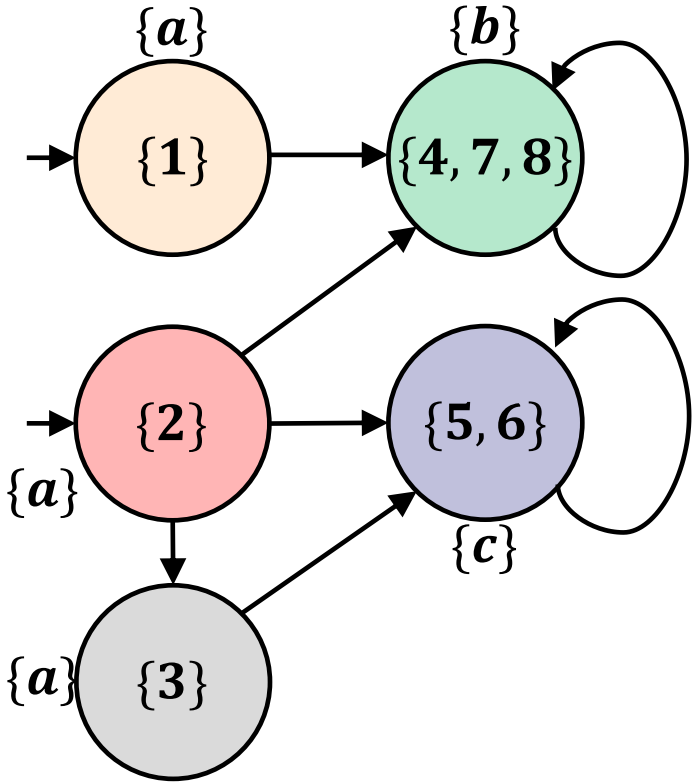
quotient system $T/_{\sim}$

- Consider equivalence relation shown by the colors
- Trivially, we have $T \preccurlyeq T/_{\sim}$
- However, $T \not\cong T/_{\sim}$ since $\sim$ is not a bisimulation (consider states $2\&3$)

Example: Quotient System



original system T



quotient system $T/_{\sim}$

- Since $\sim \subseteq X \times X$ is a bisimulation relation on T
- This time we have $T \cong T/_{\sim}$

Stage Summary

- Simulation means “no matter what you do, I can match it and preserve the ability of matching in the future”
- Two states are equivalent if they have both the same property and the same future behaviors
- Two systems are equivalent if they can simulation each other
- By aggregating equivalent states, one can build the quotient system that bisimulates the original system
- Bisimulation implies trace equivalent; hence preserves LT properties

Linear Temporal Logics

Describe LT Properties by LTL

Motivations

- It is not practical to write down $P \subseteq \left(2^{AP}\right)^{\omega}$ directly
- Propositional and predicate logics are “static”
- How to express temporal properties in a structured, user-friendly and rigorous manner

Approach

- Use linear temporal logics (LTL)
- Introduce temporal operators in addition to Boolean opertators

LTL Syntax

A (propositional) **Linear Temporal Logic (LTL)** formula ϕ over a given set of atomic proposition AP is recursively defined as

$$\phi ::= \text{TRUE} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \Diamond\phi \mid \phi_1 U \phi_2$$

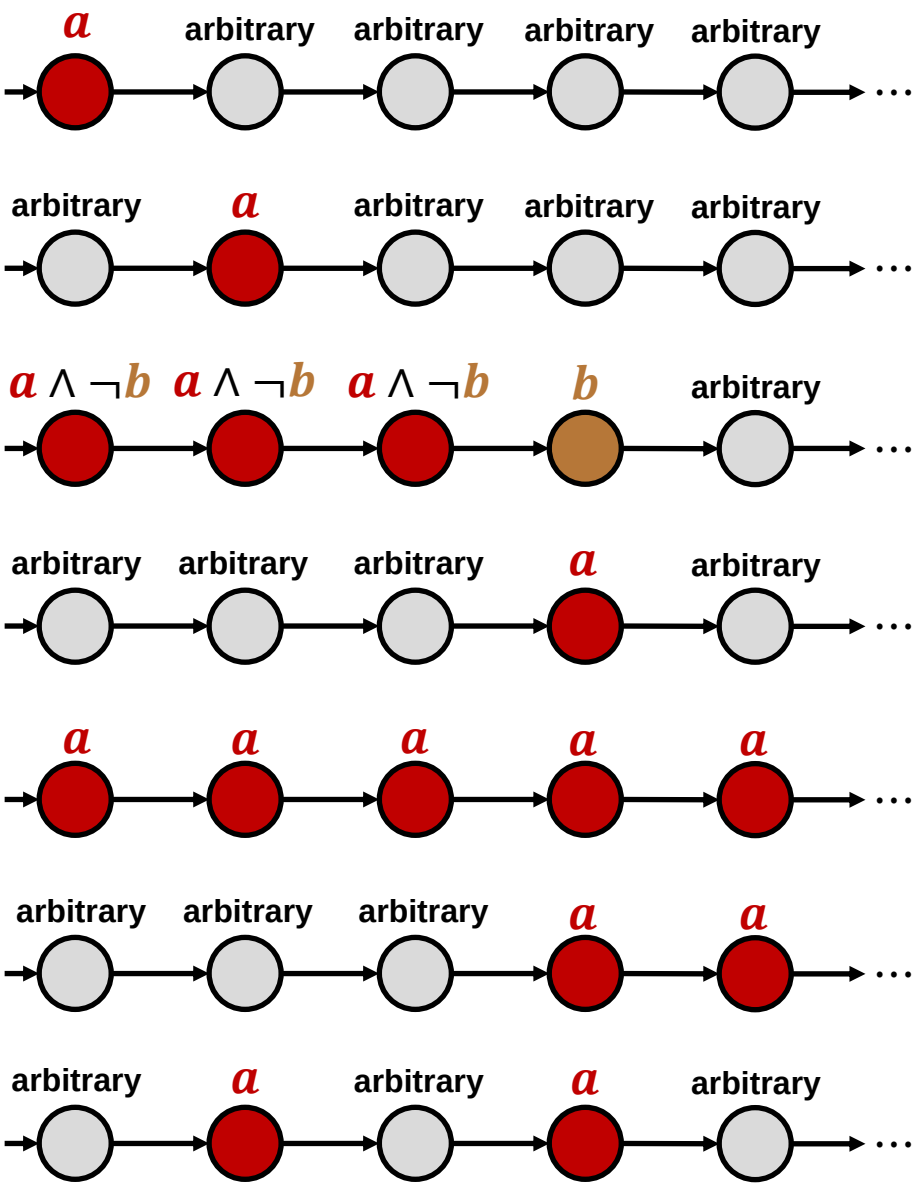
where a is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

- Formula $\bigcirc\phi$ holds at the current moment, if ϕ holds in the **next** "step"
- Formula $\phi_1 U \phi_2$ holds at the current moment, if there is some future moment for which ϕ_2 holds and ϕ_1 holds at all moments **until** that future moment.

We can also define the following operators

- Temporal oper..: "**eventually**" $\Diamond\phi \coloneqq \text{TRUE}\ U\ \phi$, "**always**" $\Box\phi \coloneqq \neg\Diamond\neg\phi$
- Boolean oper.: "**or**" $\phi_1 \vee \phi_2 \coloneqq \neg(\neg\phi_1 \wedge \neg\phi_2)$, "**implies**" $\phi_1 \rightarrow \phi_2 \coloneqq \neg\phi_1 \vee \phi_2$

LTL Semantics: Informal



LTL Semantics: Formal

- LTL formulas are used to evaluate infinite words over 2^{AP}
- Let $\sigma = A_0A_1A_2 \cdots \in (2^{AP})^{\omega}$, where $\sigma[j \cdots] = A_jA_{j+1}A_{j+2} \cdots$
- Infinite word σ satisfies formula ϕ, denoted by $\sigma \vDash \phi$, is defined by

  - $\sigma \vDash \text{True}$
  - $\sigma \vDash a$ iff $a \in A_0$ (i.e., $A_0 \vDash a$)
  - $\sigma \vDash \phi_1 \wedge \phi_2$ iff $\sigma \vDash \phi_1$ and $\sigma \vDash \phi_2$
  - $\sigma \vDash \neg\phi$ iff $\sigma \nvDash \phi$
  - $\sigma \vDash \Diamond\phi$ iff $\sigma[1 \cdots] = A_1A_2A_3 \cdots \vDash \phi$
  - $\sigma \vDash \phi_1 U \phi_2$ iff $\exists j \geq 0: \sigma[j \cdots] \vDash \phi_2$ and $\forall 0 \leq i < j: \sigma[i \cdots] \vDash \phi_1$

- The set of all words satisfying ϕ is $Word(\phi) = \{\sigma \in (2^{AP})^{\omega} : \sigma \vDash \phi\}$, i.e., $\sigma \vDash \phi$ iff $\sigma \in Word(\phi)$

LTL Example: Mutual Exclusion

- The safety property stating that P_1 and P_2 never simultaneously have access to their critical sections

$$\square(\neg crit_1 \vee \neg crit_2)$$

- The liveness requirement stating that each process P_i is infinitely often in its critical section

$$(\square\diamond crit_1) \wedge (\square\diamond crit_2)$$

- The strong fairness requirement stating that infinitely waiting process will eventually enter its critical section infinitely often:

$$(\square\diamond wait_1 \rightarrow \square\diamond crit_1) \wedge (\square\diamond wait_2 \rightarrow \square\diamond crit_2)$$

LTL Example: Traffic Light

- The traffic light is infinitely often green

$$\square\diamond green$$

- Once red, the light cannot become green immediately

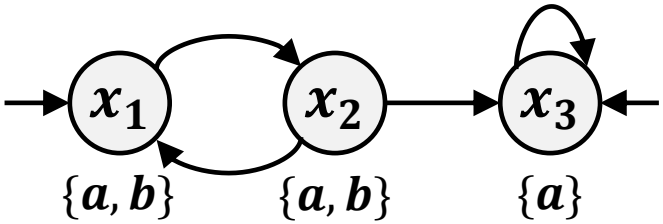
$$\square(red \rightarrow \neg\bigcirc green)$$

- Once red, the light always becomes green eventually after being yellow for some time

$$\square(red \rightarrow \bigcirc(red\ U\ (yellow \wedge \bigcirc(yellow\ U\ green))))$$

LTL Semantics on LTSs

- LTL formula ϕ evaluates infinite words over 2^{AP}
- LTS T generates a set of infinite words (traces) from initial states
- A state $x \in X$ in T satisfies ϕ, denoted by $x \models \phi$, if all traces generated from x satisfy ϕ
- We say LTL T satisfies ϕ, denoted by $T \models \phi$, if all its initial stats satisfy ϕ, i.e., $Trace(T) \subseteq Word(\phi)$

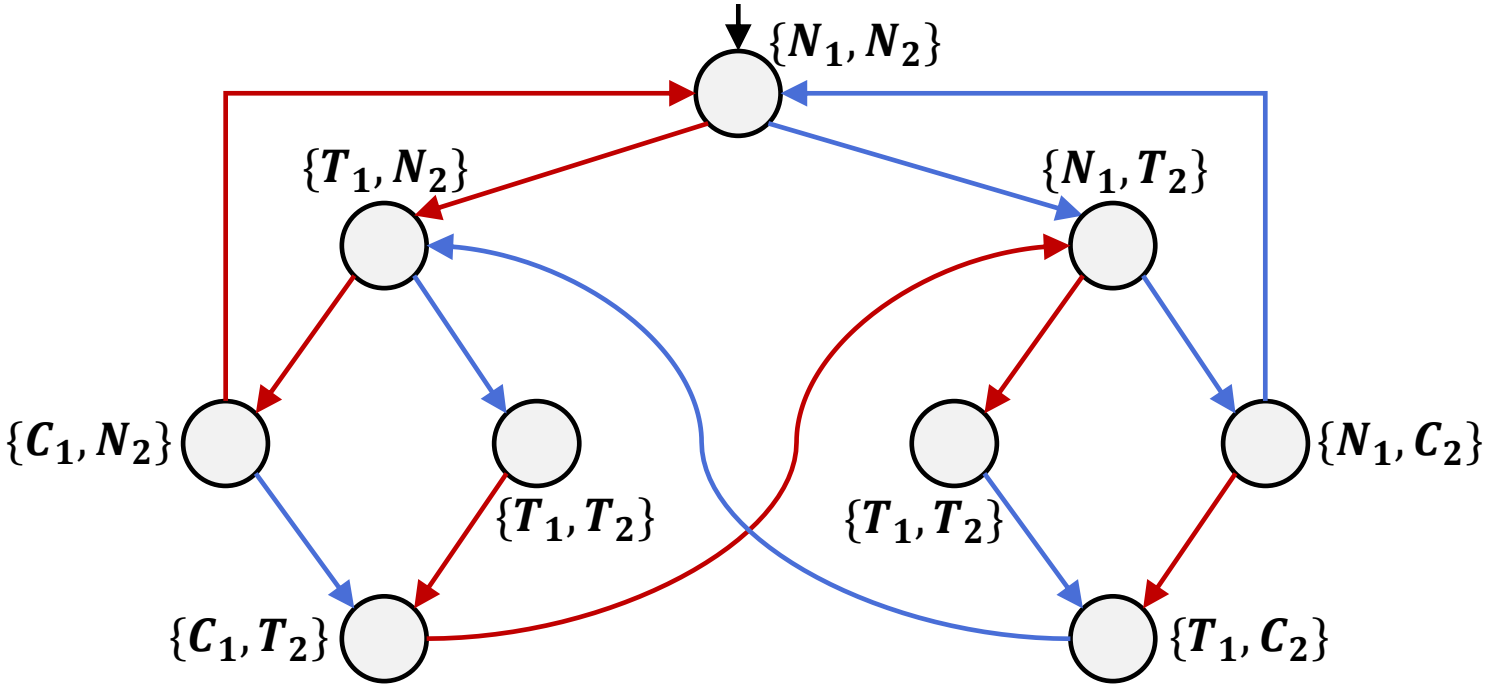


LTS T

- $T \models \Box a$
- $T \not\models \bigcirc(a \wedge b)$ since $x_1 \models \bigcirc(a \wedge b)$ but $x_3 \not\models \bigcirc(a \wedge b)$
- $T \models \Box(\neg b \rightarrow \Box(a \wedge \neg b))$
- $T \not\models b\ U\ (a \wedge \neg b)$ since $\{a, b\}^{\omega} \not\models b\ U\ (a \wedge \neg b)$

How to check whether $T \models \phi$ or not?

LTL Example: Mutually Exclusive Processes



- $T \models \square(T_1 \rightarrow \diamond C_1)$? **Yes!**
- $T \models \square\diamond C_1$ **No! Consider trace** $(\{N_1, N_2\}\{N_1, T_2\}\{N_1, C_2\})^{\omega}$
- $T \models \square\diamond T_1 \rightarrow \square\diamond C_1$? **Yes!**

Co-Safe LTL Syntax

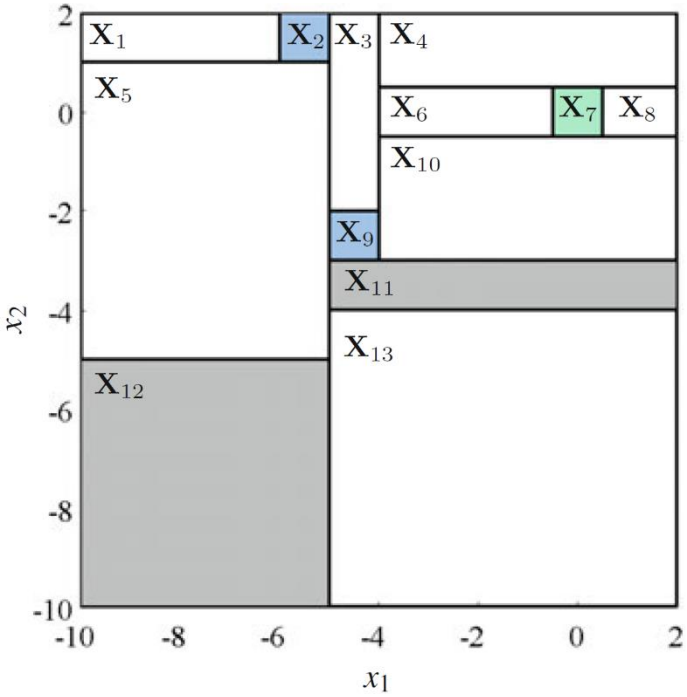
A (propositional) Co-Safe Linear Temporal Logic (scLTL) formula ϕ over a given set of atomic proposition AP is recursively defined as

$$\phi ::= \text{TRUE} \mid a \mid \neg a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 U \phi_2$$

where a is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

- Negation can only be used for atomic propositions not a general formula
- "Always" cannot be expressed since $\square \phi := \neg \diamond \neg \phi$ is not well defined
- We can only use temporal operators $\bigcirc, U$ and $\diamond$
- Any infinite word satisfying scLTL ϕ has a finite "good" prefix such that any infinite continuation of this good prefix satisfies ϕ
- Denote $\mathcal{L}_{pref,\phi}$ as the set of finite good prefixes of scLTL formula ϕ

Example: Co-Safe LTL Syntax



Consider an agent moving in the planar environment

- **Visit regions X_2 or X_9 and then the target region X_7, while avoiding X_{11} and X_{12}, and staying inside $X_2 X = [-10\ \ 2]^2$ until the target region is reached.**

$$\boldsymbol{\phi = ((\neg X_{11} \wedge \neg X_{12} \wedge \neg Out)\ U\ X_7) \wedge (\neg X_7\ U\ (X_2 \vee X_9))}$$

- **Good prefix, e.g., $X_2X_3X_4X_7$ or $X_2X_3X_9X_3X_9X_{10}X_8X_7$**
- **In general, there may have infinite many finite good prefixes**

Computation Tree Logic

- LTL implicitly quantifies universally over paths

  $\langle T, x \rangle \vDash \phi$ iff for every path π starting at x, we have $\langle T, \pi \rangle \vDash \phi$

- Properties that assert the existences of a path cannot be expressed, e.g., always has the possibility to reach some states.

- The computation tree logic (CTL) solves this problem. The idea is to evaluate over branching-time structures (trees) with path quantifiers:

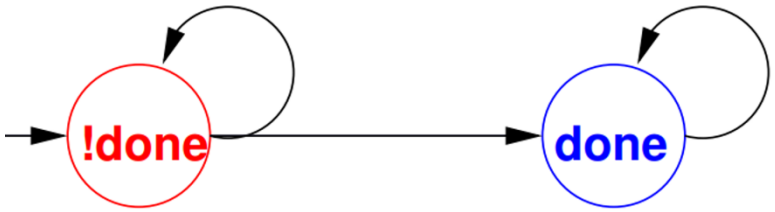
  - For All Paths: A
  - Exists a Path: E
  - Every temporal operator preceded by a path quantifier
  - Notation: $\Box \rightsquigarrow G$ globally in the future

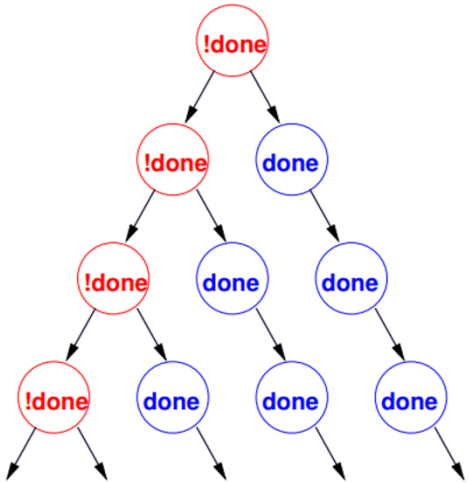
    $\bigcirc \rightsquigarrow X$ next time

    $\Diamond \rightsquigarrow F$ sometime in the future

CTL Semantics: Intuitions

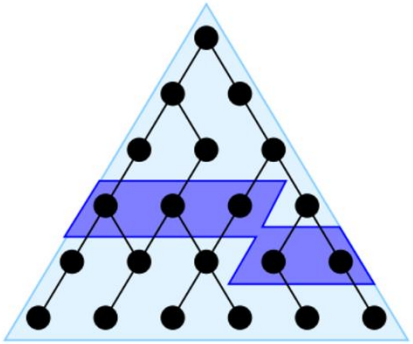
- **Globally:** $AG\phi$ **is true iff** ϕ **is always true in the future**
- **Necessarily Next:** $AX\phi$ **is true iff** ϕ **is true in every successor state**
- **Possibly Next:** $EX\phi$ **is true iff** ϕ **is true in some successor state**
- **Necessarily in the Future:** $AF\phi$ **is true iff** ϕ **is inevitably true in some future time**
- **Possibly in the Future:** $EF\phi$ **is true iff** ϕ **maybe true in some future time**





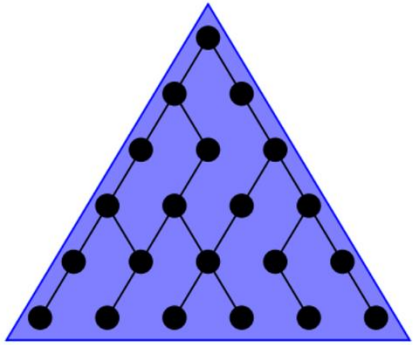
CTL Semantics: Intuitions

finally p



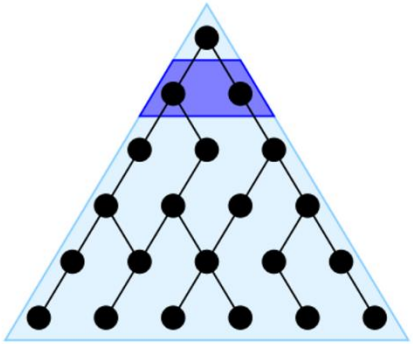
AF p

globally p



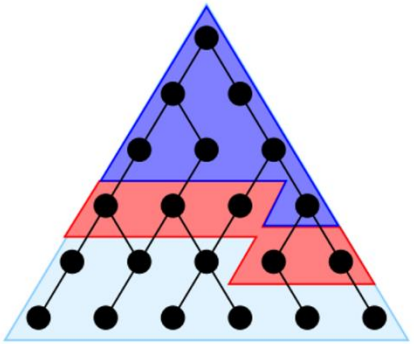
AG p

next p

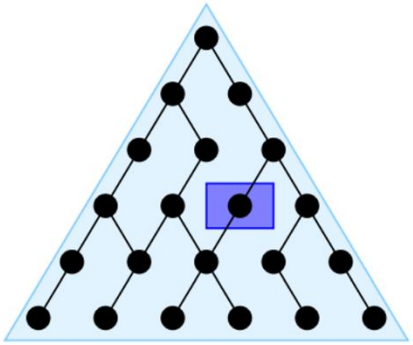


AX p

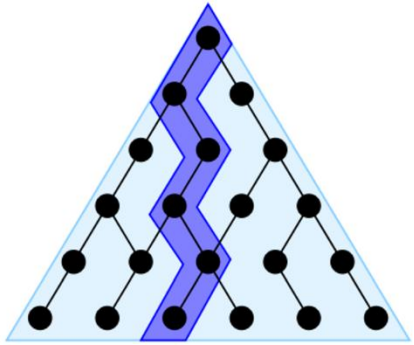
p until q



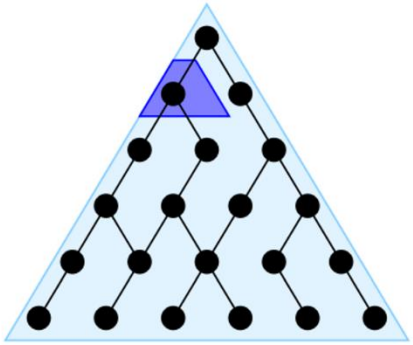
A [p U q]



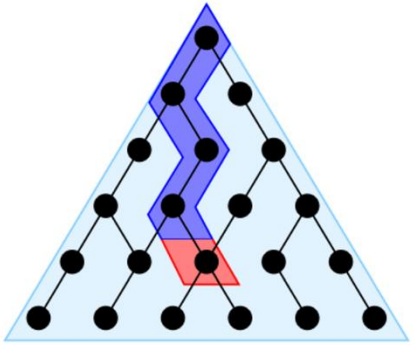
EF p



EG p



EX p



E [p U q]

Stage Summary

- LTL provides a user-friendly way for writing down LT properties
- LTL = Temporal operators + Boolean operators
- LTL formulas only evaluate infinite words
- scLTL can be satisfied in finite horizon
- LTL cannot capture branching-time properties; need CTL

Automata-Based Verification of LTL

Property Verification

- A property is a set of infinite words (language) $P \subseteq \left(2^{AP}\right)^{\omega}$
- For an LTL formula ϕ, we have $Word(\phi) = \{\sigma \in \left(2^{AP}\right)^{\omega} : \sigma \vDash \phi\}$
- To check\ whether or not $T \vDash \phi$, it suffice to check whether or not
  - $Trace(T) \subseteq Word(\phi)$; or
  - $Trace(T) \cap Word(\neg\phi) = \emptyset$
- How to efficiently represent $Word(\phi)$?
  - Need finite structure not to enumerate all strings (not possible)
  - Approach: using Automata to generate language

Finite Words & Regular Language

- **Alphabet** (event set) Σ, e.g., $\Sigma = \{a, b, c\}$
- **Finite word** (string): $w = w_1 w_2 \dots w_n$ where $w_i \in \Sigma$, e.g., $w = aabbc$
- **Kleene-closure**: Σ^* is the set of all finite strings over all including ϵ
- **Language**: a set of strings $L \subseteq \Sigma^*$, e.g., $L = \{\epsilon, a, ab, aa, aabc\}$

Regular Expression

- $\emptyset, \{\epsilon\}$ and $\{a\}, a \in \Sigma$ are regular languages
- If L_1 and L_2 are regular languages, the $L_1 \cup L_2$, $L_1 L_2$ and L_1^* are also
  - **Catenation**: $L_1 L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}$

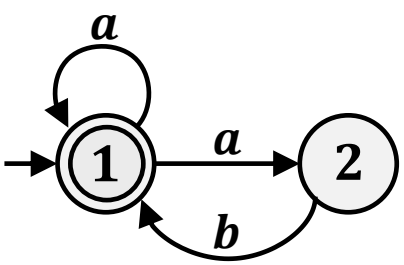
- $L = \{\epsilon, a\}\{a, b\}^* = \{\epsilon, a\}\{\epsilon, a, b, aa, ab, ba, bb, \dots\} = \{\epsilon, a, b, aa, ab, aaa, \dots\}$
- Remark: Σ can be 2^{AP} is previous examples!

Finite-State Automata

A Non-deterministic Finite-State Automata (NFA) is a tuple

$$A = (Q, Q_0, \delta, \Sigma, F)$$

- Q is a finite set of states
- $Q_0 \subseteq Q$ is the set of initial states
- Σ is the alphabet
- $\delta: Q \times \Sigma \to 2^Q$ is a partial transition function
- $F \subseteq Q$ is the set of accepting (final/marked) states.

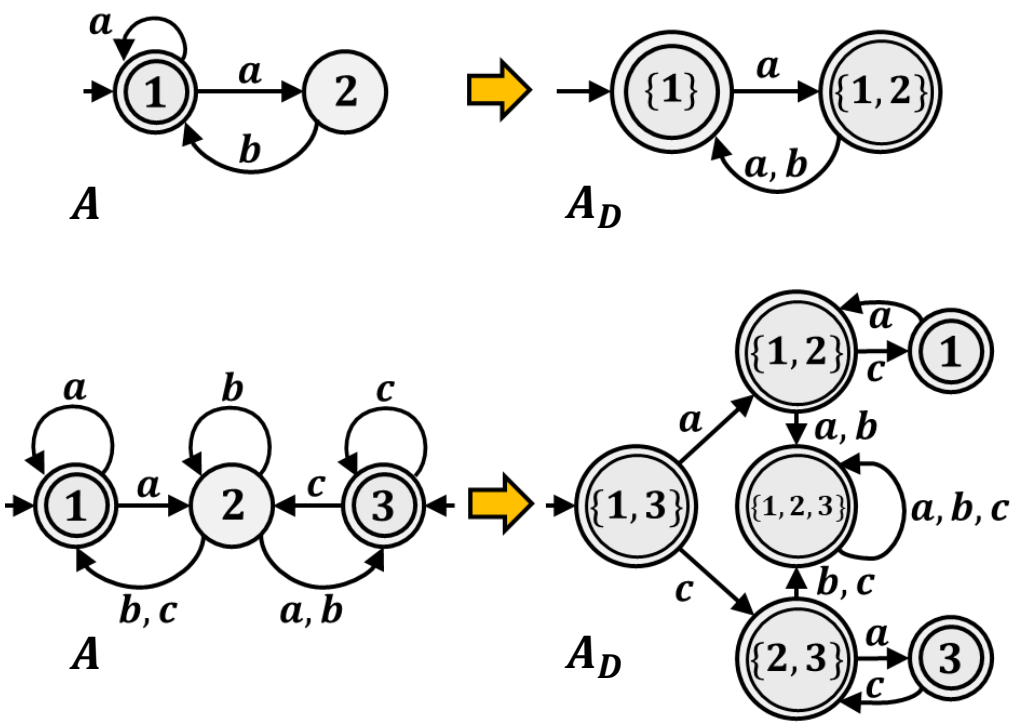


- $Q = \{1, 2\}, Q_0 = \{1\}, F = \{2\}, \Sigma = \{a, b\}, \delta(1, a) = \{1, 2\}$
- δ can be extended to $\delta: Q \times \Sigma^* \to 2^Q$, $\delta(1, aab) = \{1\}$
- Accepted Language: $\mathcal{L}(A) = \{s \in \Sigma^* : \exists q_0 \in Q_0, \delta(q_0, s) \cap F \neq \emptyset\}$
- $\mathcal{L}(A) = \{\epsilon, a, aa, ab, aab, aaba \ldots\}$

Theorem: A language is regular iff it can be accepted by a NFA.

NFA to DFA

- **Deterministic Finite-State Automata (DFA):** $|Q_0| = 1$ and $|\delta(q, \sigma)| = 1$
- **Accepted language can be simplified as** $\mathcal{L}(A) = \{s \in \Sigma^* : \delta(q_0, s) \in F\}$
- **Is NFA more powerful than DFA? No, they have the same power!**
- **Subset construction converts a NFA to a DFA with the same language**



Subset Construction

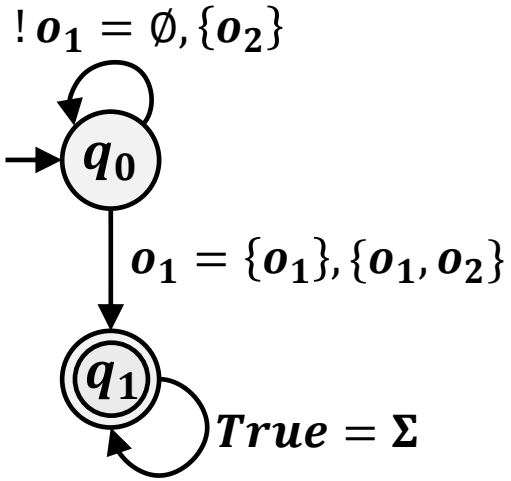
- start with Q_0
- for any $X \subseteq Q$ and $\sigma \in \Sigma$, compute

$$\delta_D(X, \sigma) = \cup_{q \in X} \delta(q, \sigma)$$

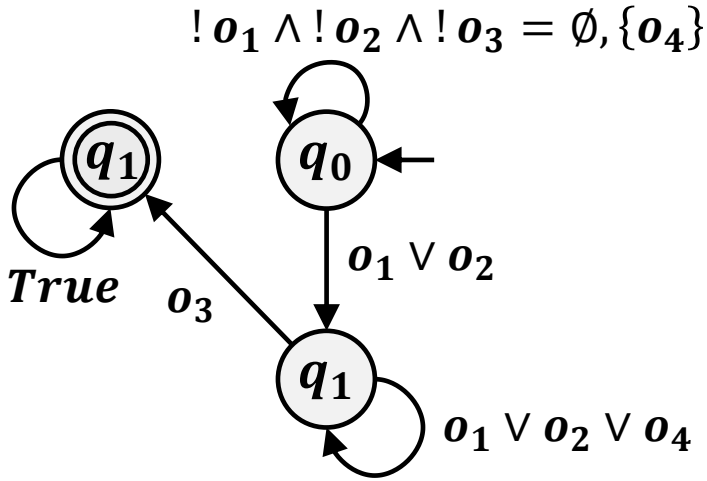
- mark X if it contains a state in F
- we have $\mathcal{L}(A_D) = \mathcal{L}(A)$
- A_D contains at most $2^{|Q|}$ states

From scLTL to DFA

For any scLTL formula ϕ over AP, there exists a DFA A_ϕ with alphabet $\Sigma = 2^{AP}$ that accepts all and only good prefixes of, i.e., $\mathcal{L}(A_\phi) = \mathcal{L}_{pref,\phi}$



- $AP = \{o_1, o_2\}$
- $\phi = \Diamond o_1$
- $\Sigma = \{\emptyset, \{o_1\}, \{o_2\}, \{o_1, o_2\}\}$



- $AP = \{o_1, o_2, o_3, o_4\}$
- $\phi = (\neg o_3 \; U \; (o_1 \vee o_2)) \wedge \Diamond o_3$

- A_ϕ **contains at most** $2^{|\phi|}$ **states**
- **Software tools: scheck2** https://github.com/jsjolen/scheck2

Infinite Words & ω-Regular Language

- A regular language is a set of finite words
- For an alphabet Σ, Σ^ω is the set of all infinite words over Σ
- For a regular language $L \subseteq \Sigma^*$, we define $L^\omega = \{w_1 w_2 \cdots : w_i \in L\}$
- Example: for $L = \{ab, c\}$, we have $L^\omega = \{ababab \cdots, ccc \ldots, abcabcabc \ldots\}$
- **ω-Regular Language**: $L_1(L_{1,inf})^\omega \cup L_2(L_{2,inf})^\omega \cup \cdots \cup L_n(L_{n,inf})^\omega$, where L_i and $L_{i,inf}$ are regular languages
- Safety: $(2^{AP})^\omega \setminus P_{safe} = BadPref(P_{safe})(2^{AP})^\omega$
- In fact, for any LTL formula ϕ, $Word(\phi)$ is ω-regular
- Question: how to generate a ω-regular language?

Non-deterministic Büchi Automata

A Non-deterministic Büchi Automata (NBA) is a tuple

$$A = (Q, Q_0, \delta, \Sigma, F)$$

- Q is a finite set of states
- $Q_0 \subseteq Q$ is the set of initial states
- Σ is the alphabet
- $\delta: Q \times \Sigma \to 2^Q$ is a partial transition function
- $F \subseteq Q$ is the set of accepting (final/marked) states.

- The structures of NBA and NFA are exactly the same
- The difference is how to interpret the accepting condition
- NBA is used to accept infinite words
- An infinite word is accepted if it visits accepting states infinitely many times

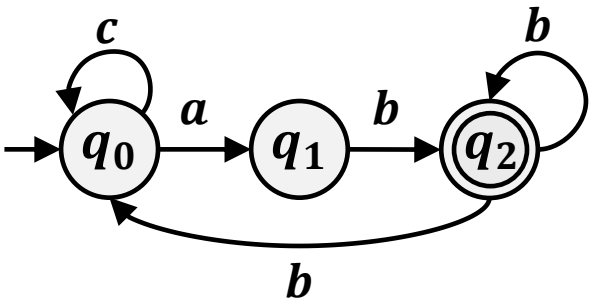
Non-deterministic Büchi Automata

- Given an infinite word $w = w_0 w_1 w_2 w_3 \cdots \in \Sigma^\omega$
- A run for w is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that

$$q_0 \in Q_0 \text{ and } \forall i \geq i: q_{i+1} \in \delta(q_i, w_i)$$

- A run $\rho = q_0 q_1 q_2 \ldots$ is said to be accepting if states in F occurs infinitely many times, i.e., $Inf(\rho) \cap F \neq \emptyset$
- Accepted language of NBA A is

$$\mathcal{L}^\omega(A) = \{w \in \Sigma^\omega : \text{there exists an accepting run for } w \text{ in } A\}$$



NBA A

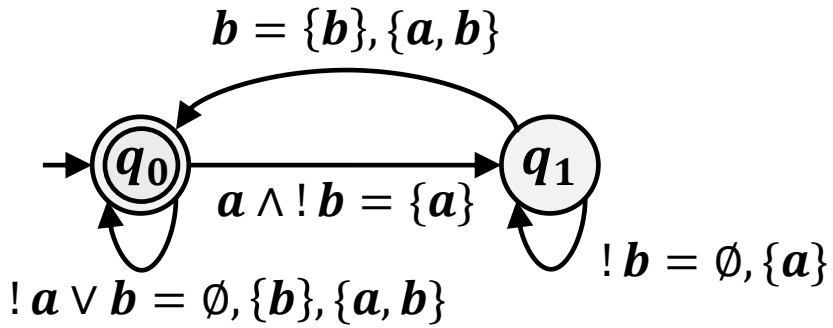
- word c^ω only has one run q_0^ω
- word ab^ω has accepting run $q_0 q_1 q_2^\omega$
- Word $(cabb)^\omega$ has accepting run $(q_1 q_1 q_2 q_3)^\omega$
- This NBA actually accepts ω-regular language

$$\{c\}^*\{ab\}(\{b\}^+ \cup \{b\}\{c\}^*\{ab\})^\omega$$

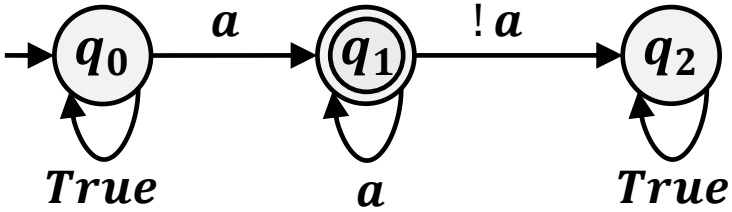
where $\{b\}^+ = \{b\}^* \setminus \{\epsilon\} = \{b, bb, bbb, \ldots\}$

From LTL to NBA

- A language is ω-regular iff it can be accepted by a NBA
- For any LTL formula ϕ over AP, there exists a NBA A_ϕ with alphabet $\Sigma = 2^{AP}$ such that $\mathcal{L}^\omega(A_\phi) = Word(\phi)$



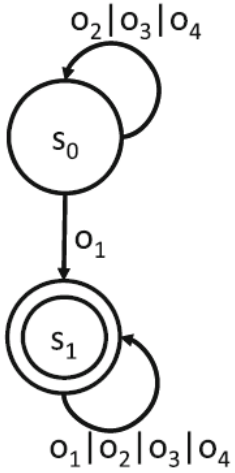
NBA for $\square(a \rightarrow \diamond b)$



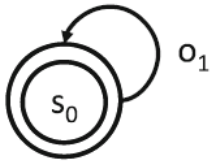
NBA for $\diamond\square a$

- A_ϕ contains at most $|\phi|2^{|\phi|}$ states
- Software tools: ltl2ba http://www.lsv.fr/~gastin/ltl2ba/

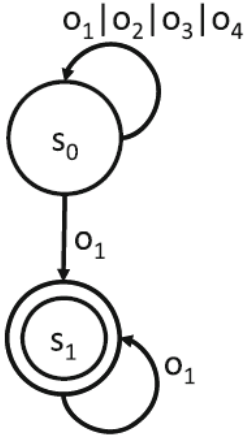
From LTL to NBA



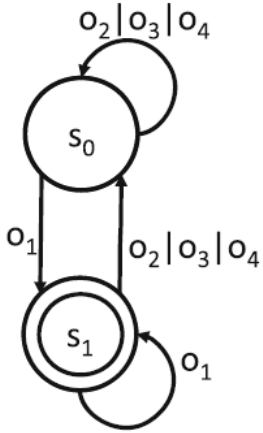
(a) $\phi_1 = \Diamond o_1$



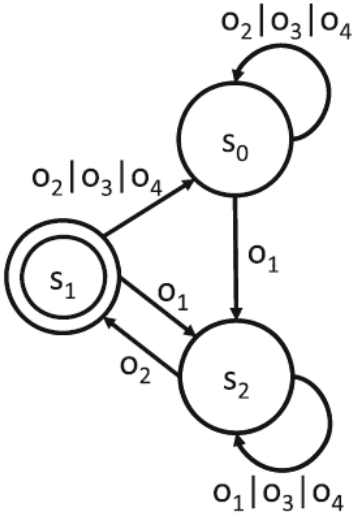
(b) $\phi_2 = \Box o_1$



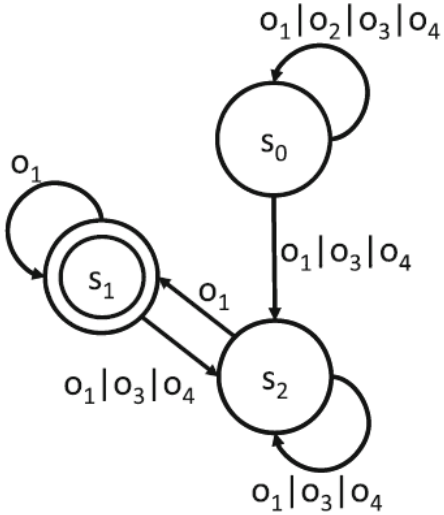
(c) $\phi_3 = \Diamond \Box o_1$



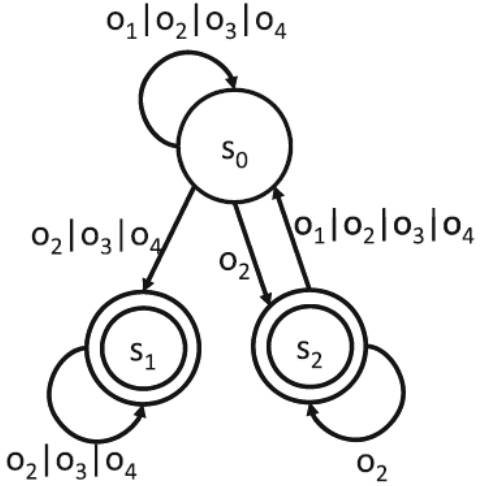
(d) $\phi_4 = \Box \Diamond o_1$



(e) $\phi_5 = \Box(\Diamond o_1 \wedge \Diamond o_2)$



(f) $\phi_6 = \Box \Diamond o_1 \wedge \neg \Box \Diamond o_2$



(g) $\phi_7 = \Box \Diamond o_1 \Rightarrow \Box \Diamond o_2$

Pictures from the book of Belta

Model Checking for Regular Safety

- P_{safe} is a safety property if each $\sigma \notin P_{safe}$ has a finite bad prefix
- P_{safe} is regular safety is its bad prefixes is a regular language
- Suppose that NFA A_{bad} accepts the bad prefixes, then

$$
\begin{aligned}
& T \not\models P \\
\text{if and only if} \quad & Trace(T) \not\subseteq P \\
\text{if and only if} \quad & Trace(T) \cap \left(\left(2^{AP}\right)^{\omega} \setminus P \right) \neq \emptyset \\
\text{if and only if} \quad & L(x_1)L(x_2) \ldots L(x_n) \in Trace(T) \cap P^c \\
\text{if and only if} \quad & Trace^f(T) \cap \mathcal{L}(A_{bad}) \neq \emptyset
\end{aligned}
$$

➢ The last condition can be checked by "synchronizing" T and A_{bad}!

Model Checking for Regular Safety

Let $T = (X, U, \rightarrow, X_0, AP, L)$ be a LTS and $A = (Q, Q_0, \delta, 2^{AP}, F)$ be an NFA. Then the product of T and A is a new tuple

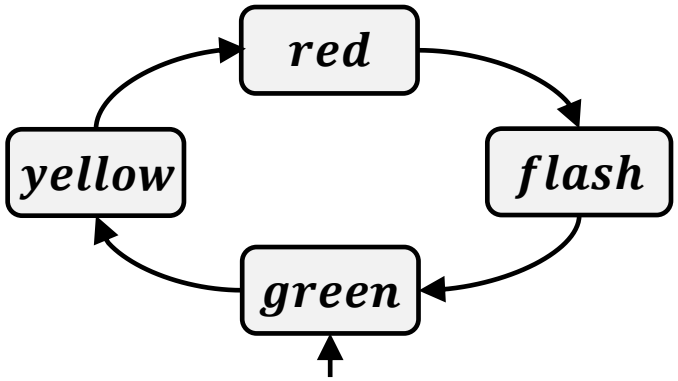
$$T \otimes A = (Q_\otimes, Q_{0\otimes}, \delta_\otimes, U, F_\otimes)$$

- $Q_\otimes = X \times Q$
- $Q_{0\otimes} = \{(x_0, q) : x_0 \in X_0 \wedge \exists q_0 \in Q_0 . q_0 \xrightarrow{L(x_0)} q\}$
- $F_\otimes = X \times F$
- $\delta_\otimes : Q_\otimes \times U \rightarrow 2^{Q_\otimes}$ is defined by:
  - $\delta_\otimes((x, q), u) = \{(x', q') \in Q_\otimes : x \xrightarrow{u} x' \text{ and } q \xrightarrow{L(x')} q'\}$

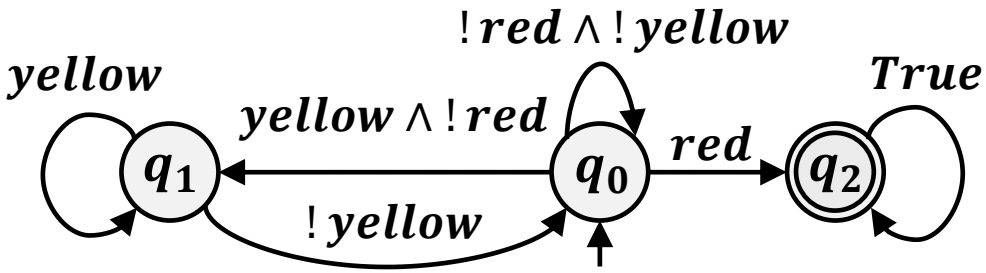
Key Observation:

- $T \otimes A$ accepts finite traces both generated by T and accepted by A
- $(x_0, q_1)(x_1, q_2) \cdots (x_n, q_{n+1}) \Rightarrow q_0 \xrightarrow{L(x_0)} q_1 \xrightarrow{L(x_1)} q_2 \xrightarrow{L(x_2)} \cdots \xrightarrow{L(x_n)} q_{n+1}$

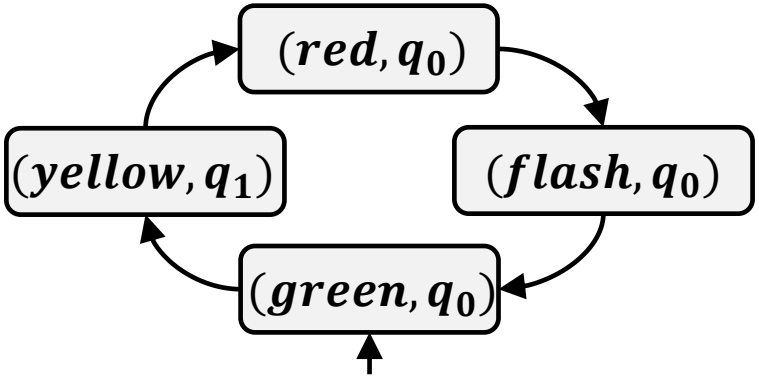
Example: LTS-NFA Product



LTS T for traffic light



DFA A_{bad} for "each red is preceded by yellow"



$T \otimes A$ as the product

Model Checking for Regular Safety

Given: T and NFA A_{bad}

- Build the product $T \otimes A$
- If $\mathcal{L}(T \otimes A) = \emptyset$, then return "safe"
- If $\mathcal{L}(T \otimes A) \neq \emptyset$, i.e., there is a reachable accepting state in $T \otimes A$, then return "not safe"

Model Checking for LTL

- Suppose we have an ω-regular property $P \subseteq \left(2^{AP}\right)^{\omega}$
- Now we have an LTS $T = (X, U, \rightarrow, X_0, AP, L)$ and we want to check whether or not $T \models P$
- Based the previous discussions, we have

$$
\begin{aligned}
& T \not\models P \\
\text{if and only if} \quad & Trace(T) \not\subseteq P \\
\text{if and only if} \quad & Trace(T) \cap \left(\left(2^{AP}\right)^{\omega} \setminus P\right) \neq \emptyset \\
\text{if and only if} \quad & Trace(T) \cap P^{c} \neq \emptyset \\
\text{if and only if} \quad & Trace(T) \cap \mathcal{L}^{\omega}(A^{c}) \neq \emptyset
\end{aligned}
$$

- $\left(2^{AP}\right)^{\omega} \setminus P$ is also ω-regular
- without loss of generality, we can assume $\mathcal{L}^{\omega}(A^{c}) = P^{c}$
- If $P = Word(\phi)$, then $P^{c} = Word(\neg\phi)$ and we can build $A_{\neg\phi}$!

Product between LTS and NBA

Let $T = (X, U, \rightarrow, X_0, AP, L)$ be a LTS and $A = (Q, Q_0, \delta, 2^{AP}, F)$ be an NBA.

Then the product of T and A is a new tuple

$$T \otimes A = (Q_\otimes, Q_{0\otimes}, \delta_\otimes, U, F_\otimes)$$

- $Q_\otimes = X \times Q$
- $Q_{0\otimes} = \{(x_0, q) : x_0 \in X_0 \wedge \exists q_0 \in Q_0 . q_0 \xrightarrow{L(x_0)} q\}$
- $F_\otimes = X \times F$
- $\delta_\otimes : Q_\otimes \times U \rightarrow 2^{Q_\otimes}$ is defined by:
  - $\delta_\otimes((x, q), u) = \{(x', q') \in Q_\otimes : x \xrightarrow{u} x' \text{ and } q \xrightarrow{L(x')} q'\}$

Exactly the same as the case of NFA!

LTL Model Checking Algorithm

- Suppose that we have an LTS T and an LTL formula ϕ
- We have $T \not\models \phi \Leftrightarrow Trace(T) \subseteq Word(\phi) \Leftrightarrow Trace(T) \cap Word(\neg\phi) \neq \emptyset$
- Let $A_{\neg\phi}$ be an NBA such that $\mathcal{L}^{\omega}(A_{\neg\phi}) = Word(\neg\phi)$. Then

$$Trace(T) \cap Word(\neg\phi) \neq \emptyset \Leftrightarrow \mathcal{L}^{\omega}(T \otimes A_{\neg\phi}) \neq \emptyset$$

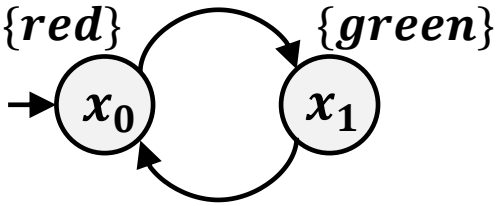
- The above is equivalent to the existences an accepting state in $T \otimes A_{\neg\phi}$ that can be reached infinitely often, i.e., in a cycle!

Model Checking for LTL

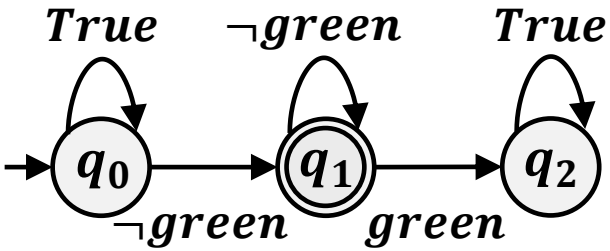
Given: T and LTL Formula ϕ

- Build the NBA $A_{\neg\phi}$ that accepts $Word(\neg\phi)$
- Build the product $T \otimes A_{\neg\phi}$
- Find all strongly connected components (SCC) of $T \otimes A_{\neg\phi}$
- Check if there exists a SCC that contains a state in $F_{\otimes}$ and at least a transition
- If so, return "$T \not\models \phi$"; otherwise, return "$T \models \phi$"

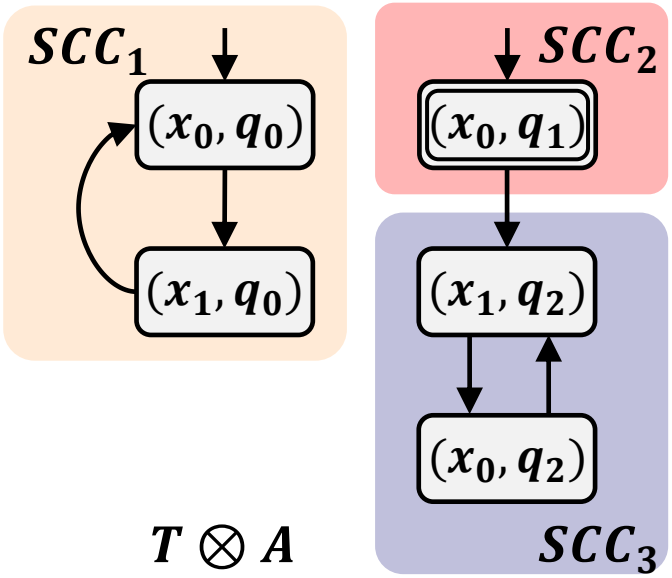
Example: LTL Model Checking



LTS T for traffic light



NBA A for $\neg(\square\diamond green) = \diamond\square\neg green$

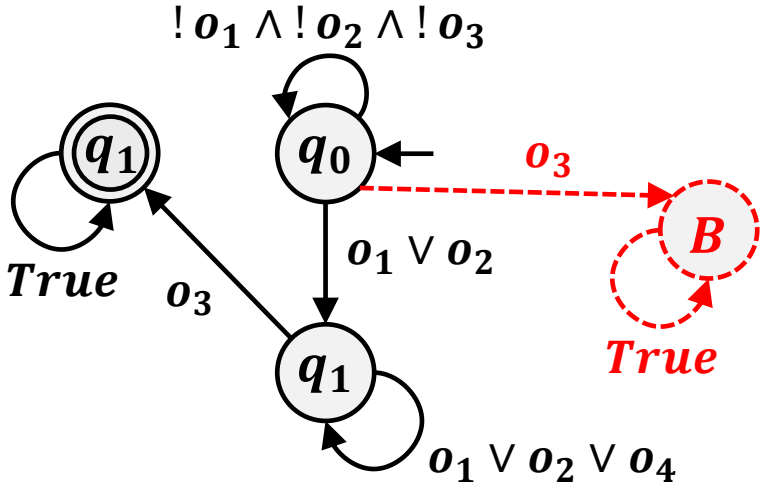


$T \otimes A$

- There are three SCCs in $T \otimes A$
- SCC_2 contains $F_\otimes$ but does not have transition
- SCC_1, SCC_3 have transitions but have no $F_\otimes$
- No infinite accepting word can be generated
- Therefore, $T \vDash \square\diamond green$

Case of scLTL

- For any scLTL formula ϕ, there is a DFA $A = (Q, q_0, \delta, \Sigma, F)$ that accepts all good prefixes, i.e., $\mathcal{L}(A) = \mathcal{L}_{pref,\phi}$
- Non-satisfaction means
  - Never reach an accepting state, i.e., loop in non-accepting; or
  - Outside of the transitions of A
- Build A_{com} to "complete" the transition and compute $T \otimes A_{com}$
- If $T \otimes A_{com}$ contains a cycle in which there is no accepting state, then $T \not\models \phi$



- $AP = \{o_1, o_2, o_3, o_4\}$
- $\phi = (\neg o_3 \; U \; (o_1 \vee o_2)) \wedge \Diamond o_3$

Build $A_{com} = (Q^c, q_0, \delta^c, \Sigma, F)$

- $Q^c = Q \cup \{Bad\}$
- If $\delta(q,w)!$, then $\delta^c(q,w) = \delta(q,w)$
- If $\delta(q,w)\neg!$, then $\delta^c(q,w) = Bad$

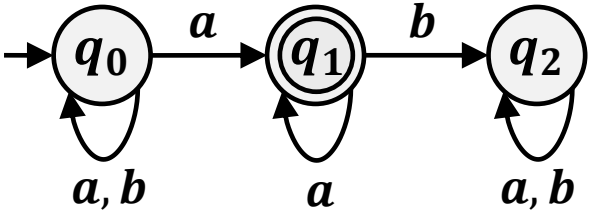
Discussions

- NFA accepts finite words, i.e., generates regular language
- NFA can DFA are equivalent according to the subset construction
- NBA accepts infinite words, i.e., generates ω-regular language
- Regular safety is essentially non-reachability
- Co-Safe LTL is essentially safety + finite reachability
- Safety can be converted to non-reachability by adding state Bad
- General LTL is essentially persistence
- Are NBA and DBA equivalent? No!

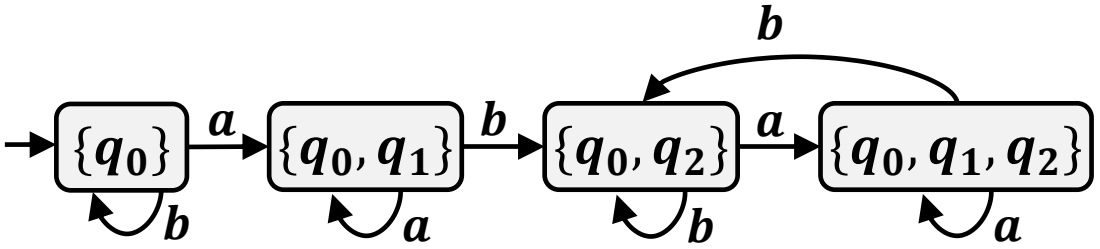
NBA v.s. DBA

NBA is strictly more powerful than DBA, i.e., there exists ω-regular language that cannot be accepted by a DBA.

- **Deterministic Büchi Automata (DBA): $|Q_0| = 1$ and $|\delta(q, \sigma)| = 1$**
- **"eventually for every" cannot be captured by DBA**
- **Consider ω-regular language $L = \{a, b\}^*\{a\}^\omega$**
- **We need to nondeterministically decide from which instant the proposition a is continuously true**



NBA for $L = \{a, b\}^*\{a\}^\omega$



- **Automata obtained by the subset constriction**
- **Defining accepting states is problematic!**

Rabin Automata

- In many problems we do need deterministic mechanism, but the expressiveness of DFA is limited
- Using different accepting condition: **Rabin acceptance**

A **Deterministic Rabin Automata (DRA)** is a tuple

$$A = (Q, q_0, \delta, \Sigma, Acc)$$

- Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is the alphabet
- $\delta: Q \times \Sigma \to Q$ is a partial deterministic transition function
- $Acc = \{(L_1, K_1), \ldots, (L_n, K_n)\} \subseteq 2^Q \times 2^Q$ is the acceptance condition.

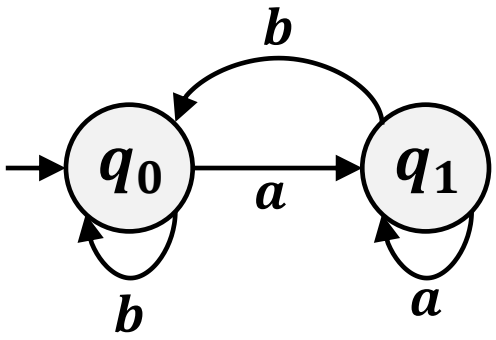
- A run $\rho = q_0 q_1 q_2 \ldots$ is **accepting if there exists a pair** $(L, K) \in Acc$ **s.t.**

$$[Inf(\rho) \cap L = \emptyset] \wedge [Inf(\rho) \cap K \neq \emptyset]$$

- Accepted language of DBA A is

$$\mathcal{L}^\omega(A) = \{w \in \Sigma^\omega : \text{the run induced by } w \text{ is accepting in } A\}$$

Rabin Automata

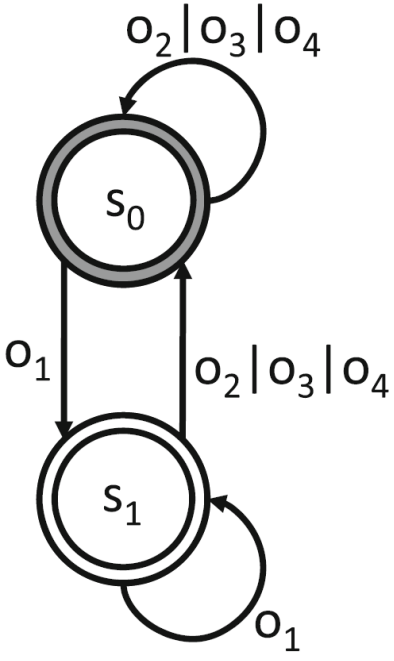


- DRA for "◇□a"
- $Acc = \{(\{q_0\}, \{q_1\})\}$
- Looping between q_0 and q_1 is rejected

- The class of languages accepted by DRA is the same as that of NBA
- For any LTL formula ϕ over AP, there exists a DRA A_ϕ with alphabet $\Sigma = 2^{AP}$ such that $\mathcal{L}^\omega(A_\phi) = Word(\phi)$
- Cost: we may need $2^{2^{|\phi| \cdot \log|\phi|}}$ states and $2^{|\phi|}$ pairs
- Tools: ltl2dstar https://www.ltl2dstar.de/

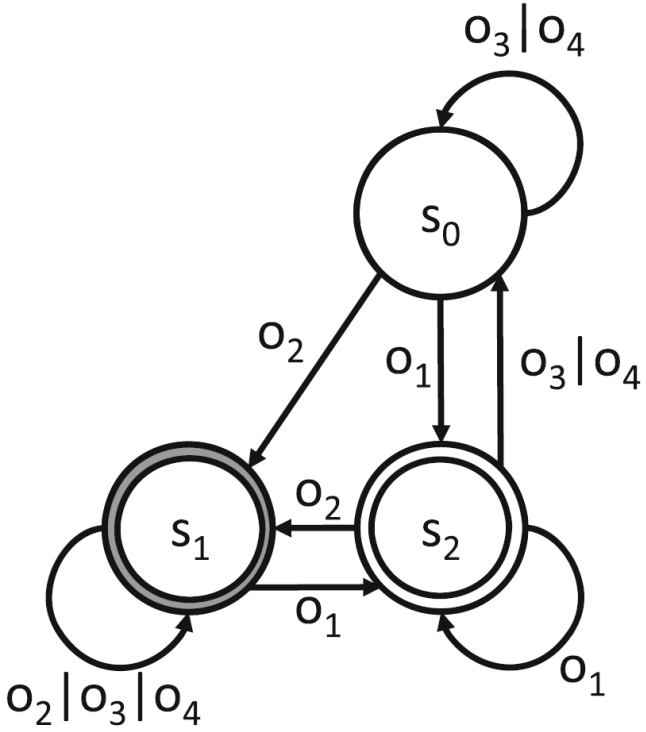
Rabin Automata: More Examples

$Acc = \{(\{s_0\}, \{s_1\})\}$



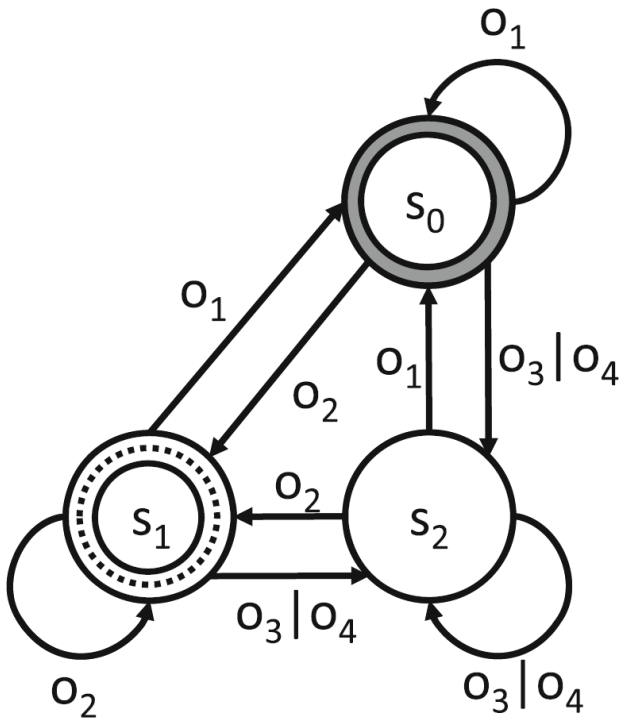
(a) $\phi_3 = \Diamond\Box o_1$

$Acc = \{(\{s_1\}, \{s_2\})\}$



(b) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$

$Acc = \{(\{s_0\}, \{s_1, s_2\}), (\emptyset, \{s_1\})\}$



(c) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Pictures from the book of Belta

Another Definition of Product

Alternative Definition

Let $T = (X, U, \rightarrow, X_0, AP, L)$ be a LTS and $A = (Q, Q_0, \delta, 2^{AP}, F)$ be an NBA.

Then the product of T and A is a new tuple

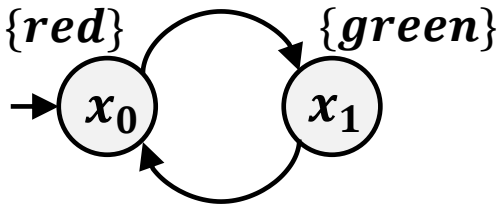
$$T \otimes A = (Q_\otimes, Q_{0\otimes}, \delta_\otimes, U, F_\otimes)$$

- $Q_\otimes = X \times Q$, $Q_{0\otimes} = X_0 \times Q_0$, $F_\otimes = X \times F$
- $\delta_\otimes : Q_\otimes \times U \rightarrow 2^{Q_\otimes}$ is defined by:
  - $\delta_\otimes((x,q),u) = \{(x',q') \in Q_\otimes : x \xrightarrow{u} x' \text{ and } q \xrightarrow{L(x)} q'\}$

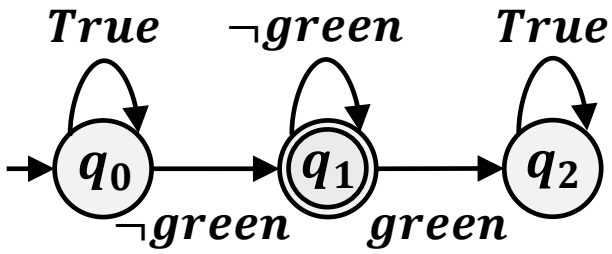
Previous Definition

- $Q_{0\otimes} = \{(x_0, q) : x_0 \in X_0 \wedge \exists q_0 \in Q_0 . q_0 \xrightarrow{L(x_0)} q\}$
- $\delta_\otimes : Q_\otimes \times U \rightarrow 2^{Q_\otimes}$ is defined by:
  - $\delta_\otimes((x,q),u) = \{(x',q') \in Q_\otimes : x \xrightarrow{u} x' \text{ and } q \xrightarrow{L(x')} q'\}$

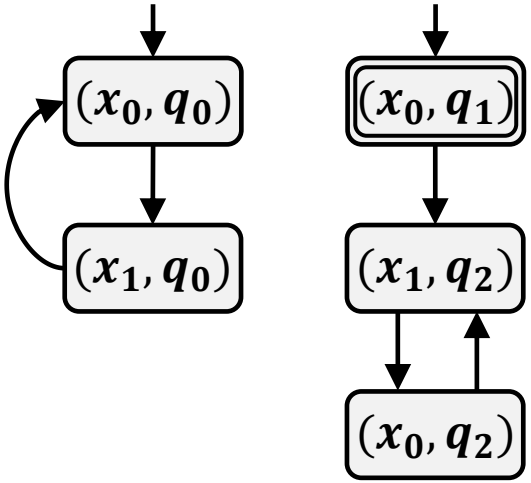
Example: LTS-NBA Product



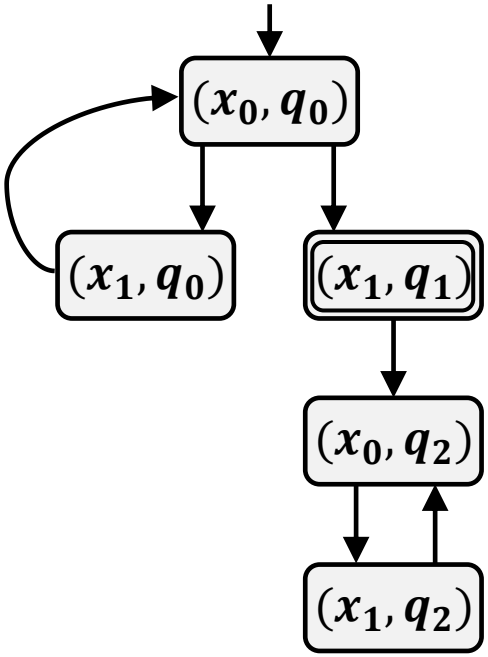
LTS T for traffic light



NBA A for $\neg(\square\diamond green) = \diamond\square\neg green$

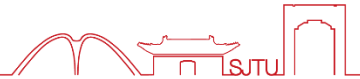


$T \otimes A$ by Definition 1



$T \otimes A$ by Definition 2

Stage Summary

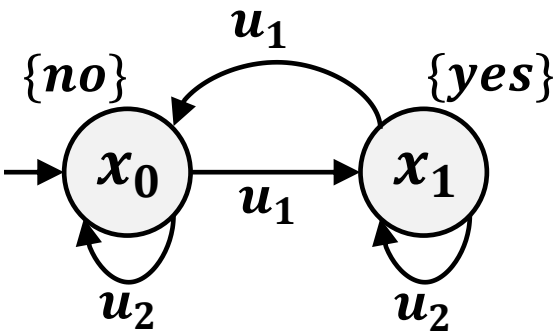


- Any regular language can be accepted by an NFA
- Any ω-regular language can be accepted by an NBA
- NFA and DFA are equivalent but NBA and DBA are not equivalent
- Any LTL formula can be translated to an NBA (DBA is not enough)
- Any LTL formula can be translated to DRA
- Büchi is smaller but need to pay nondeterminism
- Rabin can resolve nondeterminism but need to pay larger state-space
- Good prefixes of scLTL can be accepted by DFA
- Model checking by synchronizing the LTS and the automata for LTL

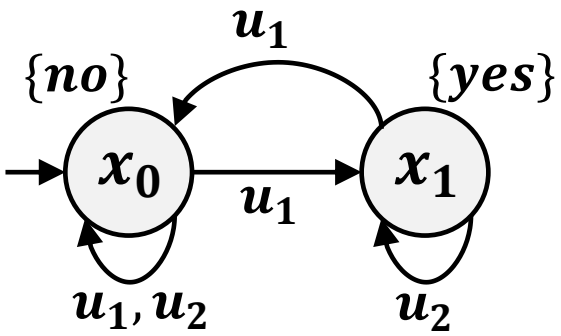
Game-Based LTL Control Synthesis

Role of Inputs

- Control input is not important in the verification problem since we want to check the satisfaction for all runs
- Some "bad" runs can be avoided by suitably choosing inputs
- $T_1 \not\models \Diamond\Box yes$, e.g., $(x_0x_1)^\omega$ yields $(\{no\}\{yes\})^\omega$
- We can choose input sequence $u_1(u_2)^\omega$, which gives $\{no\}(\{yes\})^\omega$
- In general, the effect of an input is non-deterministic $|Post(x,u)| > 1$
- We also need to handle all possible consequences of the input



Deterministic LTS T_1 with $U = \{u_1, u_2\}$



Non-Deterministic LTS T_2

Synthesis Problem

- A control strategy is a function $C: X(X)^* \to U$
- State run under C: $x_0 x_1 \cdots x_n \cdots$ such that $x_{i+1} \in Post\big(x_i, C(x_0 \cdots x_n)\big)$
- The set of all infinite runs of T under C: $Run(C/T)$
- The set of all infinite traces of T under C: $Trace(C/T)$

Control Synthesis Problem

Given an LTS T and a property $P \subseteq \left(2^{AP}\right)^{\omega}$, find a control strategy $C: X(X)^* \to U$ such that $C/T \vDash P$, i.e., $Trace(C/P) \subseteq P$.

- for LTL formula ϕ, $C/T \vDash \phi$ means $Trace(C/P) \subseteq Word(\phi)$

Case of Deterministic System

- A control strategy is not a single input sequence
- A controller should be reactive to non-determinism
- A singe (infinite) input sequence is enough when it is deterministic
- Deterministic synthesis problem (path planning problem) can be solved by model checking (returns a counter-example if negative)

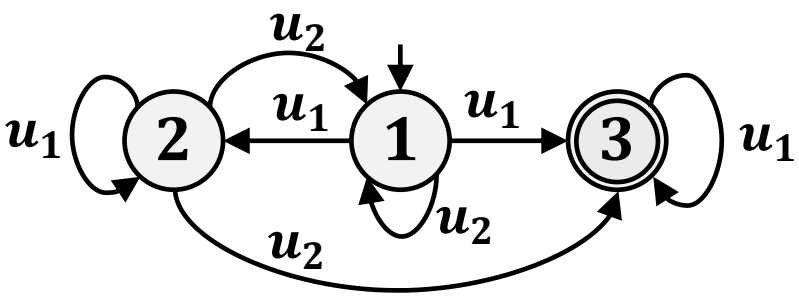
LTL Path Planning by Model Checking

Given: T and LTL Formula ϕ

- Use model checker to verify whether or not $T \models \neg\phi$
- If $\textsc{Check}(T, \neg\phi) = \text{"Yes"}$, then return "no controller exists"
- If $\textsc{Check}(T, \neg\phi) = \text{"No"}$, then the model-checker will provide an infinite run $\rho \in X^{\omega}$ as counter-example, i.e., $L(\rho) \not\models \neg\phi$. Return "$\rho$" as the planned path

General Case as a Two-Player Game

- **Player-C: controller** chooses an input $u \in U$ that is defined at $x \in X$
- **Player-A:** adversary chooses a successor $x' \in Post(x, u)$ and the system moves to x'; then Player-C chooses and so forth…
- The strategy of Player-C is actually a controller $C: X(X)^* \to U$
- The strategy of Player-A is a function $A: X(X)^* U \to X$
- If C and A are given, the initial-state $x_0 \in X_0$ is given, then the run is uniquely determined, denoted by $\rho(A, C, x_0)$ and denote the trace by $\sigma(A, C, x_0)$
- Then $C/T \vDash P$ iff $\forall A, \forall x_0 \in X_0: \sigma(A, C, x_0) \in P$



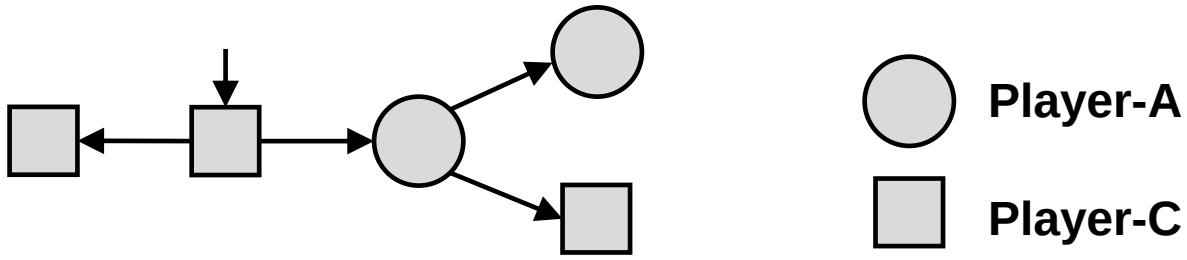
- $C/T \vDash \square\neg 3$ can be achieved by fixing u_2
- $C/T \vDash \Diamond\neg 3$ cannot be achieved by any C

control with non-determinism as a two-player game

Two-Player Games

- **Safety Game**: stay within safe states (not to reach unsafe states)
- **Reachability Game**: reach desired states within finite number of steps
- **Büchi Game**: visit desired states infinitely often
- **Rabin Game**: visit desired states infinitely often avoiding rejected states

- Safety game is related to regular safety
- Reachability game is related to scLTL
- Büchi game and Rabin game are related to general LTL
- Two-player game can also be formulated by explicitly partitioning the state-space for each player



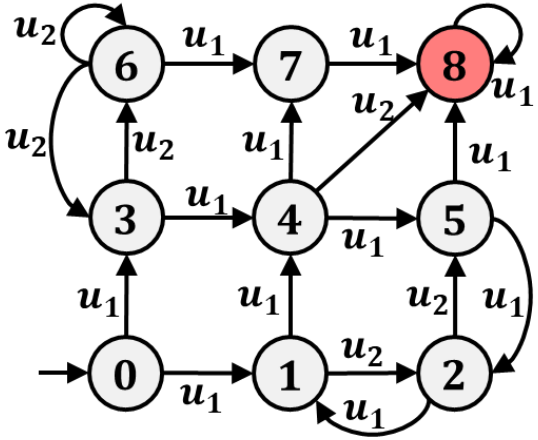
A different formulation of two-player game

Safety Game

Safety Game (Reach Avoid Game)

For LTS T and a set of unsafe region $B \subseteq X$, find a controller C such that the run never reaches B under any possible adversary A.

- **Winning Region:** the set of states from which Player-C can win
- Player-C wins (exists a controller) if $X_0 \subseteq X_{win}$
- By definition, control strategy is history based, but state-based (memoryless) strategy is sufficient for many games



Can you avoid state 8?

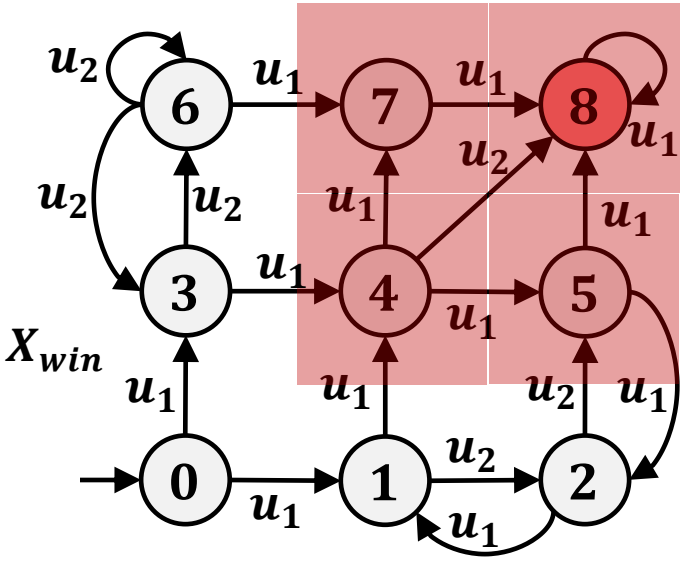
Solving Safety Game

- We need to avoid unsafe states $B \subseteq X$
- To avoid B, we need to avoid states that cannot avoid B

$$B_1 = \mathbf{Avoid}(B) = \{x \in X : \forall u \in U, Post(x,u) \cap B \neq \emptyset\}$$

- Then we also need to avoid $B_2 = \mathbf{Avoid}(B_1)$
- Keep deleting states until we get the winning region $X_{win} \subseteq X$ s.t.

$$X_{win} \cap B = \emptyset \quad \text{and} \quad \forall x \in X_{win}, \exists u \in U : Post(x,u) \subseteq X_{win}$$



Safety Game Algorithm

- Delete B from T, $B \leftarrow B \cup \mathbf{Avoid}(B)$
- Repeat the above until $B = B \cup Avoid(B)$
- State remained are X_{win}
- If $X_0 \not\subseteq X_{win}$, then "no controller"
- Otherwise, C choose an input $u \in U$ at each $x \in X$ s.t. $Post(x,u) \subseteq X_{win}$

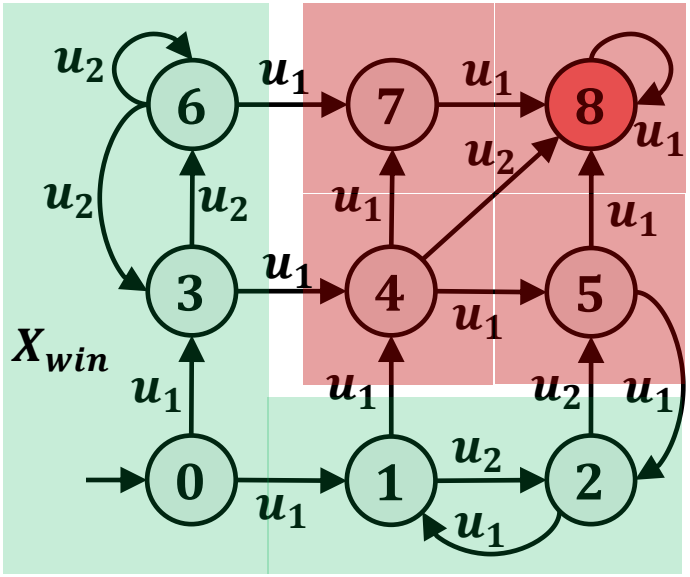
Solving Safety Game

- We need to avoid unsafe states $B \subseteq X$
- To avoid B, we need to avoid states that cannot avoid B

$$B_1 = \mathbf{Avoid}(B) = \{x \in X : \forall u \in U, Post(x,u) \cap B \neq \emptyset\}$$

- Then we also need to avoid $B_2 = \mathbf{Avoid}(B_1)$
- Keep deleting states until we get the winning region $X_{win} \subseteq X$ s.t.

$$X_{win} \cap B = \emptyset \quad \text{and} \quad \forall x \in X_{win}, \exists u \in U : Post(x,u) \subseteq X_{win}$$



Safety Game Algorithm

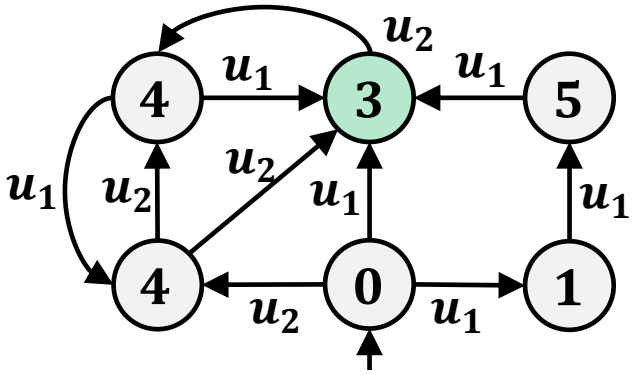
- Delete B from T, $B \leftarrow B \cup \mathbf{Avoid}(B)$
- Repeat the above until $B = B \cup Avoid(B)$
- State remained are X_{win}
- If $X_0 \not\subseteq X_{win}$, then "no controller"
- Otherwise, C choose an input $u \in U$ at each $x \in X$ s.t. $Post(x,u) \subseteq X_{win}$

Reachability Game

Reachability Game

For LTS T and a set of desired region $D \subseteq X$, find a controller C such that the run can always reaches D within finite steps under any possible A.

- Player-C losses the game iff the adversary can let the system loop in a cycle in which there is no desired state
- If Player-C wins the game, then it can always reaches D within $|X|$ steps



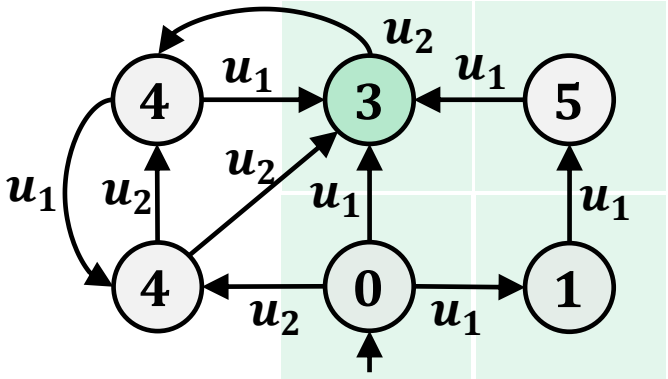
Can you reach state 3?

Solving Reachability Game

- To guarantee reach $D \subseteq X$ in one step, we must in states $D_1 = D \cup \mathbf{CPre}(D)$

$$\mathbf{CPre}(D) = \{x \in X : \exists u \in U, Post(x,u) \subseteq D\}$$

- To guarantee reach D in two steps, we must in states $D_2 = D_1 \cup \mathbf{CPre}(D_1)$
- By keep expending the region of attraction, we get the winning region $X_{win} = \mathbf{Attr}(D) := D \cup D_1 \cup \cdots \cup D_n = D_n$. For each D_{i+1} we can always move to D_i to be "closer" to the target region



- $D_0 = \{3\}, D_1 = \{3,5\}, D_3 = \{1,3,5\}$
- $D_4 = X_{win} = \{0,1,3,5\}$
- $C(0) = C(1) = C(5) = u_1, C(3) = u_2$

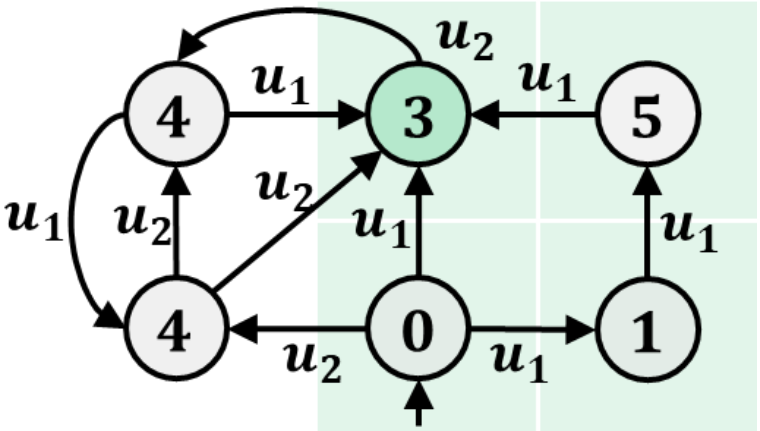
Reachability Game Algorithm

- Define $D_0 = D$
- Repeat $D_{i+1} = D_i \cup \mathbf{CPre}(D_i)$ until $D_i = \mathbf{CPre}(D_i)$
- If $X_0 \not\subseteq X_{win} = D_n$, then "no controller"
- Otherwise, C choose an input $u \in U$ at each $x \in D_i$ s.t. $Post(x,u) \subseteq D_0 \cup \cdots \cup D_{i-1}$

Büchi Game

Büchi Game

For LTS T and a set of accepting states $F \subseteq X$, find a controller C such that the run can always visits F infinitely often under any possible A.



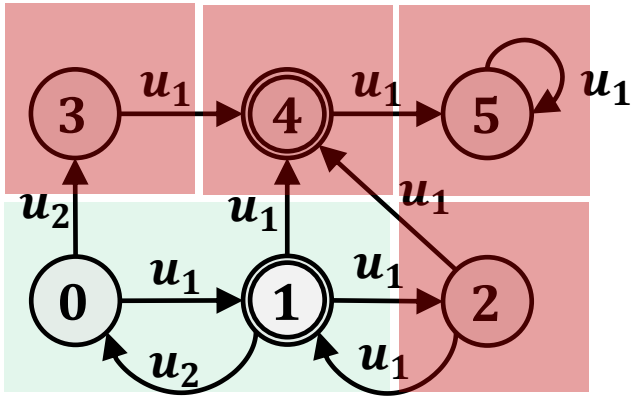
- Player-C wins the reachability game but cannot win the Büchi game for $F = \{3\}$
- State 3 can only be guaranteed to be visited once
- We should also take care of recurrence for what happens after reaching F

Solving Büchi Game

- To visit F again, we must in $\mathbf{Attr}(F)$
- We need to avoid $W_A = X \setminus \mathbf{Attr}(F)$ from F
- Therefore, we shrink accepting states to $F = F \setminus \mathbf{APre}(W_A)$, where

$$\mathbf{APre}(W_A) = \{x : \forall u \in U, Post(x,u) \cap W_A \neq \emptyset\}$$

- Since F is changed, we need to computed $\mathbf{Attr}(F)$ and $\mathbf{APre}(W_A)$ again



- $F = \{1,4\}, W_A = \{5\}, \mathbf{APre}(W_A) = \{4,5\}$
- $F = \{1\}, \mathbf{Attr}(F) = \{0,1\}, W_A = \{2,3,4,5\}$, $\mathbf{APre}(W_A) = \{3,4,5\}$
- $F \setminus \{3,4,5\} = \{1\}$

Büchi Game Algorithm

- $F = F \setminus \mathbf{APre}\big(X \setminus \mathbf{Attr}(F)\big)$
- Repeat above until $\mathbf{APre}\big(X \setminus \mathbf{Attr}(F)\big) \cap F = \emptyset$
- If $X_0 \not\subseteq X_{win} = Attr(F)$, then "no controller"
- Otherwise, C choose an input $u \in U$ based on the reachability game for F

Rabin Game

Rabin Game

For LTS T and a set of accepting pairs $Acc = \{(L_1, K_1), \dots, (L_n, K_n)\} \subseteq 2^X \times 2^X$, find a controller C such that for any adversary A there exists a pair (L_i, K_i) such that the run visits K_i infinite times and L_i only finite times.

General Idea:

- For each pair (L_i, K_i), consider a **Büchi Game for K_i + Safety Game for L_i**
- Then we get $K_i' \subseteq K_i$ that can be visited infinitely often without visiting L_i
- Then we consider a **reachability game for** $\cup_{i=1,\dots,n} K_i'$
- The winning region is actually $\mathbf{Attr}(\cup_{i=1,\dots,n} K_i')$

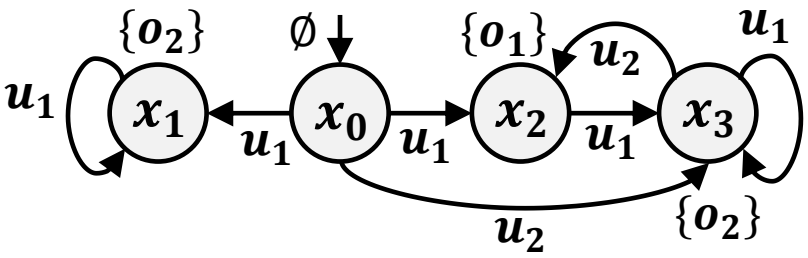
LTL Synthesis: General Case

- Suppose we have an LTS T and an LTL formula ϕ
- We want to find a C such that $C/T \vDash \phi$
- We first build DRA A_ϕ such that $\mathcal{L}^\omega(A_\phi) = Word(\phi)$
- Then we build $T \otimes A_\phi$ with $Acc_\otimes = \{(X \times L_1, X \times K_1), \dots, (X \times L_n, X \times K_n)\}$
- Solve the Rabin game for $T \otimes A_\phi$ with $Acc_\otimes$
- The winning strategy in $T \otimes A_\phi$ can be mapped directly to T by looking at the first component

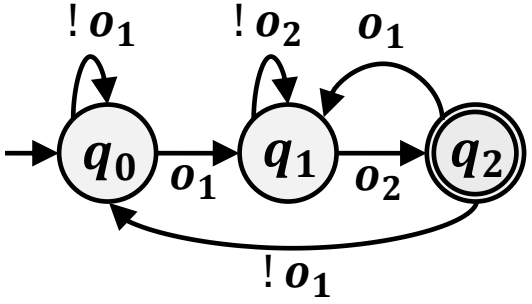
➢ Note: we cannot use NBA for ϕ because the LTS T may be non-deterministic; otherwise, we cannot really control the system

LTL Synthesis: Deterministic Case

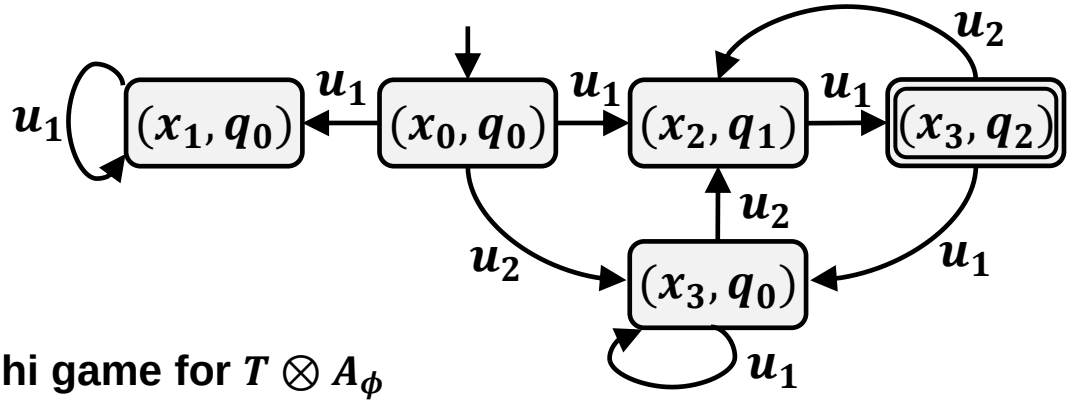
- In case the LTL formula ϕ can be accepted by a DBA A_ϕ (dLTL)
- We can just build $T \otimes A_\phi$ and solve the Büchi game



LTS T



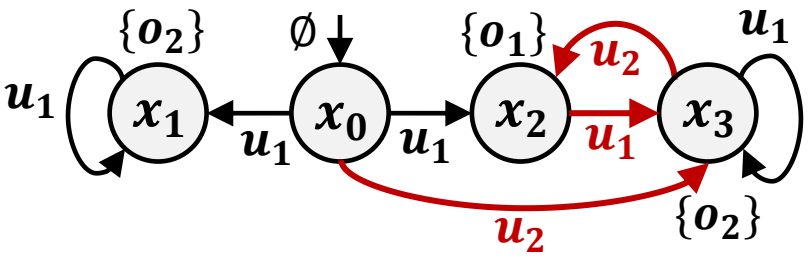
DBA for dLTL $\phi = \square(\diamond o_1 \wedge \diamond o_2)$



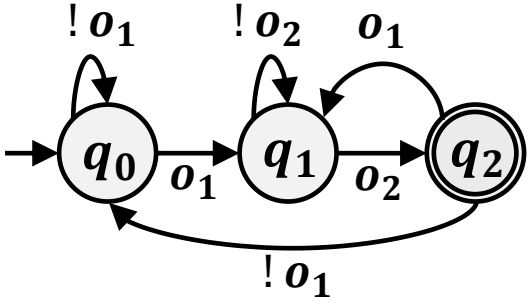
Büchi game for $T \otimes A_\phi$

LTL Synthesis: Deterministic Case

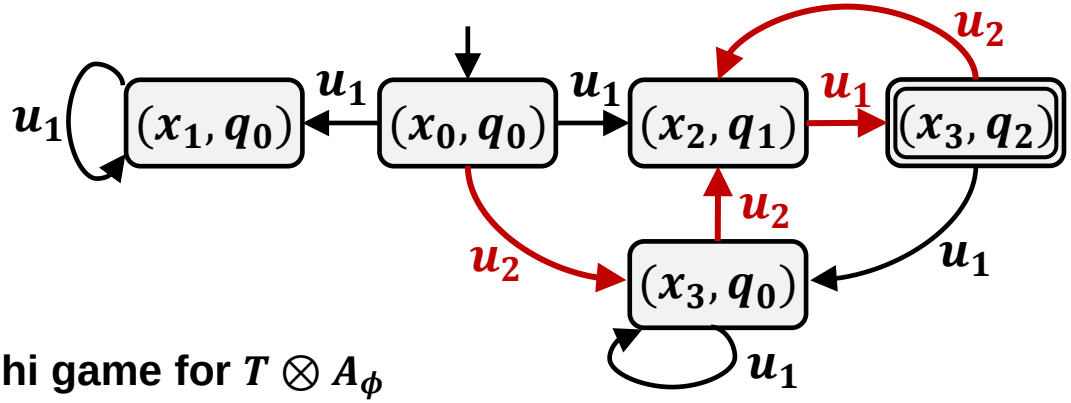
- In case the LTL formula ϕ can be accepted by a DBA A_ϕ (dLTL)
- We can just build $T \otimes A_\phi$ and solve the Büchi game



LTS T



DBA for dLTL $\phi = \square\,(\diamond o_1 \wedge \diamond o_2)$



Büchi game for $T \otimes A_\phi$

Stage Summary

- Control problem can be viewed as a two-player game
- Safety game can be solved by inductively extending the unsafe region
- Reachability game can be solved by using n-step attractor
- Büchi game can be solved by identify recurrent accepting states
- Rabin game can be solved by combing safety, reachability and Büchi
- LTL control synthesis can be solved as a game over the product
- General LTL needs to solve Rabin game
- dLTL can be solved by Büchi game
- scLTL can be solved by reachability game

Course Summary

- 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
- 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

Advanced Topics

- Timed & hybrid dynamic systems
- Formal abstraction of continuous dynamic systems
- Stochastic systems and probabilistic verification/synthesis
- Real-valued & real-time logics, e.g., MTL and STL
- Information-flow analysis or hyper-properties
- Control synthesis under imperfect information
- Verification & synthesis for multi-agent systems
- Temporal logics guided learning

Thank You!

yinxiang@sjtu.edu.cn
http://xiangyin.sjtu.edu.cn