Outline

- **Deadlock Detection**

Actual Goal

- Deadlock for concurrent message-passing blocking C programs

- Tackle complexity using automated abstraction and compositional reasoning

- Obtain precise answers using automated iterative abstraction refinement

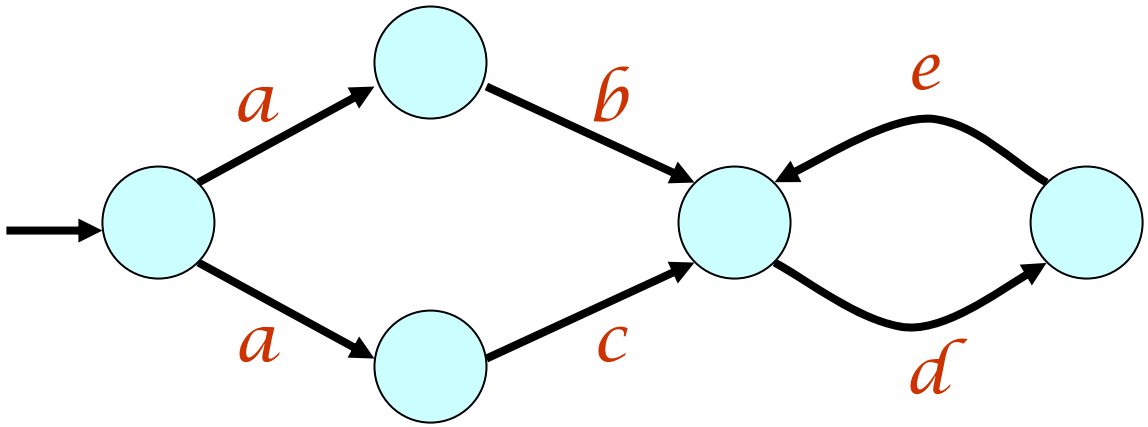
For this part of the lecture

- Focus on finite state machines
  - Labeled transition systems (LTSs)

- Parallel composition of state machines
  - Synchronous communication
  - Asynchronous execution
  - Natural for modeling blocking message-passing C programs

Finite LTS

- $\mathcal{P} = (\mathcal{Q} , \mathcal{I} , \Sigma , \mathcal{T})$
  - $\mathcal{Q} \equiv$ **non-empty set of states**
  - $\mathcal{I} \in \mathcal{Q} \equiv$ **initial state**
  - $\Sigma \equiv$ **set of actions** $\equiv$ **alphabet**
  - $\mathcal{T} \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q} \equiv$ **transition relation**



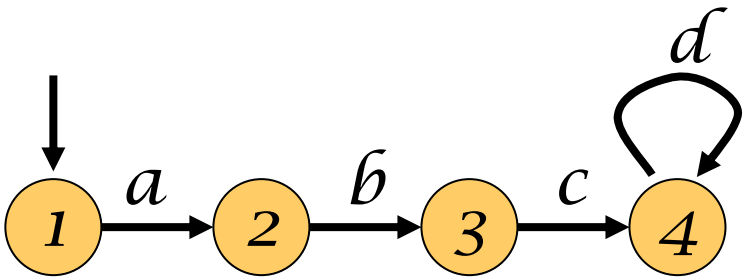
$$\Sigma(\mathcal{P}) = \{a,b,c,d,e,f\}$$

Concurrency

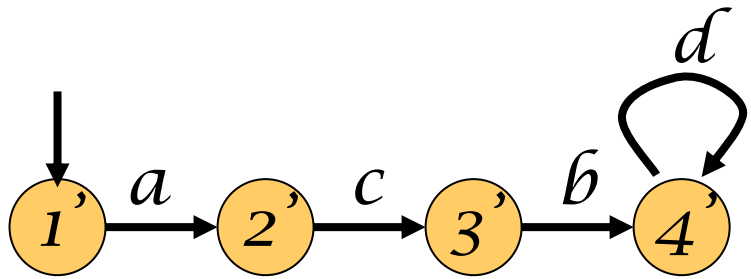
- Components communicate by handshaking (synchronizing) over shared actions
- Else proceed independently (asynchronously)
- Essentially CSP semantics

- Composition of $\mathcal{A}_1$ & $\mathcal{A}_2 \equiv \mathcal{A}_1 \parallel \mathcal{A}_2$

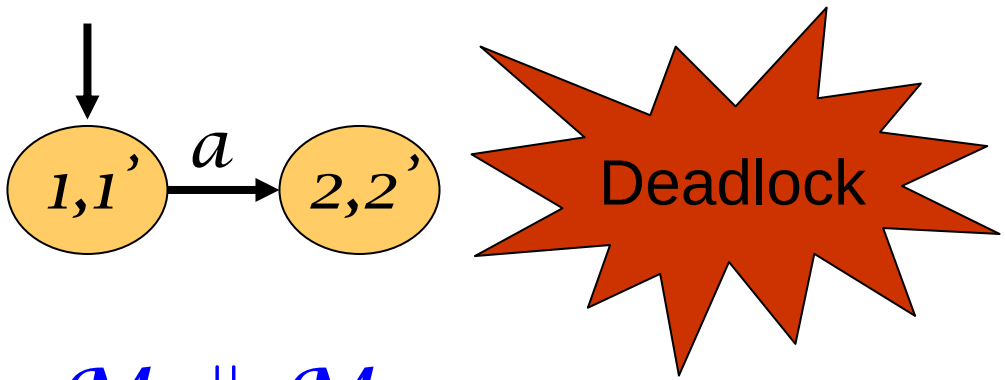
Deadlock



$\mathcal{M}_1 \quad \Sigma = \{a,b,c,d\}$



$\mathcal{M}_2 \quad \Sigma = \{a,b,c,d\}$

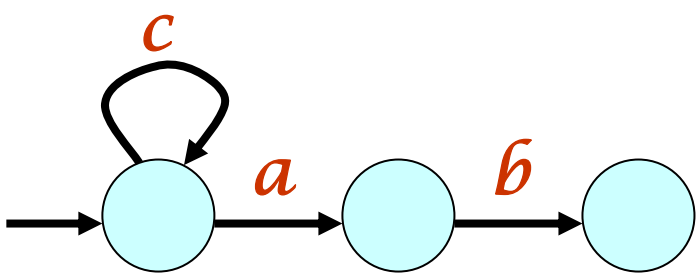


$\mathcal{M}_1 \parallel \mathcal{M}_2$

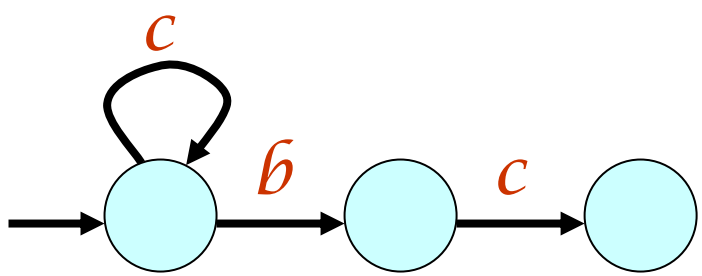
Deadlock $\Leftrightarrow$ a reachable state cannot perform any actions at all

Deadlock and Composition

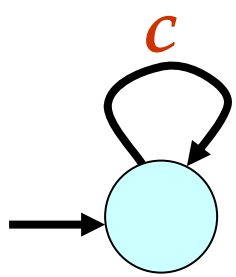
Deadlock



$\mathcal{M}_1$



$\mathcal{M}_2$

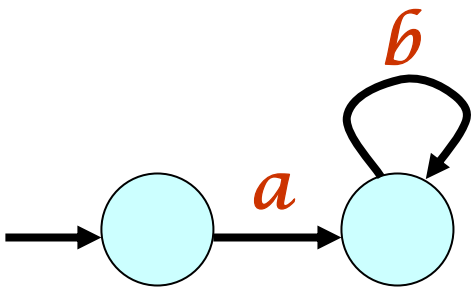


$\mathcal{M}_1 \| \mathcal{M}_2$

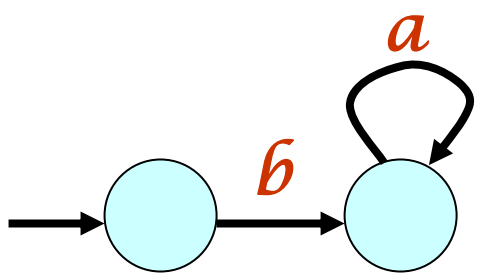
No Deadlock

Deadlock and Composition

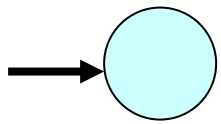
No Deadlock



$\mathcal{M}_1$



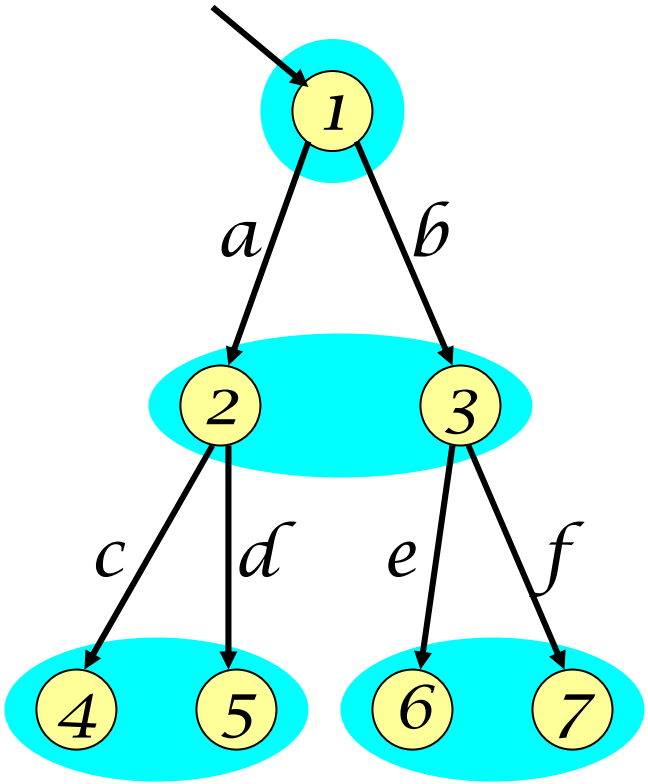
$\mathcal{M}_1$



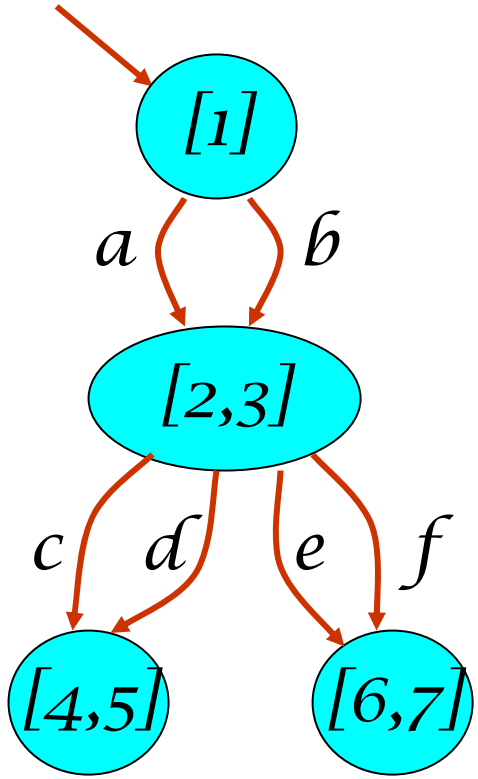
Deadlock

$\mathcal{M}_1 \| \mathcal{M}_2$

Conservative Abstraction



$\mathcal{P}$

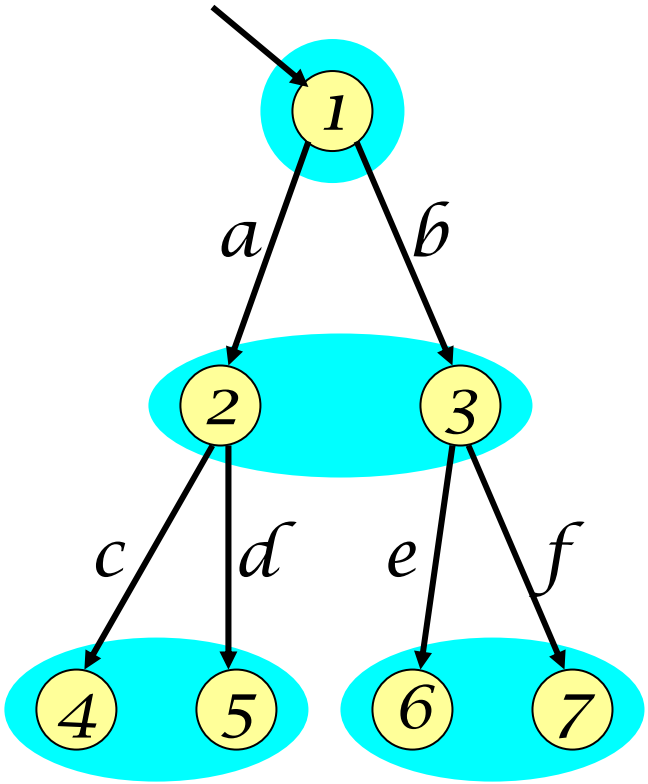


$\mathcal{A}$

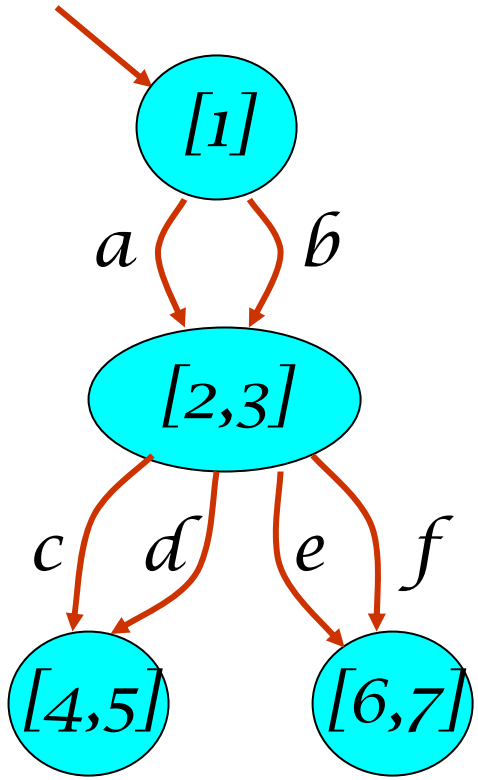
Conservative Abstraction

- **Every** trace of $\mathcal{P}$ is a trace of $\mathcal{A}$
  - **Preserves safety** properties: $\mathcal{A} \models \phi \Rightarrow \mathcal{P} \models \phi$
  - $\mathcal{A}$ **over-approximates** what $\mathcal{P}$ can do
- **Some** traces of $\mathcal{A}$ may **not** be traces of $\mathcal{P}$
  - May yield **spurious** counterexamples - $\langle a, e \rangle$
- **Eliminated** via abstraction **refinement**
  - **Splitting** some clusters in smaller ones
  - Refinement can be **automated**

Original Abstraction

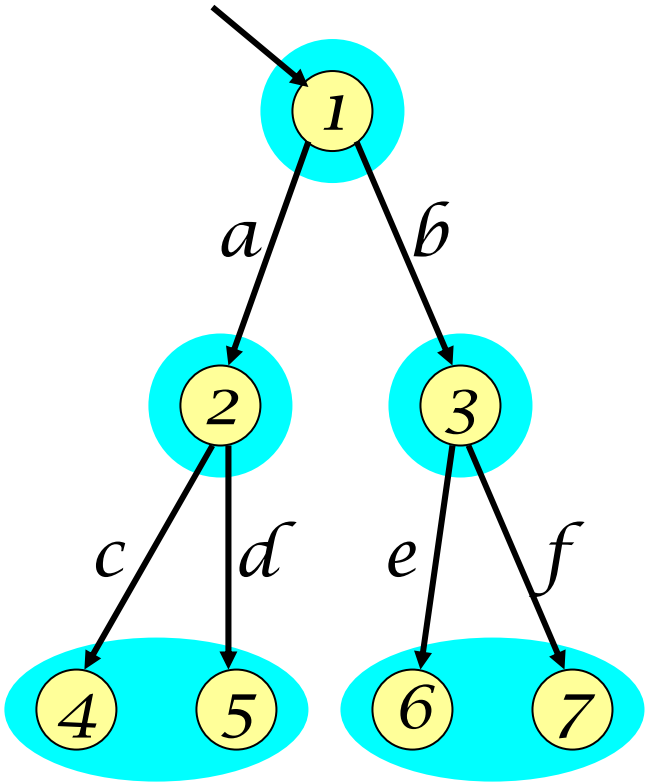


$\mathcal{P}$

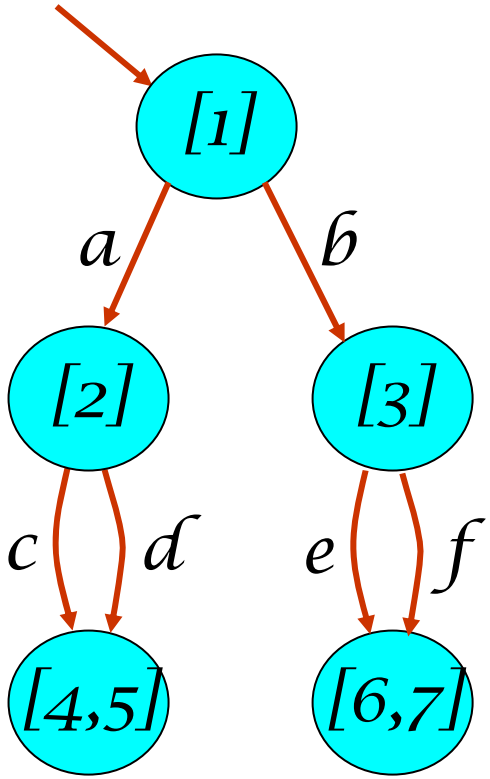


$\mathcal{A}$

Refined Abstraction



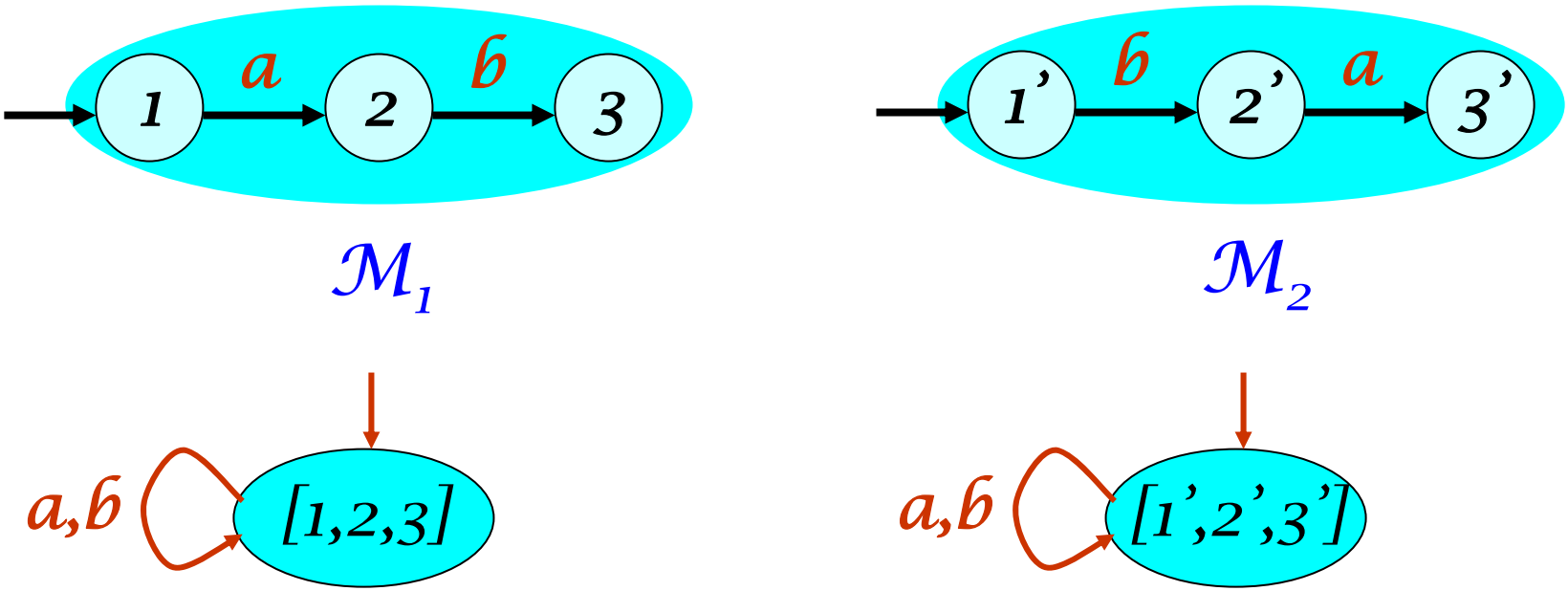
$\mathcal{P}$



$\mathcal{A}$

Deadlock : Problem

- **Deadlock is not preserved by abstraction**



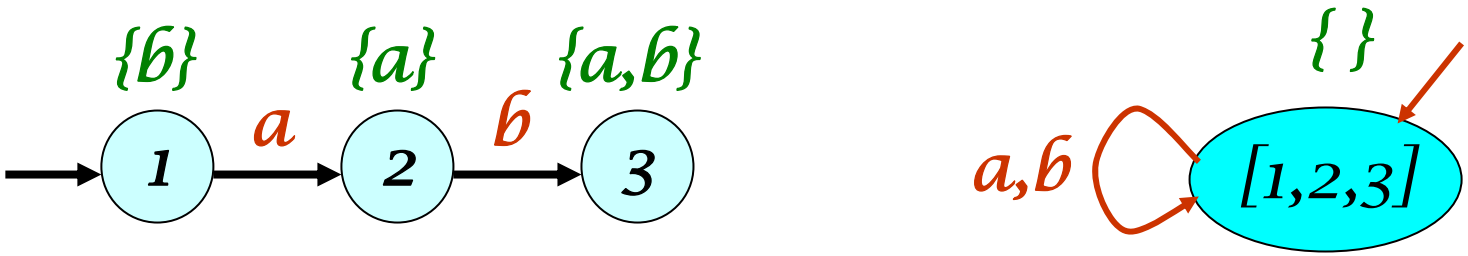
Deadlock Detection : Insight

- **Deadlock ⇔ a reachable state cannot perform any actions at all**
  - **Deadlock depends on the set of actions that a reachable state cannot perform**

- **In order to preserve deadlock $\mathcal{A}$ must over-approximate not just what $\mathcal{P}$ can do but also what $\mathcal{P}$ refuses**

Refusal & Deadlock

- $\mathcal{R}ef(s)$ = set of actions s cannot perform

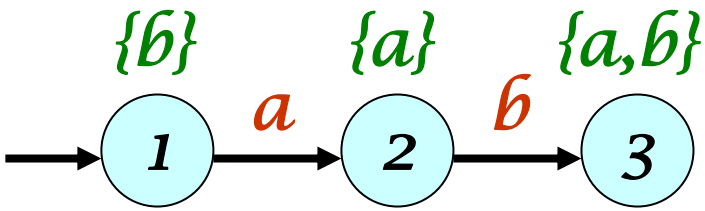


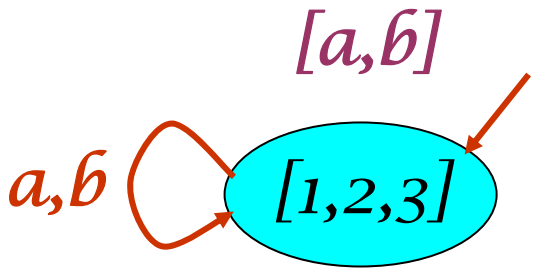
- $\mathcal{M}$ **deadlocks** iff there is a reachable state s such that $\mathcal{R}ef(s) = \Sigma$
  - Denote by $\mathcal{DL}ock(\mathcal{M})$

- $\mathcal{R}ef([s_1 .. s_n]) = \mathcal{R}ef(s_1) \cap .. \cap \mathcal{R}ef(s_n)$

Abstract Refusal

- $\mathcal{AR}([s_1 .. s_n]) = \mathcal{R}ef(s_1) \cup .. \cup \mathcal{R}ef(s_n)$





- $\mathcal{AR}([M_1] .. [M_n]) = \mathcal{AR}([M_1]) \cup .. \cup \mathcal{AR}([M_n])$

Abstract Deadlock

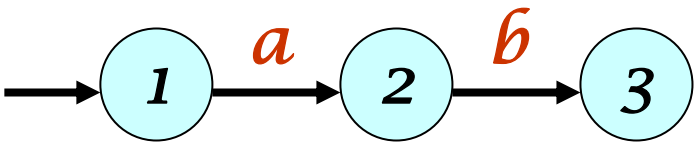
- $\mathcal{M}$ **abstractly deadlocks iff there is a reachable state** s **such that** $\mathcal{AR}(s) = \Sigma$
  - **Denote by** $\mathcal{ADLock}(\mathcal{M})$

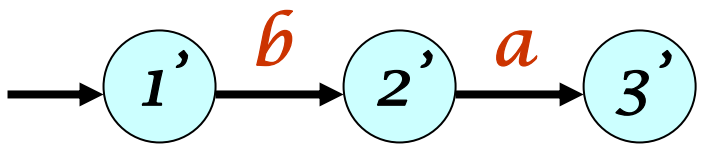
$$\neg\ \mathcal{ADLock}([M_1] \parallel .. \parallel [M_n])$$

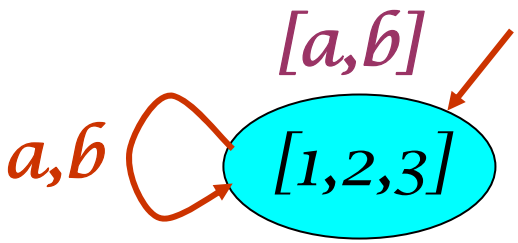
$$\Rightarrow$$

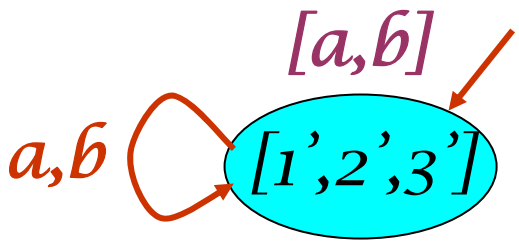
$$\neg\ \mathcal{DLock}(M_1 \parallel .. \parallel M_n)$$

Iterative Deadlock Detection

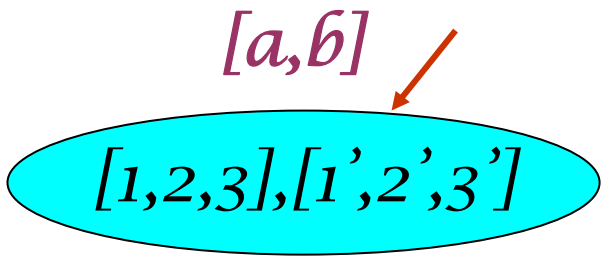




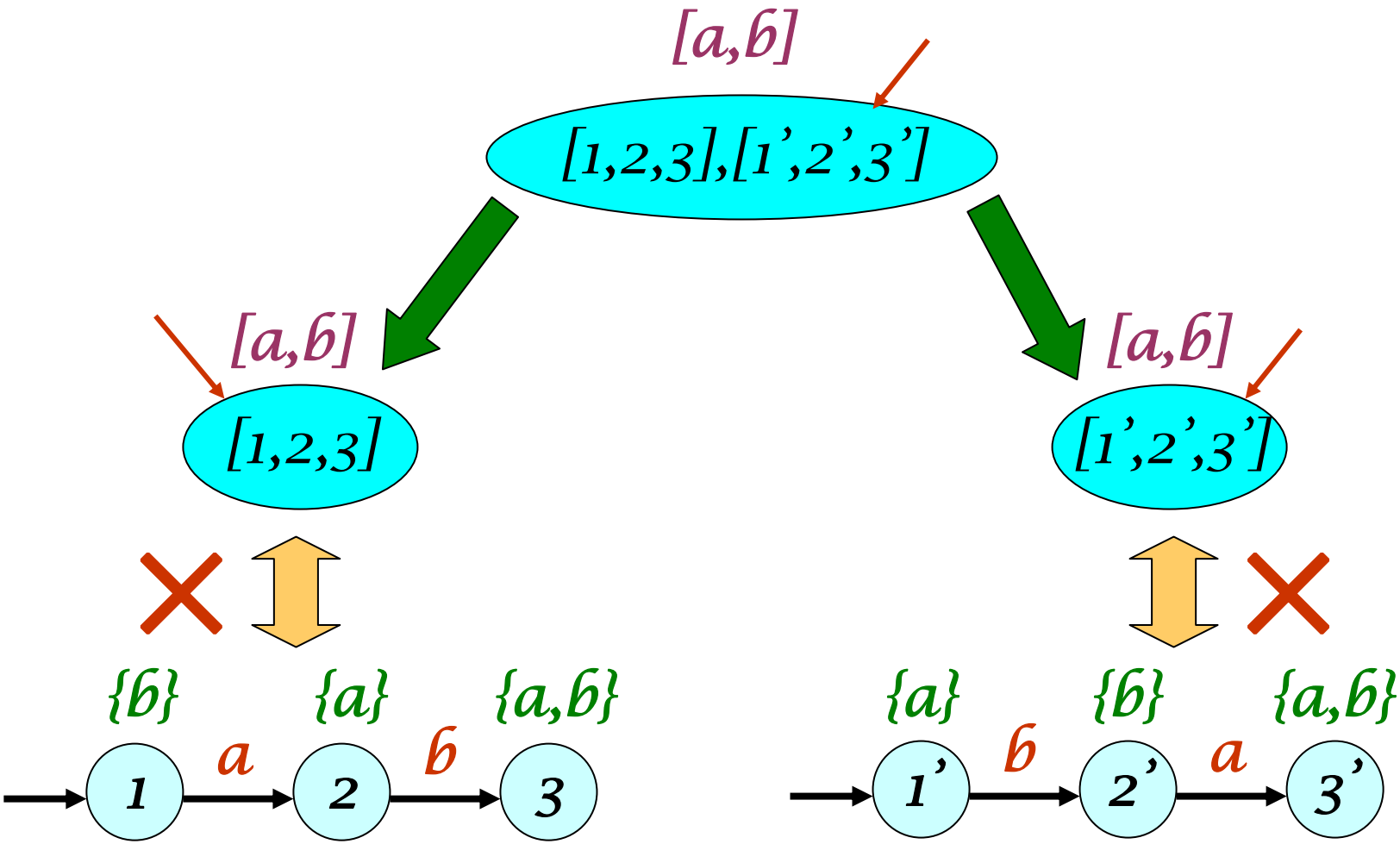




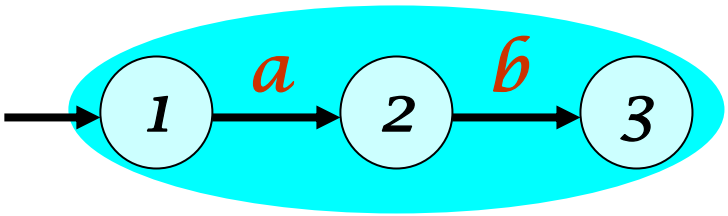
Counterexample to Abstract Deadlock

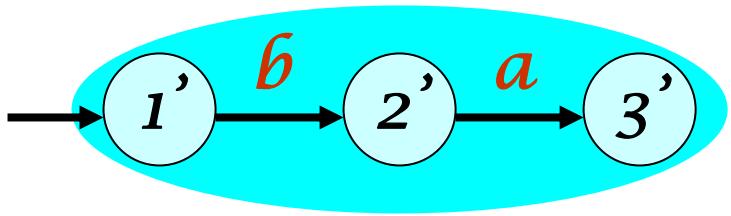


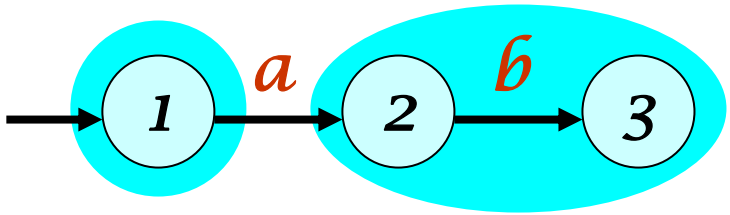
Counterexample Validation

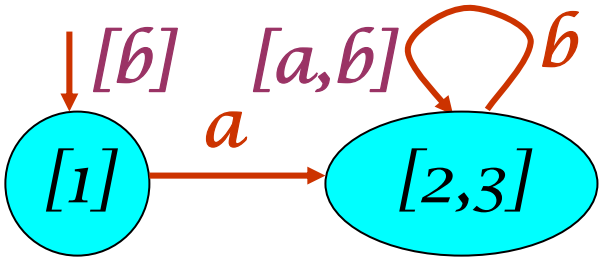


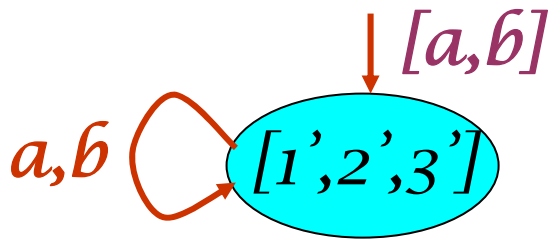
Refinement



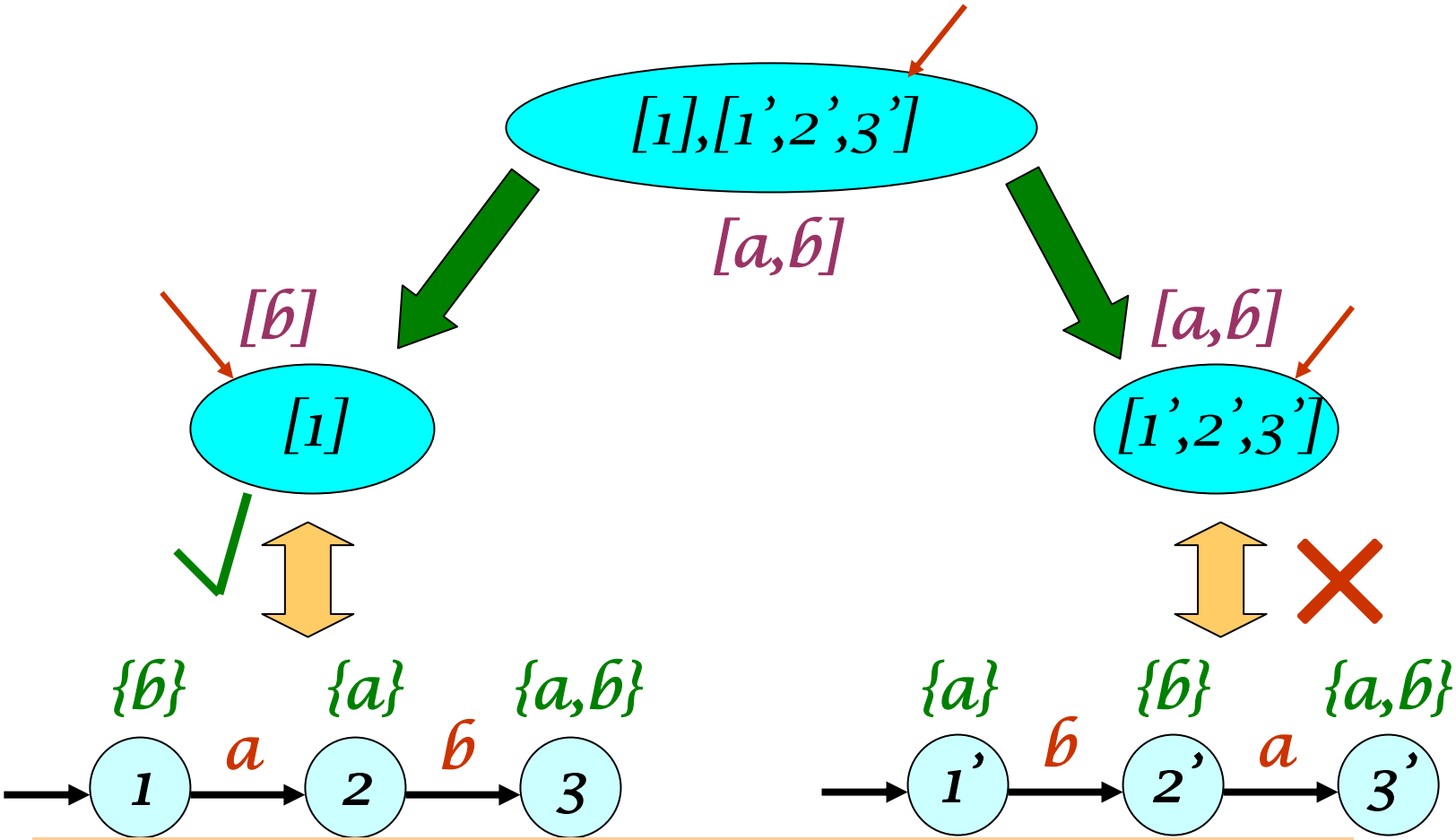






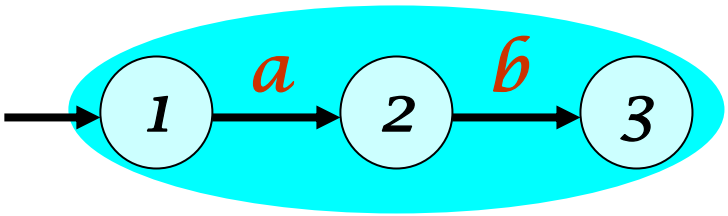


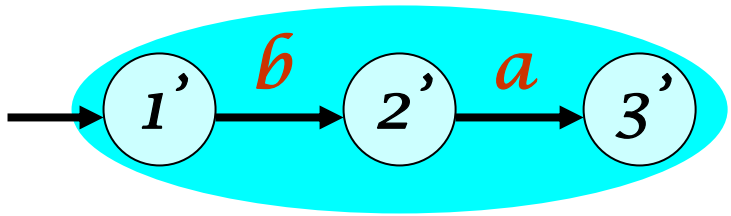
Counterexample Validation

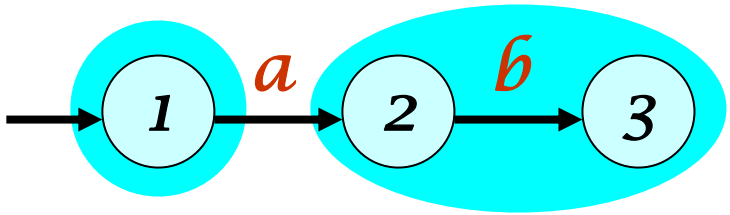


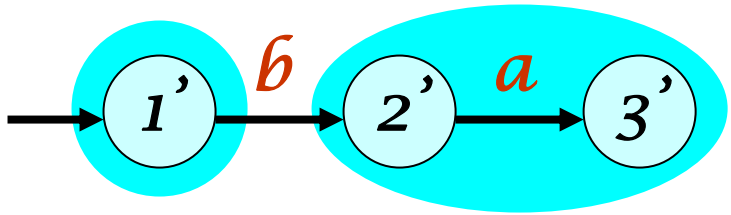
Another spurious counterexample

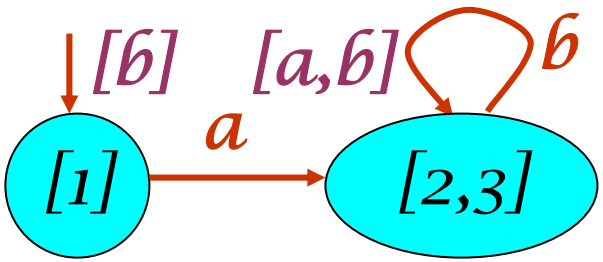
Refinement

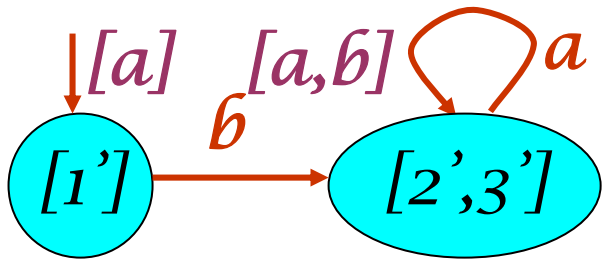




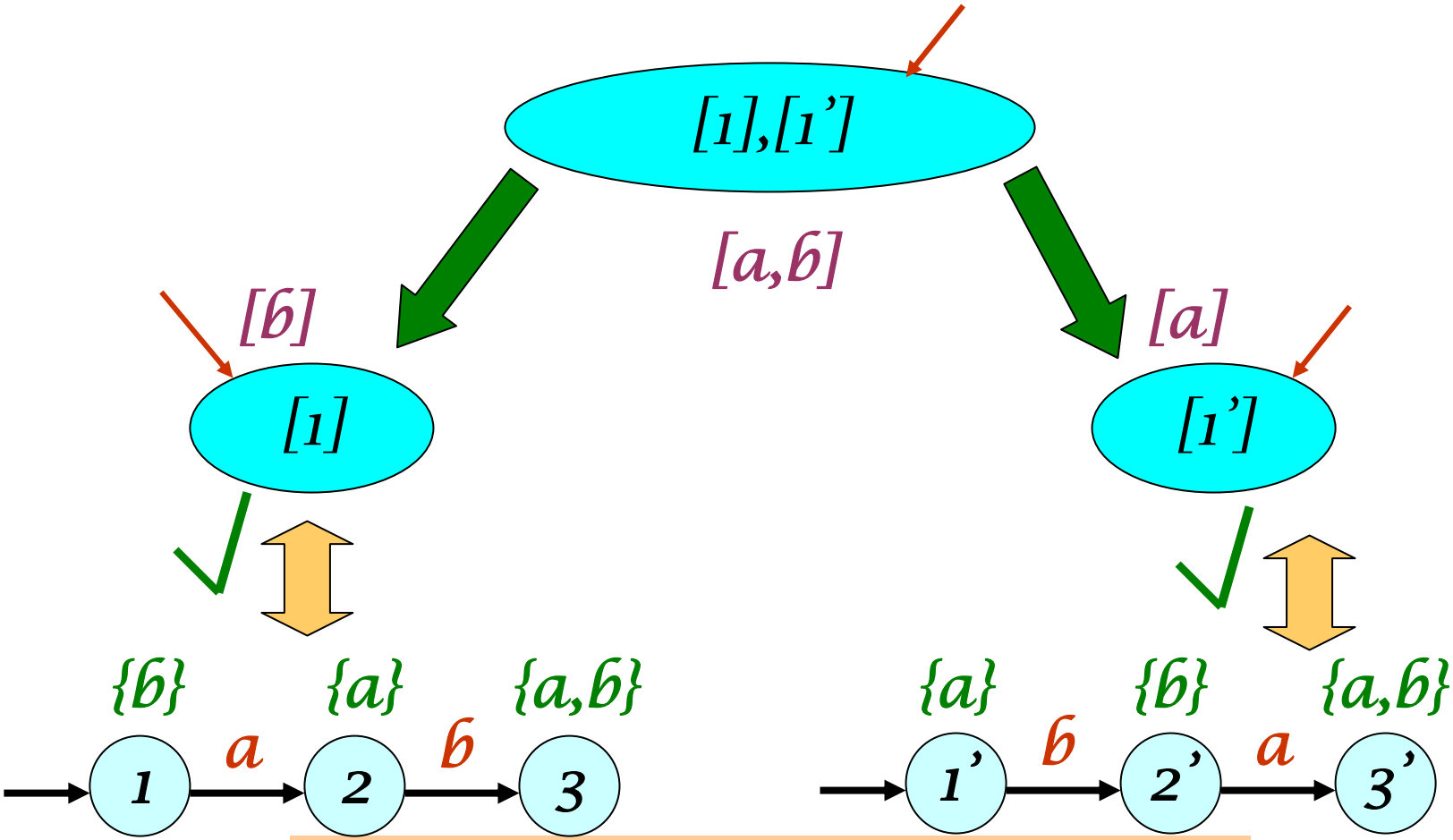






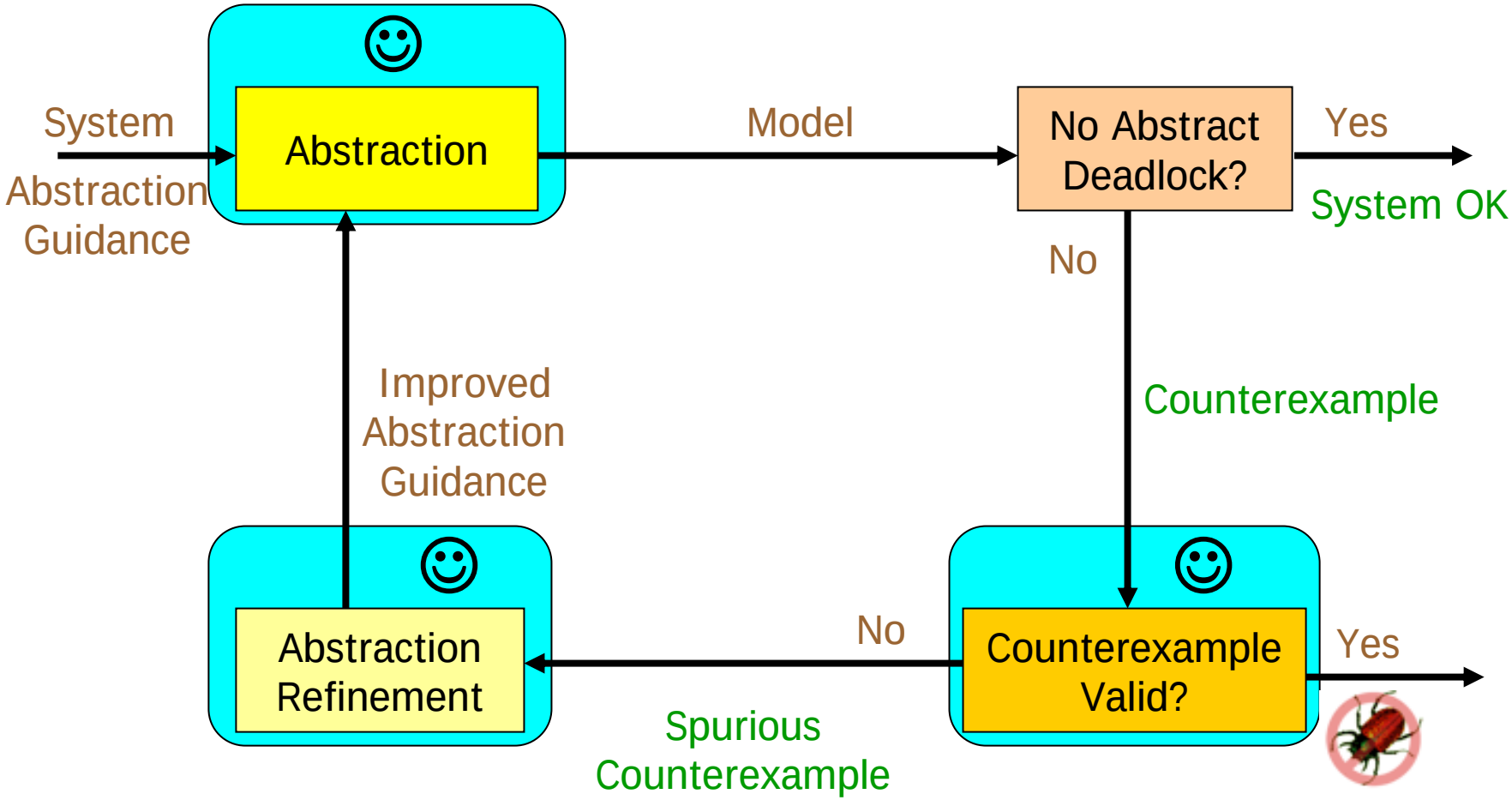


Counterexample Validation



Real Deadlock Detected

Iterative Deadlock



Case Studies

- **MicroC/OS-II**
  - **Real-time OS for embedded applications**
  - **Widely used (cell phones, medical devices, routers, washing machines…)**
  - **6000+ LOC**

•ABB IPC Module

–Deployed by a world leader in robotics

–15000+ LOC

–4 components

–Over 30 billion states after predicate abstraction

Results

	Plain			IterDeadlock			
Name	St	T	Mem	St	It	T	Mem
ABB	*	*	162	1973	861	1446	33.3
SSL	25731	44	43.5	16	16	31.9	40.8
µCD-3	*	*	58.6	4930	120	221.8	15
µCN-6	*	*	219.3	71875	44	813	30.8
DPN-6	*	*	203	62426	48	831	26.1
DPD-10	38268	87.6	17.3	44493	51	755	18.4

* indicates out of time limit (1500s)

Results

	Plain			IterDeadlock			
Name	St	T	Mem	St	It	T	Mem
ABB	*	*	162	1973	861	1446	33.3
SSL	25731	44	43.5	16	16	31.9	40.8
μCD-3	*	*	58.6	4930	120	221.8	15
μCN-6	*	*	219.3	71875	44	813	30.8
DPN-6	*	*	203	62426	48	831	26.1
DPD-10	38268	87.6	17.3	44493	51	755	18.4

References

- **Sagar Chaki, Edmund M. Clarke, Joël Ouaknine, Natasha Sharygina: *Concurrent software verification with states, events, and deadlocks*. Formal Aspects of Computing 17(4): 461-483 (2005)**

- **Chiara Braghin, Natasha Sharygina, Katerina Barone-Adesi: *Automated Verification of Security Policies in Mobile Code,* In Proc. of Integrated Formal Methods 2007 conf., LNCS 4591:37 - 54 (2007)**