Formal Verification of Digital Systems
Part I: Language Containment

Dr. Gitanjali Swamy

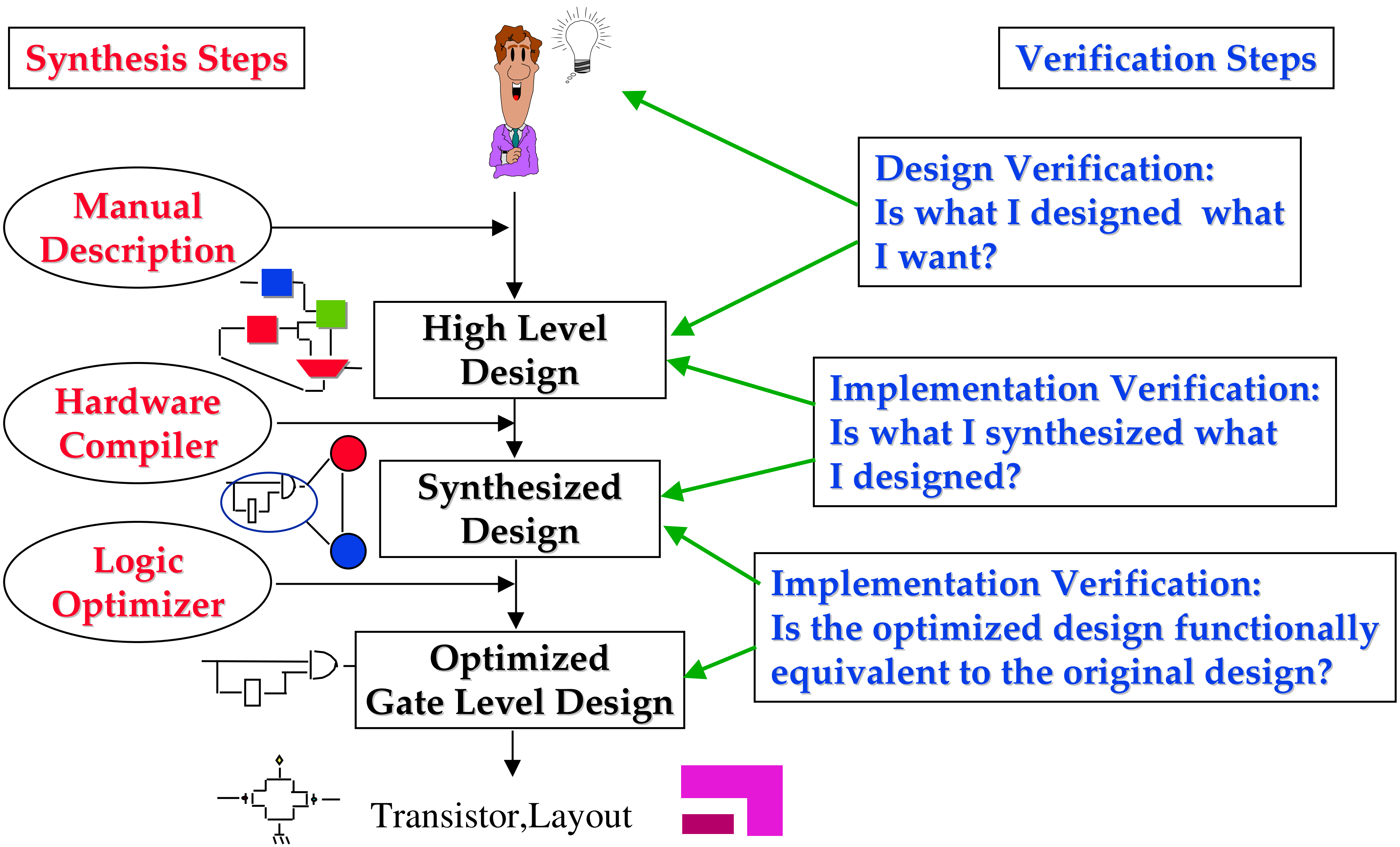
Boston Advanced Development Labs

Mentor Graphics Corporation

Mentor Graphics

The Power to Create

Synthesis and Verification In Top Down Design Methodology



Formal Design Verification: Motivation

- Design verification:
  Is what I designed what I wanted ?
- Traditional solution:
  Simulation.

Formal Design Verification

- Problem:
  Exhaustive simulation not always possible.
- Solution:
  Formally specify and check the design.

- Formal verifier: An automated decision procedure that can prove or disprove a set of statements in some logical system of reasoning.

Formal Verification

- System: Design to Check
- Properties: Requirements

- Problem in Formal Verification:
  What to compare against ?
  i.e. How to formulate properties.

Properties or Requirements

- **Safety properties:** Invariance
- Failure can be demonstrated by a finite sequence, i.e. something happens.
- E.g., absence of deadlock

Properties

- **Liveness properties:** eventuality
- Failure can only be demonstrated by an infinite sequence, i.e. something never happens.
- E.g. acknowledgement of a request.

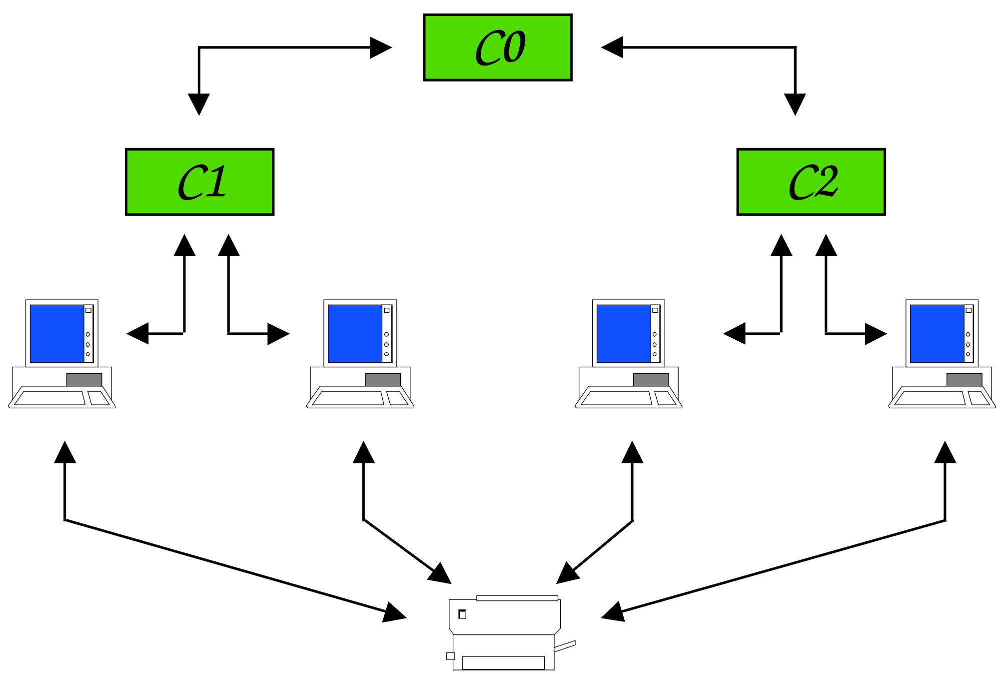
- Examples: Responsiveness, Absence of unsolicited response, Intermittent assertion, Guaranteed accessibility.

Some Methods for Verification

	Technique	Expressibility	Automation
	Simulation	Lowest: No queries, examine all	High: Once the vectors have been found
	Symbolic simulation/ STE	Low: Limited Properties	High
	FSM based	High	High
	Theorem proving	Highest	Lowest
	Equivalence Checking	Medium	High

Example: tree arbiter

- Four computers sharing a single printer
  - ensure **mutual exclusion** and **responsiveness**



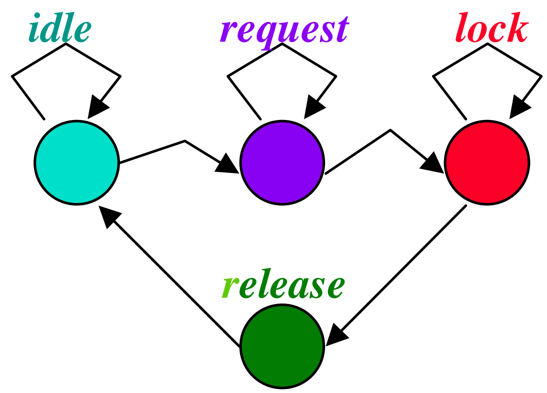
Properties

- It is never the case that two processors simultaneously *lock* the printer (Mutual Exclusion).

- If a processor requests the printer, it eventually gets it (Responsiveness).

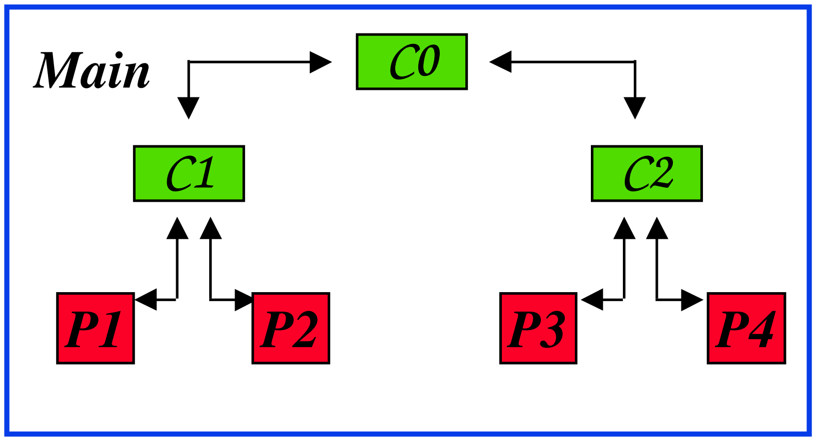
Design Description

```
/* Abstracted design in simplified pseudo Verilog -like form */
/* just for illustration */
/* procStateType i.e. states of processor are  idle,request,lock,release */

input clk, tok;
output state;
procStateType reg state;
wire randomChoice;

 /* nondeterministic assignment */
/* using system task */
assign randomChoice = $ND(0,1);

always @(posedge clk)
      begin
            if (state == idle && randomChoice)
                  state = request;

.......................................................................................
```

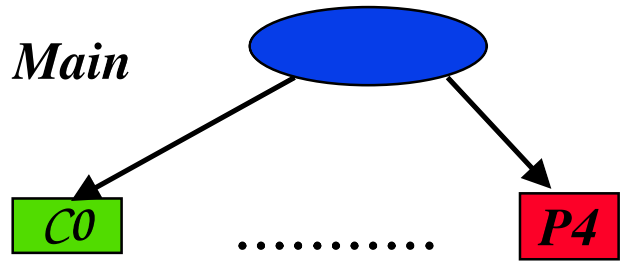


Processor

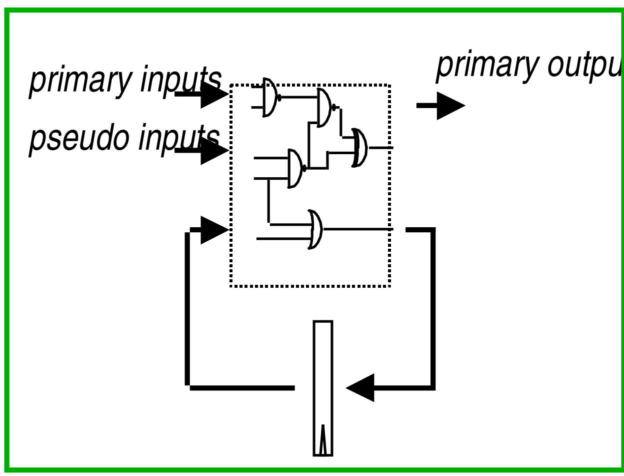
Representations of the Design



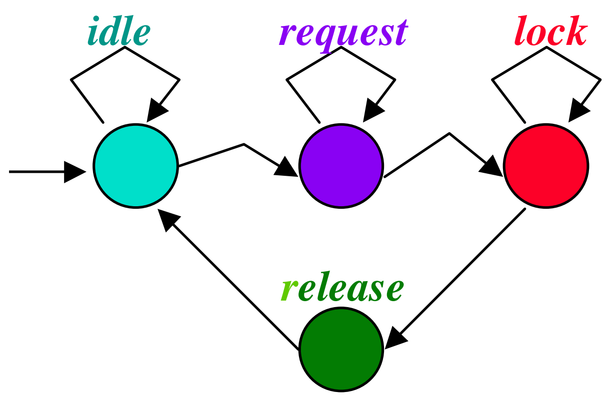
Interacting FSMs



Hierarchy Tree



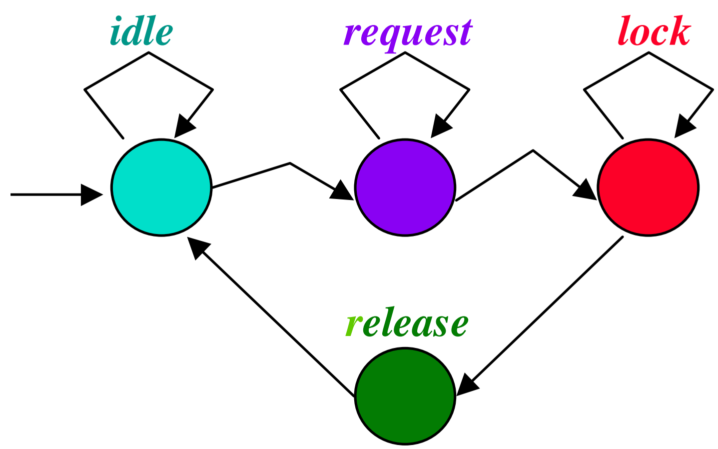
Network



FSM

Finite State Machine

- Q: A Set of States, I: Initial States
- O:Set of Inputs/Outputs
- T:Q X O X Q -> {0,1} Transition Relation
- x,y: present/next state var, o:output/input
- T(x,y), T(x,o,y), I(x),



- Next States of $R(x) = \exists_y T(y,x) .R(y)$

Fairness/ Acceptance Conditions

- Fairness/Acceptance Conditions: Way of restricting examined behavior.
  Language = Acceptable behavior

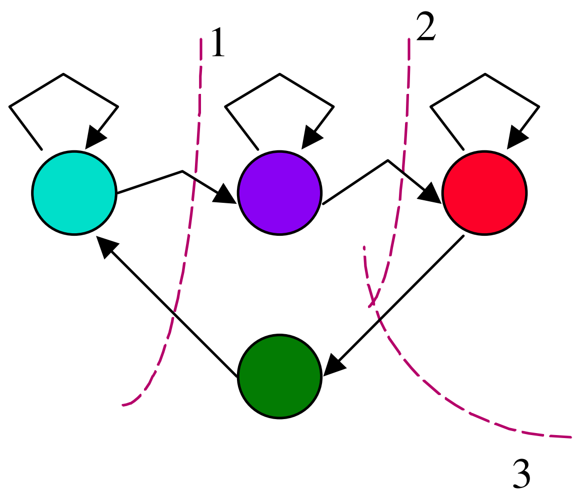
- Fairness in example: Accept only behavior where no processor is infinitely in lock.
- Automata = FSM + Fairness.

Different Classes of W-Automata

- W-Automata: Accepting Infinite Behavior.
  Examples:
  - Buchi Fairness: F(Ui)
    Sets of States Ui that must be traversed infinitely often in any acceptable behavior.
  - Streett Fairness:
  - Rabin Fairness:
  - ....

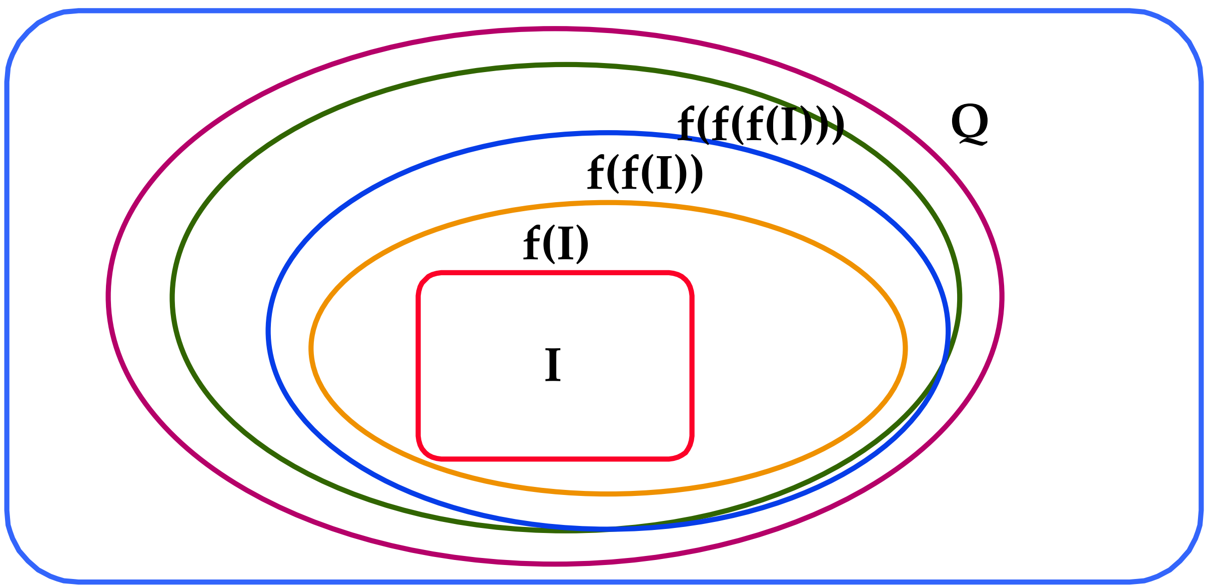
Basic algorithms

- Key verification computations are iterative Fixed Points.
  - Each step in the fixed point involves traversing the state transition graph of the FSM.



Fixed Point Computation

- Fixed Point: Let f( ) be a monotone function, the fixed point FP of f given I (an initial set), is set $Q=f^i(I)$, where $f(Q) = Q$.
- f( ) decreasing $\Rightarrow$ FP called Greatest Fixed Point (GFP).
- f( ) increasing $\Rightarrow$ FP called Least Fixed Point (LFP).
- FP dependent on initial set.



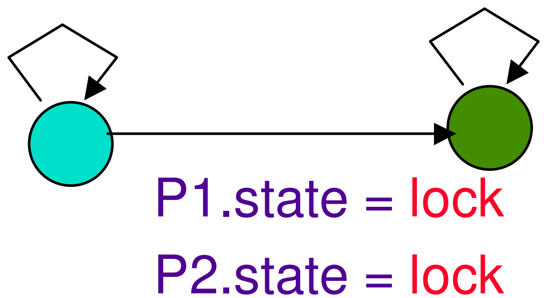
Language Containment

- Problem: Is what I designed what I wanted ?
- Language containment solution:
  (Vardi, Wolper 86) (Kurshan 87)
  Specify design and property as FSMs/automata.

- Language containment engine: Complement the property FSM and compose it into design FSM to form product. Traverse the product to check for language emptiness (no behavior that satisfies fairness constraints).

Properties as Automata

- It is never the case that two processors simultaneously *lock* the printer (Mutual Exclusion).



only accept behavior

where state occurs

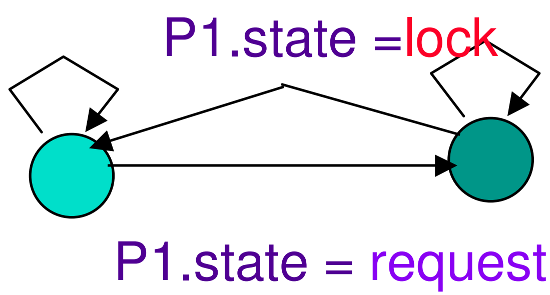
infinitely

- If a processor requests the printer, it eventually gets it (Responsiveness)

only accept behavior

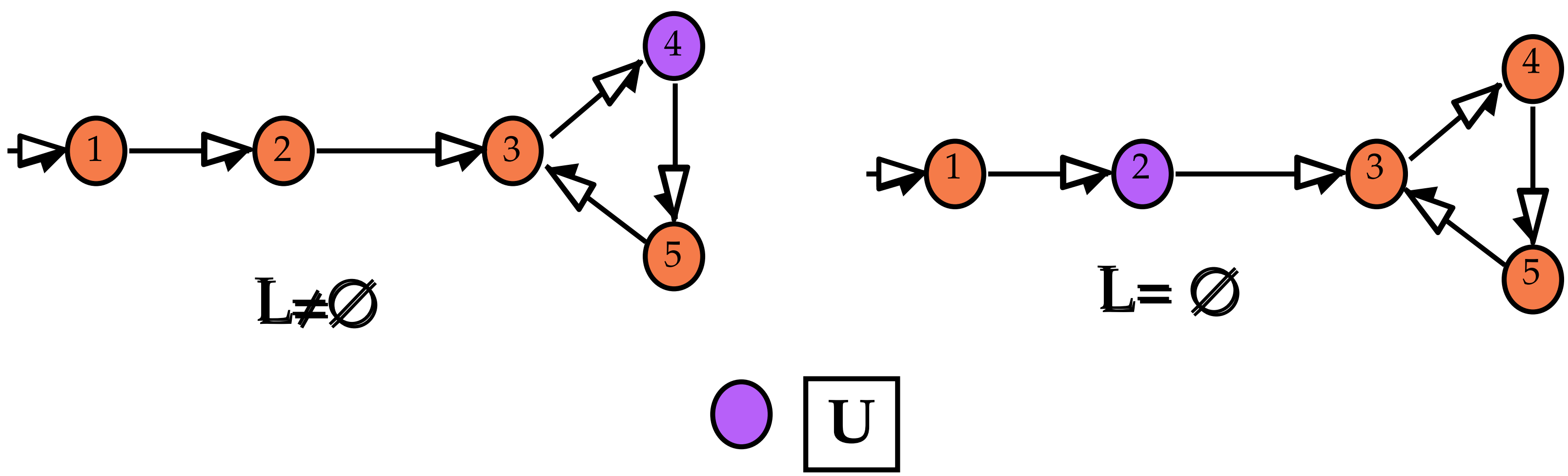
where state occurs

infinitely



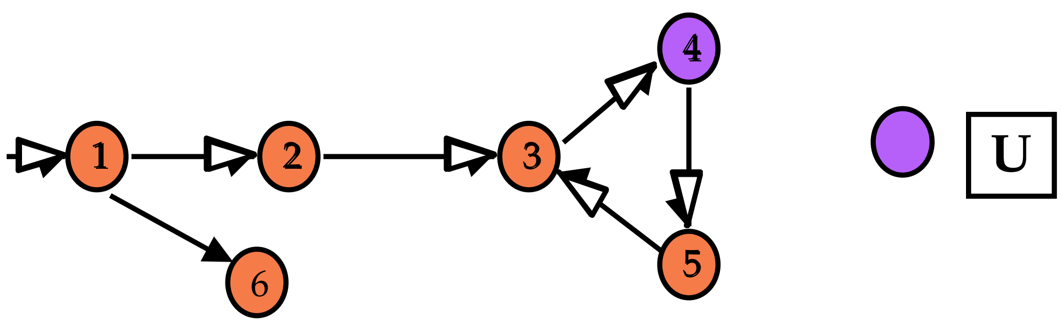
Language Emptiness

- Language is not empty $\leftrightarrow$ there is a path from the initial states to a cycle satisfying fairness constraints.



Language Emptiness

- Fair$^+$ states
  - reachable states on a path to a cycle satisfying fairness constraints.
  - are witness to the non-emptiness of the product machine language.



Fair$^+$ = {1,2,3,4,5}

Computing Fair$^+$

- Algorithm Computing Fair$^+$
  Begin with Reachable states (R)
  - Alternately remove until fixed point (GFP)
    - » States that cannot reach a cycle (FSS operator)
    - » States that cannot reach a state in U (FFP operator)
    - » (iterate)
  - return remaining states
- Fair$^+$ = GFP(f(Q), R(x))
  where f(Q) = FFP(FSS(Q(x)))

Computing Fair$^+$