

# Basics of Model Checking

# Modal Logic

# Kripke Structure

- $K = \langle S, R, L \rangle$

$S$ : set of states (may be infinite)

$R$ : transition relation between states

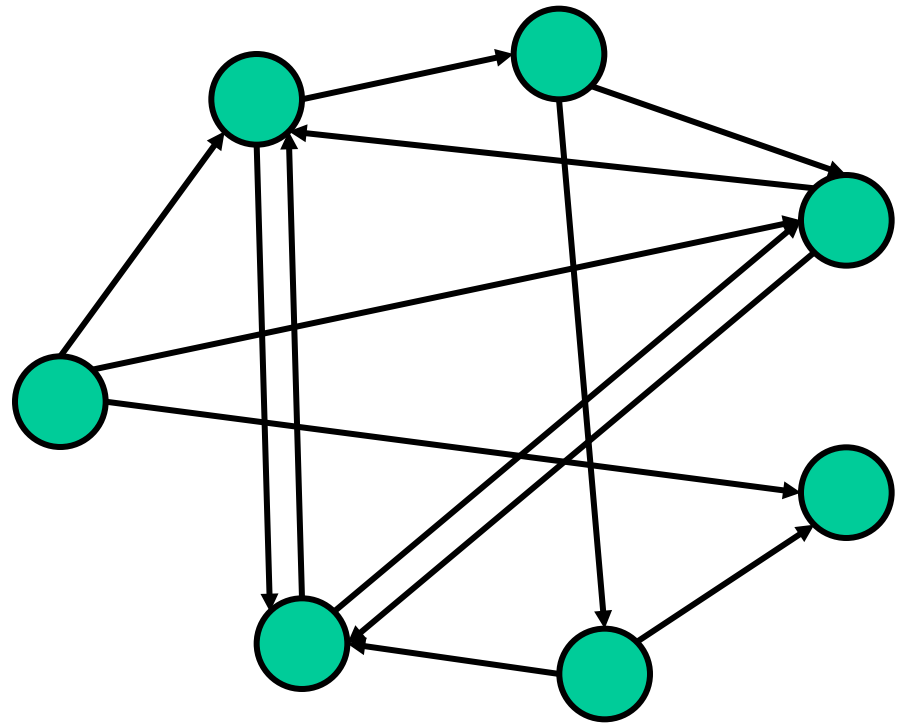
$$R \subseteq S \times S$$

$L$ : map from states to sets of propositional symbols prop. symbols

$L(s)$  denotes the set of prop. symbols  
that hold at state  $s \in S$

# Kripke Structure

- $K = \langle S, R, L \rangle$
- $G = \langle S, R \rangle$   
directed graph



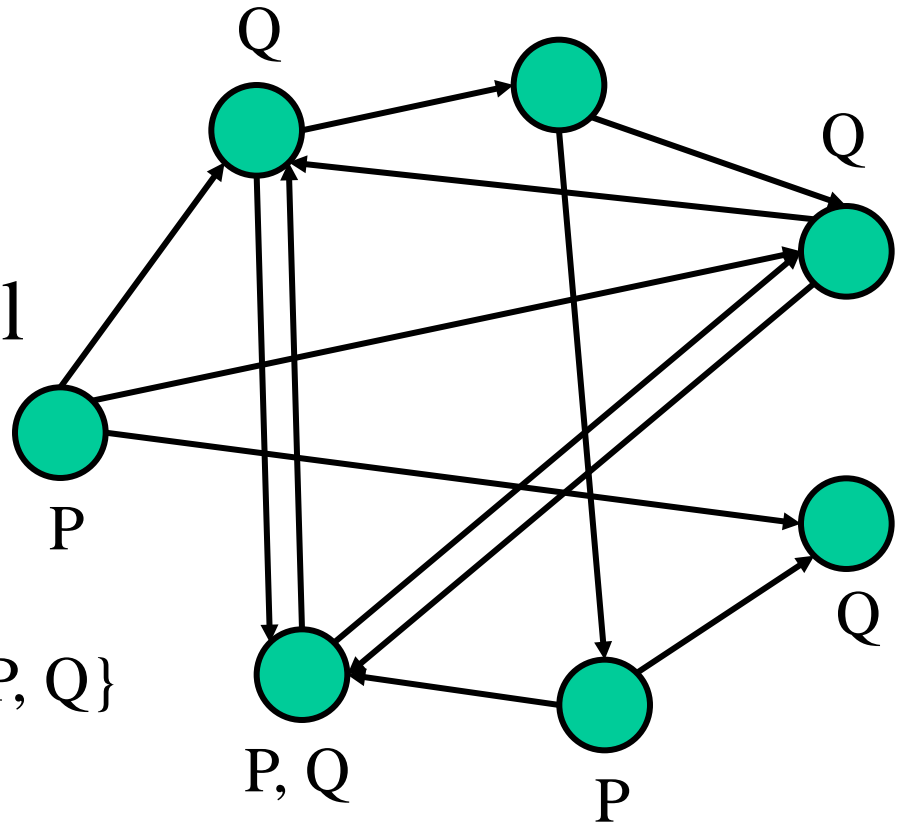
# Kripke Structure

- $K = \langle S, R, L \rangle$

- $L : S \rightarrow 2^{\text{Atom}}$

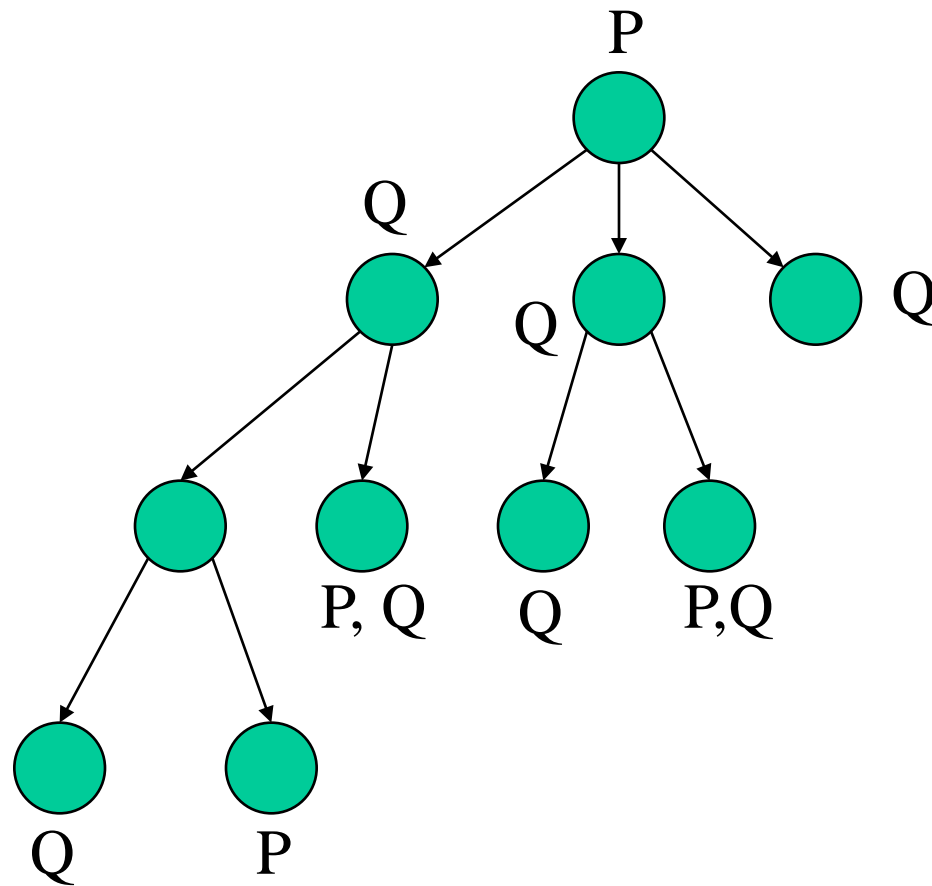
**Atom** : the set of all  
prop. sym.

$$\text{Atom} = \{P, Q\}$$



# Various Kripke Structures

- Tree
- Forest
- Infinite tree



# Modal Formula

$\varphi, \psi$	$::=$	$P$	prop. symbols
		$\neg\varphi$	negation
		$\varphi \wedge \psi$	conjunction
		$\varphi \vee \psi$	disjunction
		$\Box\varphi$	necessity
		$\Diamond\varphi$	possibility

# Semantics

$s \models P$  **iff**  $P \in L(s)$

$s \models \neg\varphi$  **iff** **not**  $s \models \varphi$

$s \models \varphi \wedge \psi$  **iff**  $s \models \varphi$  **and**  $s \models \psi$

$s \models \varphi \vee \psi$  **iff**  $s \models \varphi$  **or**  $s \models \psi$

$s \models \Box\varphi$  **iff**  $t \models \varphi$  **for any**  $t$  **s.t.**  $R(s, t)$

$s \models \Diamond\varphi$  **iff**  $t \models \varphi$  **for some**  $t$  **s.t.**  $R(s, t)$

$\Diamond\varphi$  is equivalent to  $\neg\Box\neg\varphi$

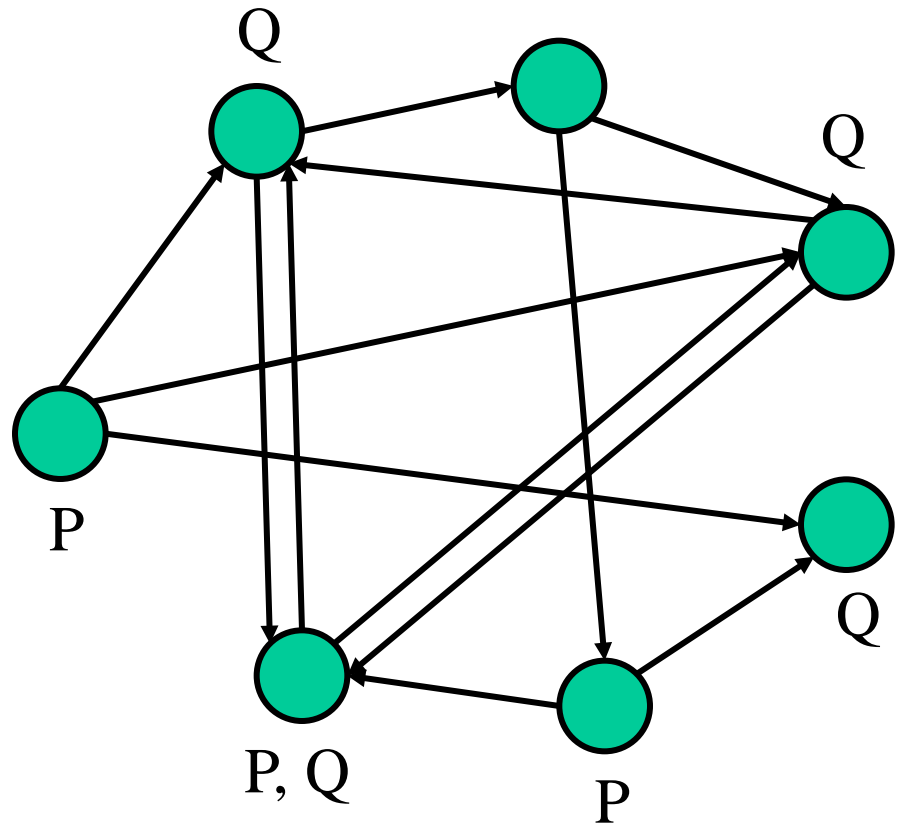


# Semantics

$\square Q$

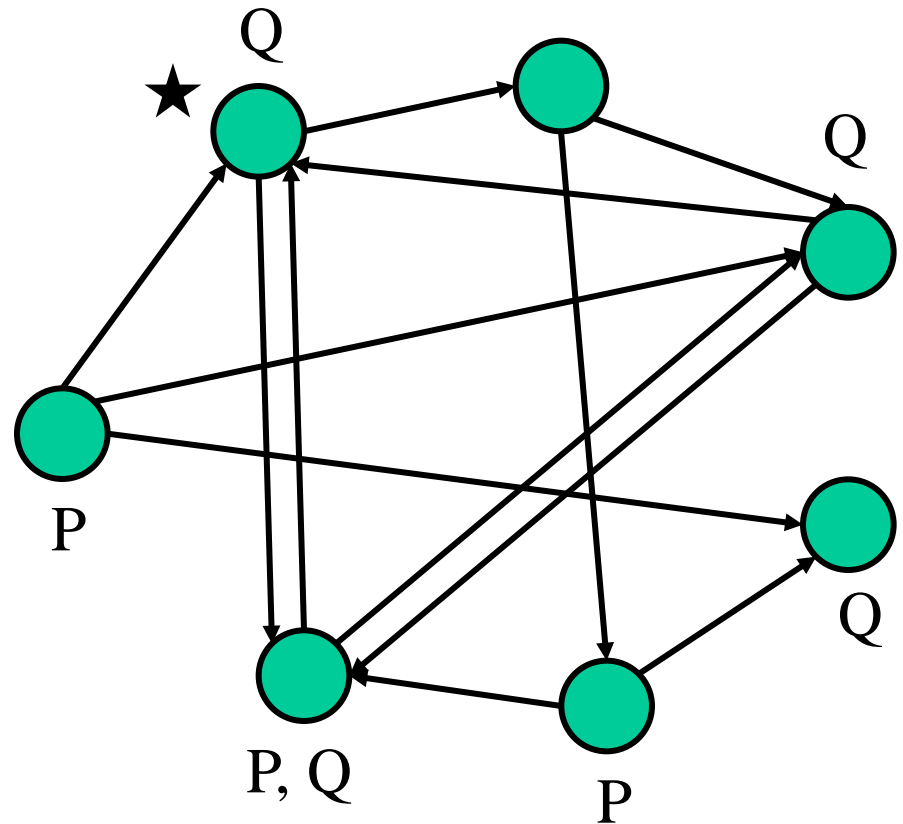
$\diamond(P \wedge Q)$

$\square \diamond P$



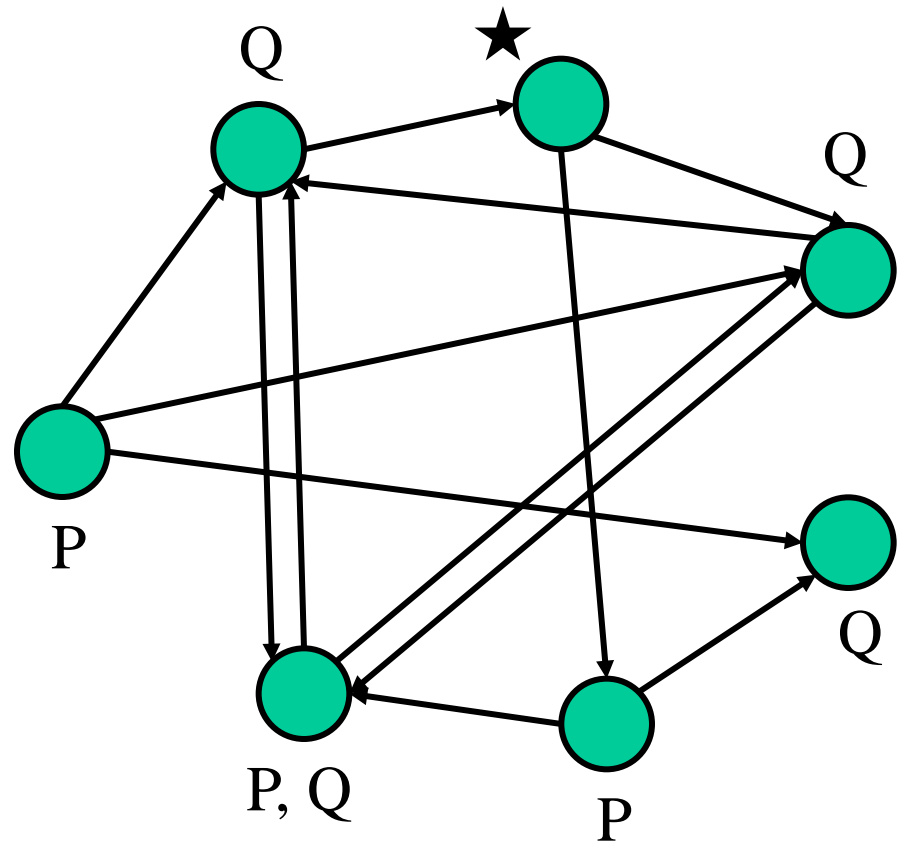
# Which holds at ★?

1.  $\Box Q$
2.  $\Diamond(P \wedge Q)$
3.  $\Box \Diamond P$



# Which holds at ★?

1.  $\Box Q$
2.  $\Diamond(P \wedge Q)$
3.  $\Box \Diamond P$





# Various Modal Logics

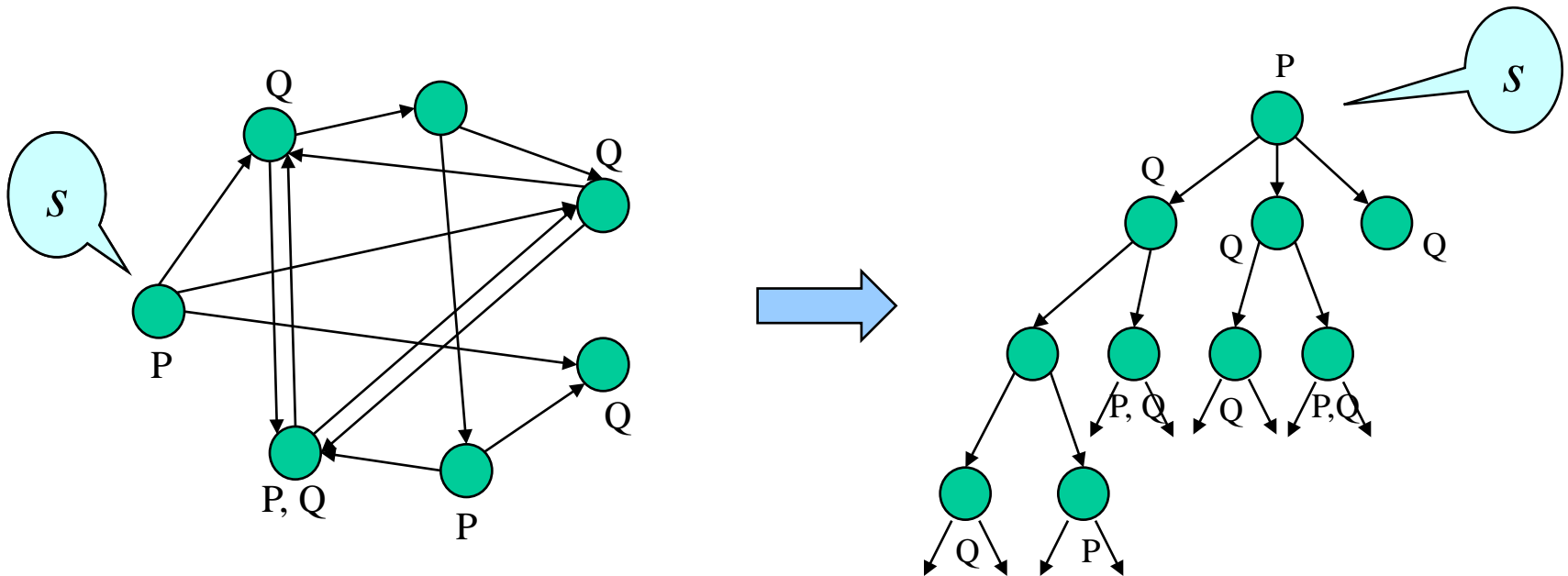
- So far is the minimal modal logic **K**
- Restricting the transition relation
  - Reflective **T** / Transitive **K4**
- Allowing recursive propositions
  - Computation Tree Logic / Modal  $\mu$ -calculus
- Multiple modalities / Operators on modalities
  - Boolean logic / Dynamic logic
- Interpreting formulas on a path over a graph
  - Linear-time temporal logic
- Restricting graphs to trees

# Satisfiability & Finite-model Property

- Modal formula  $\varphi$  is satisfiable
  - if and only if there exists a Kripke structure  $K$  and its state  $s$  such that  $s \models \varphi$  under  $K$
  - $K$  is said to be a model of  $\varphi$
- In the minimal logic, any satisfiable formula has a finite model
- Moreover, it has a finite tree model

# Tree-model Property of Modal Logic

- In general, expanding a Kripke structure (graph) at some node  $s$  yields an (infinite) tree
- If a formula is satisfiable at  $s$ , then it is also satisfiable at the root of the tree



**CTL**



# Computation Tree Logic

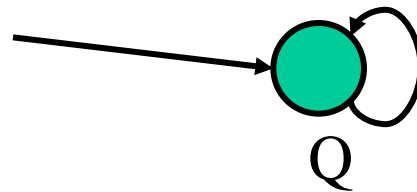
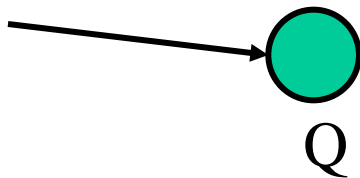
- A kind of branching-time temporal logic

$\varphi, \psi$	$::=$	...	
		<b>AG</b> $\varphi$	globally on any path
		<b>AF</b> $\varphi$	finally on any path
		<b>EG</b> $\varphi$	globally on some path
		<b>EF</b> $\varphi$	finally on some path

$\square$  is written **AX**, and  $\diamond$  is written **EX**

# About Paths

- CTL usually treats infinite execution paths
- Dead-end states complicate the situation
- So, each state is assumed to have at least one successor
- For example, a self-loop is added to a final state



# Semantics

$s \models \mathbf{AG}\varphi$  **iff**

any state  $t$  that is reachable from  $s$  satisfies

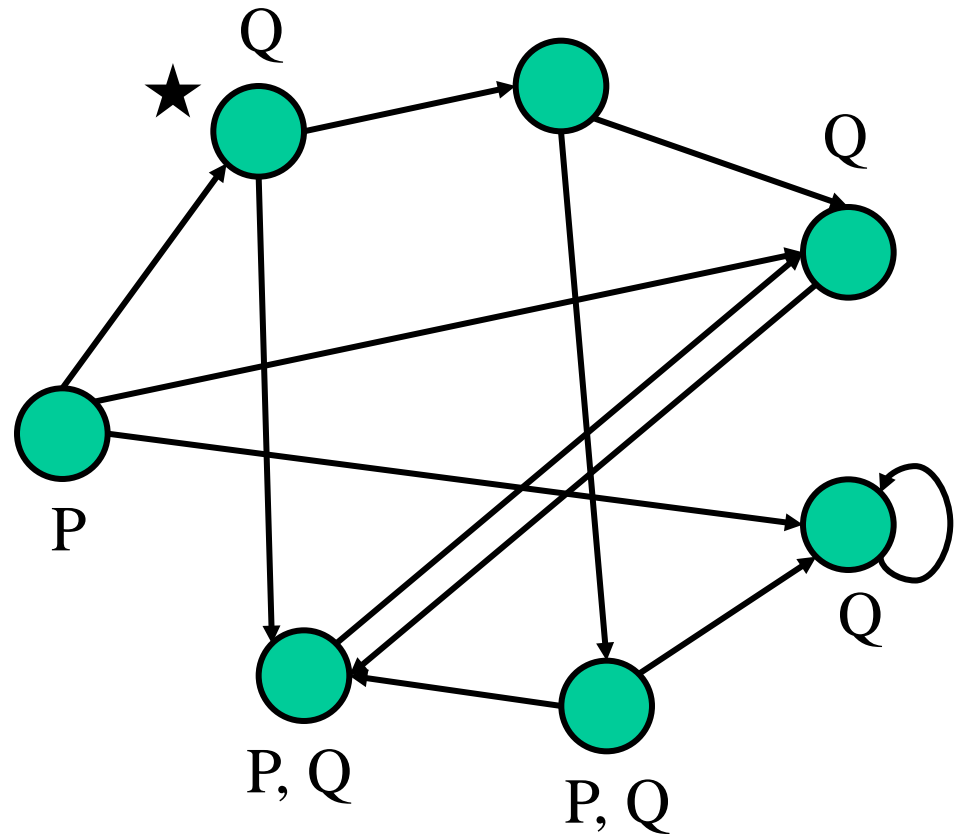
$t \models \varphi$

$s \models \mathbf{AF}\varphi$  **iff**

on any path starting from  $s$ , there exists a state  $t$  such that  $t \models \varphi$

Which holds at ★?

1. **AG P**
2. **AG  $\neg P$**
3. **AF Q**
4. **AF  $\neg Q$**



# Semantics

$s \models \mathbf{EF}\varphi$  **iff**

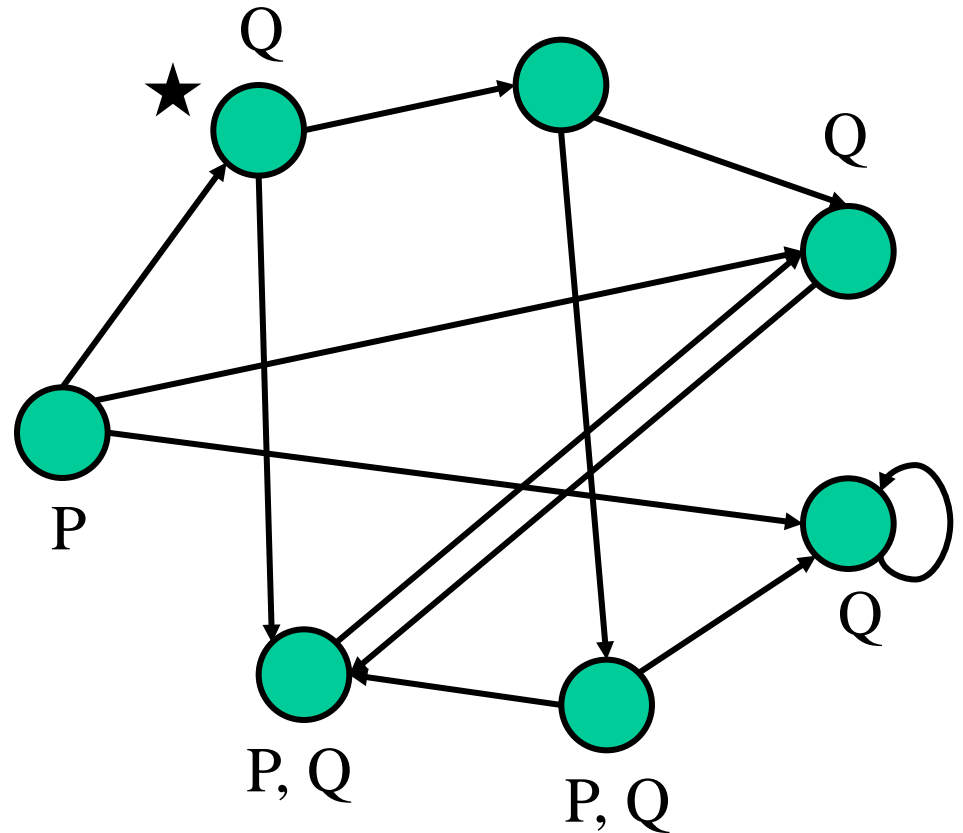
there exists a state  $t$  that is reachable from  $s$   
such that  $t \models \varphi$

$s \models \mathbf{EG}\varphi$  **iff**

there exists a path starting from  $s$  such that  
any state  $t$  on the path satisfies  $t \models \varphi$

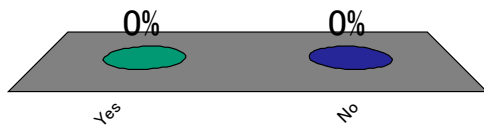
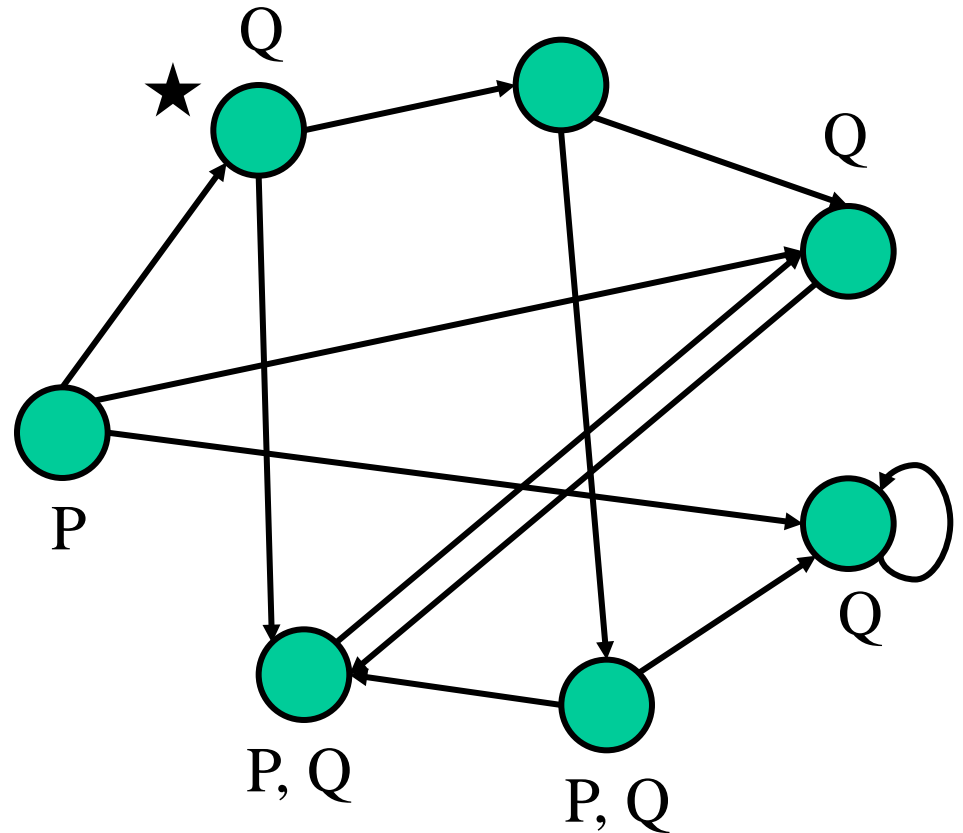
Which holds at ★?

1. **EF** ( $P \wedge \neg Q$ )
2. **EF AG** P
3. **EG** Q
4. **EG**  $\neg Q$



# Does $\mathbf{AF AG Q}$ hold at $\star$ ?

1. Yes
2. No



# Computation of State Sets

- Global model checking
- Let  $[[\varphi]] = \{ s \in S \mid s \models \varphi \}$ 
  - $[[P]] = \{ s \in S \mid P \in L(s) \}$
  - $[[\neg\varphi]] = S - [[\varphi]]$
  - $[[\varphi \wedge \psi]] = [[\varphi]] \cap [[\psi]]$
  - $[[\varphi \vee \psi]] = [[\varphi]] \cup [[\psi]]$
  - $[[\Box\varphi]] = \{ s \in S \mid \text{for any } t \in S, \\ \text{if } R(s,t) \text{ then } t \in [[\varphi]] \}$
  - $[[\Diamond\varphi]] = \{ s \in S \mid \text{there exists } t \in S \text{ such that} \\ R(s,t) \text{ and } t \in [[\varphi]] \}$



# Computation of State Sets

- Computation of  $[[\mathbf{EF}\varphi]]$

$X := \emptyset$

**loop**

$Y := [[\varphi]] \cup$

$\{ s \in S \mid \text{there exists } t \in S \text{ s.t. } R(s,t) \text{ and } t \in X \}$

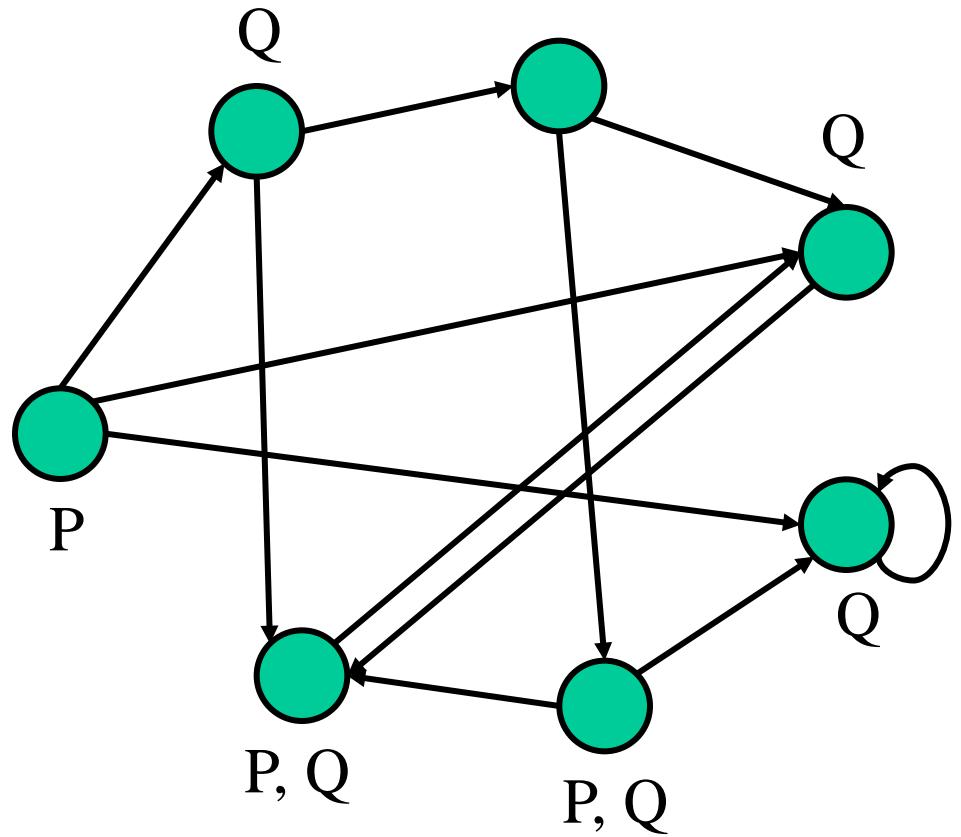
**if**  $Y == X$  **then break**

$X := Y$

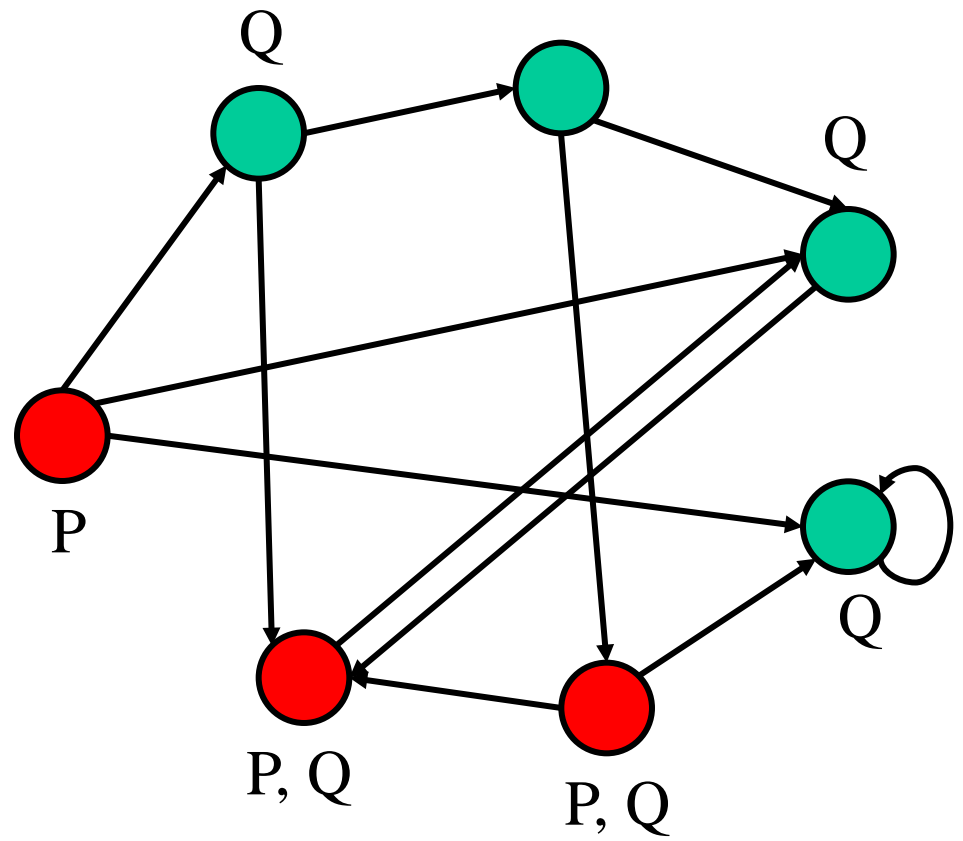
**end**

- Computing the minimal  $X$  such that  $X = f(X)$ , where  $f(X)$  denotes the R.H.S. of  $Y :=$

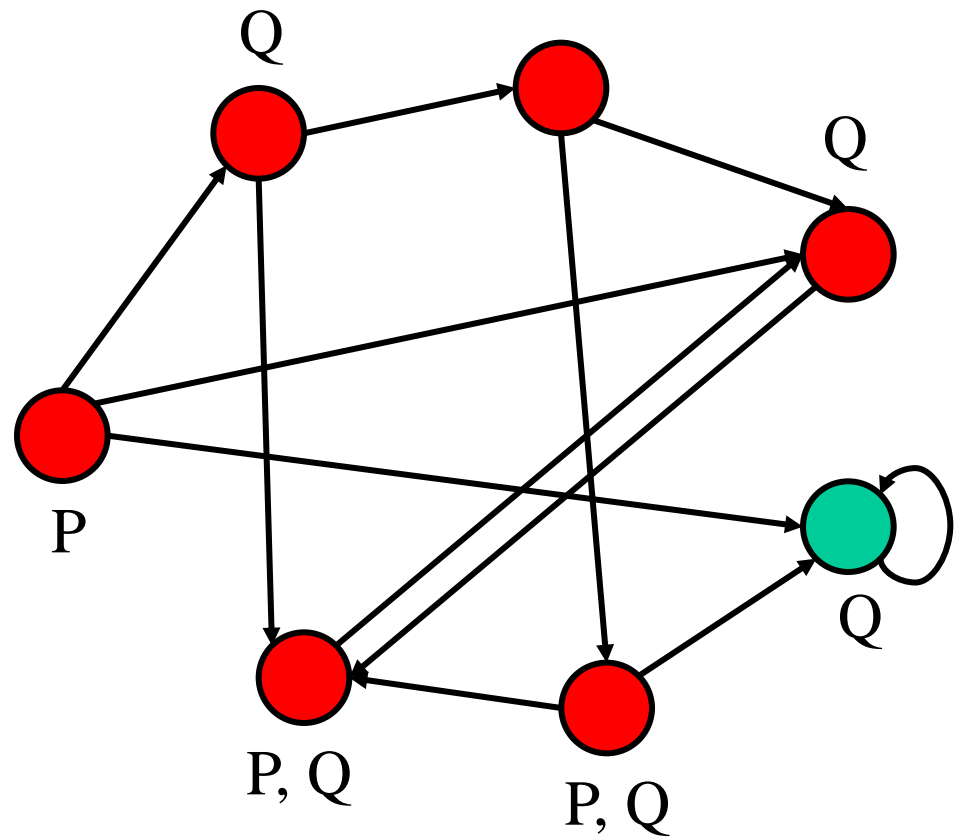
**EF P**



**EF P**



**EF P**



# Computation of State Sets

- Computation of  $[[\mathbf{EG}\varphi]]$

$X := S$

**loop**

$Y := [[\varphi]] \cap$

$\{ s \in S \mid \text{there exists } t \in S \text{ s.t. } R(s,t) \text{ and } t \in X \}$

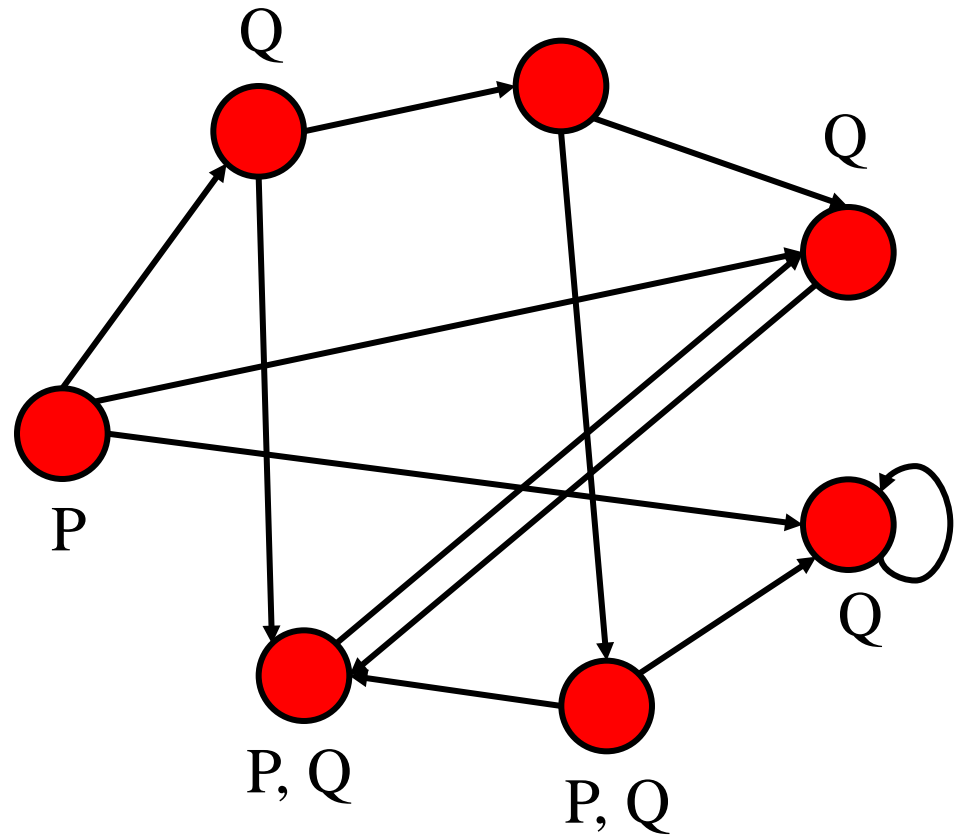
**if**  $Y == X$  **then break**

$X := Y$

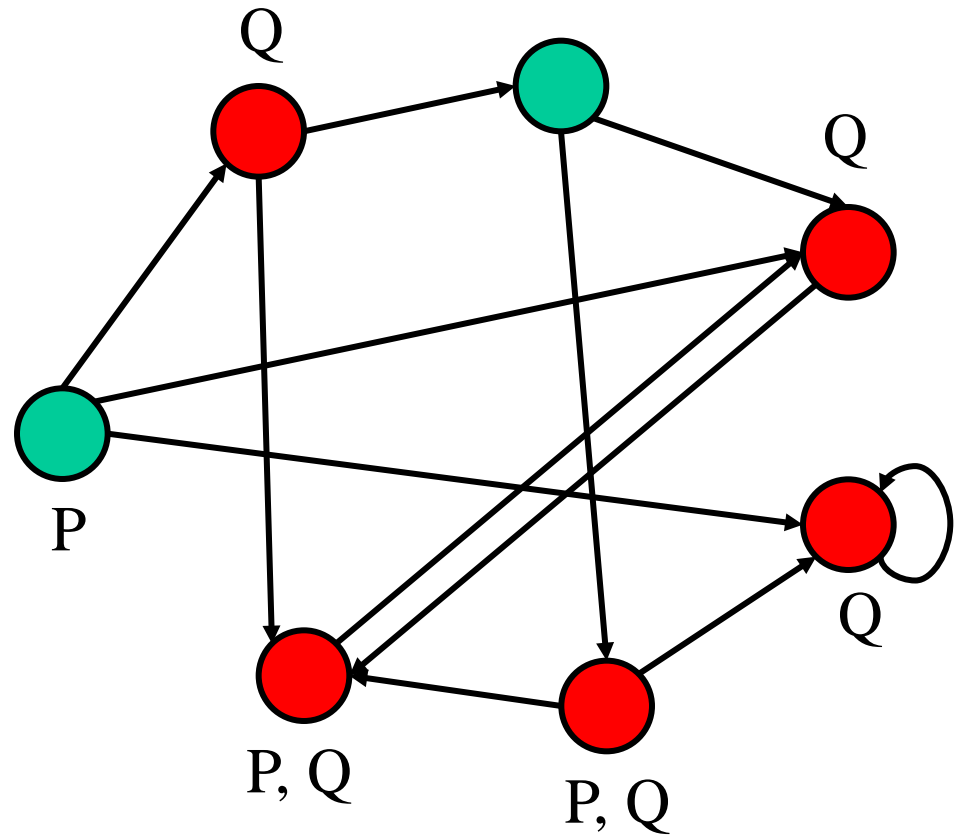
**end**

- Computing the maximal  $X$  such that  $X = f(X)$ , where  $f(X)$  denotes the R.H.S. of  $Y :=$

# EG Q



# EG Q



# Computation of State Sets

- Computation of  $[[\mathbf{AG}\phi]]$

$X := S$

**loop**

$Y := [[\phi]] \cap$

$\{ s \in S \mid \text{for any } t \in S, \text{ if } R(s,t) \text{ then } t \in X \}$

**if**  $Y == X$  **then break**

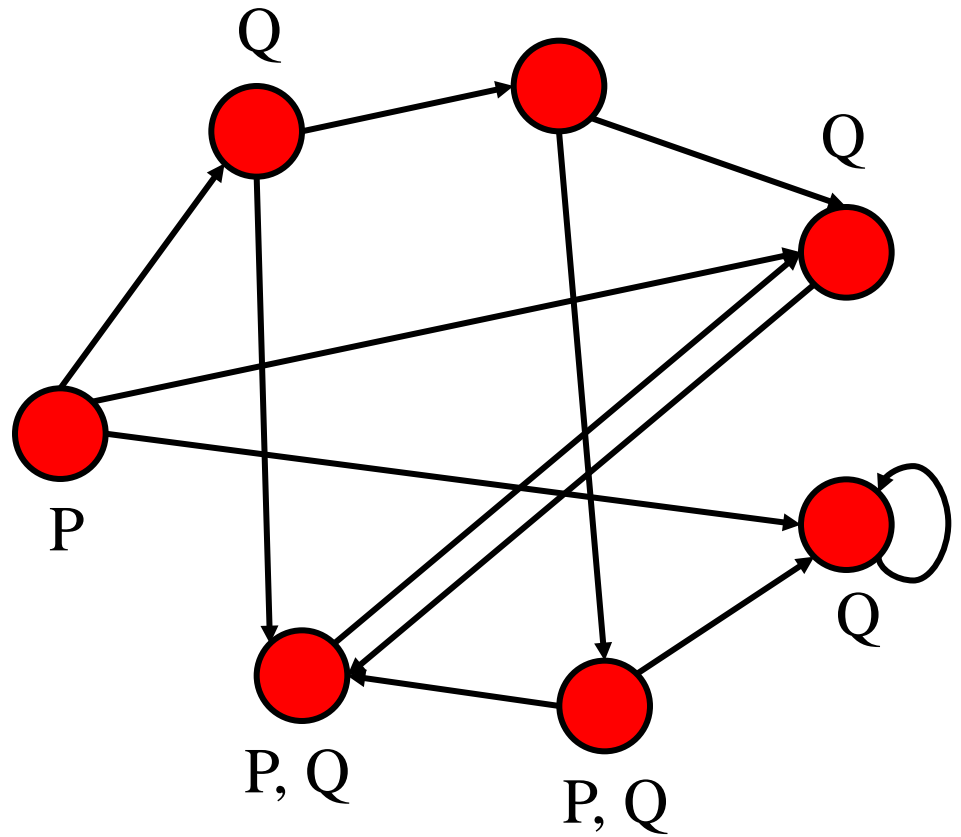
$X := Y$

**end**

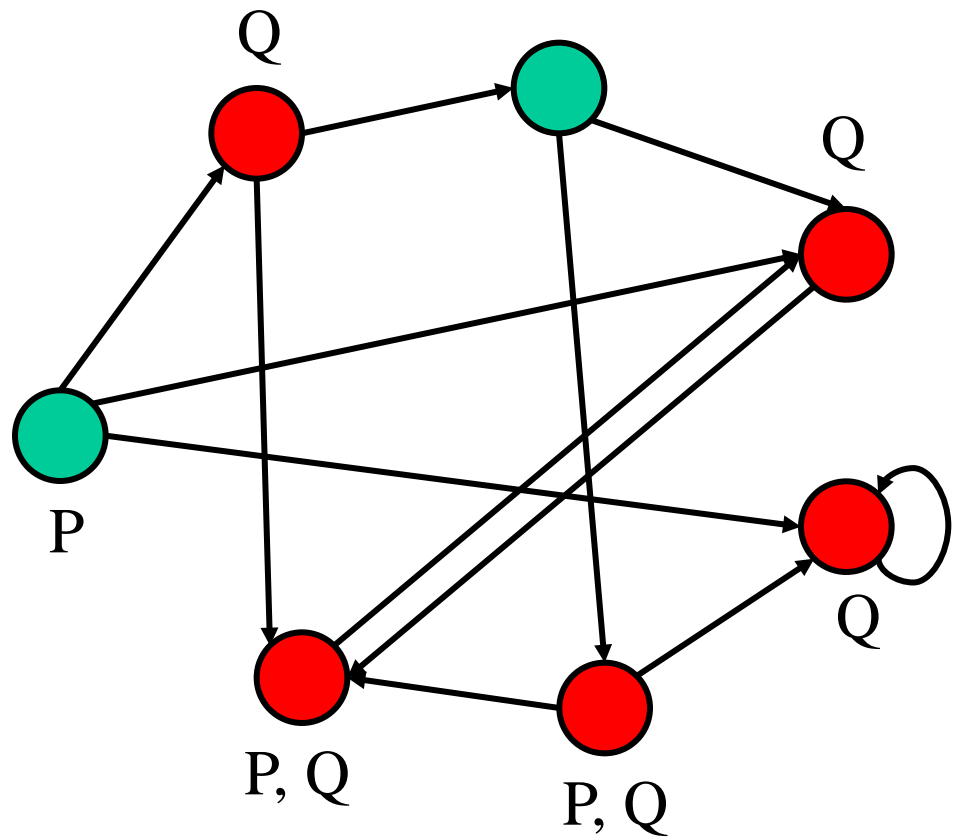
- Computing the maximal  $X$  such that  $X = f(X)$ , where  $f(X)$  denotes the R.H.S. of  $Y :=$



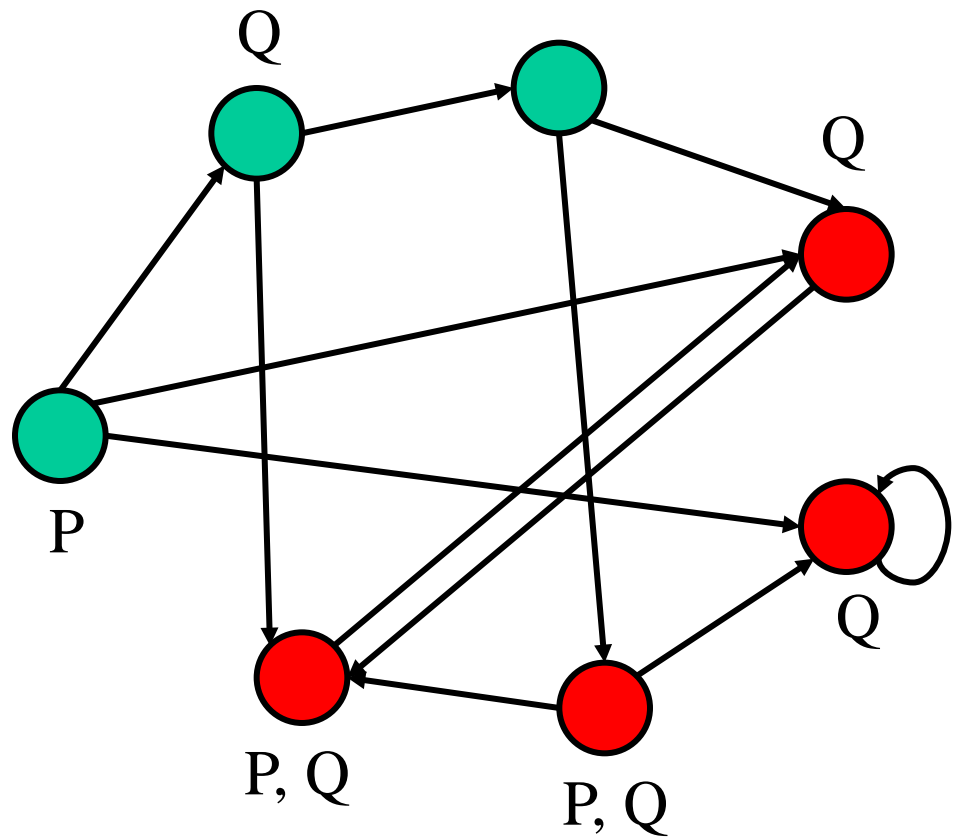
**AG Q**



**AG Q**



**AG Q**



# Computation of State Sets

- Computation of  $[[\mathbf{AF}\phi]]$

$X := \emptyset$

**loop**

$Y := [[\phi]] \cup$

$\{ s \in S \mid \text{for any } t \in S, \text{ if } R(s,t) \text{ then } t \in X \}$

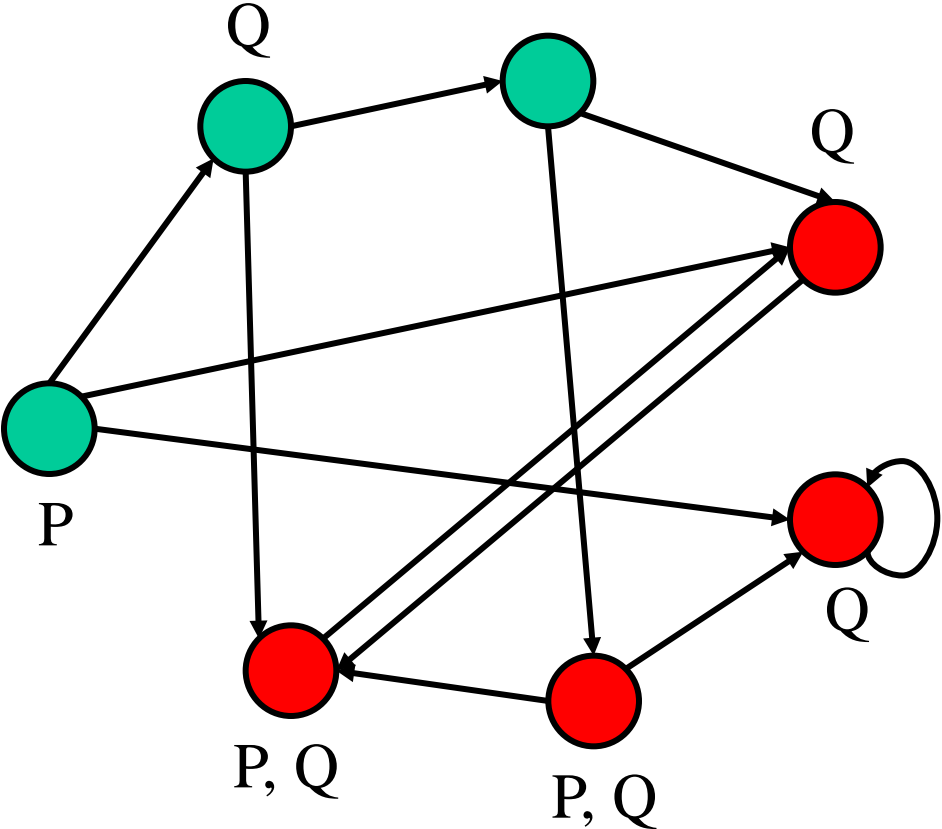
**if**  $Y == X$  **then break**

$X := Y$

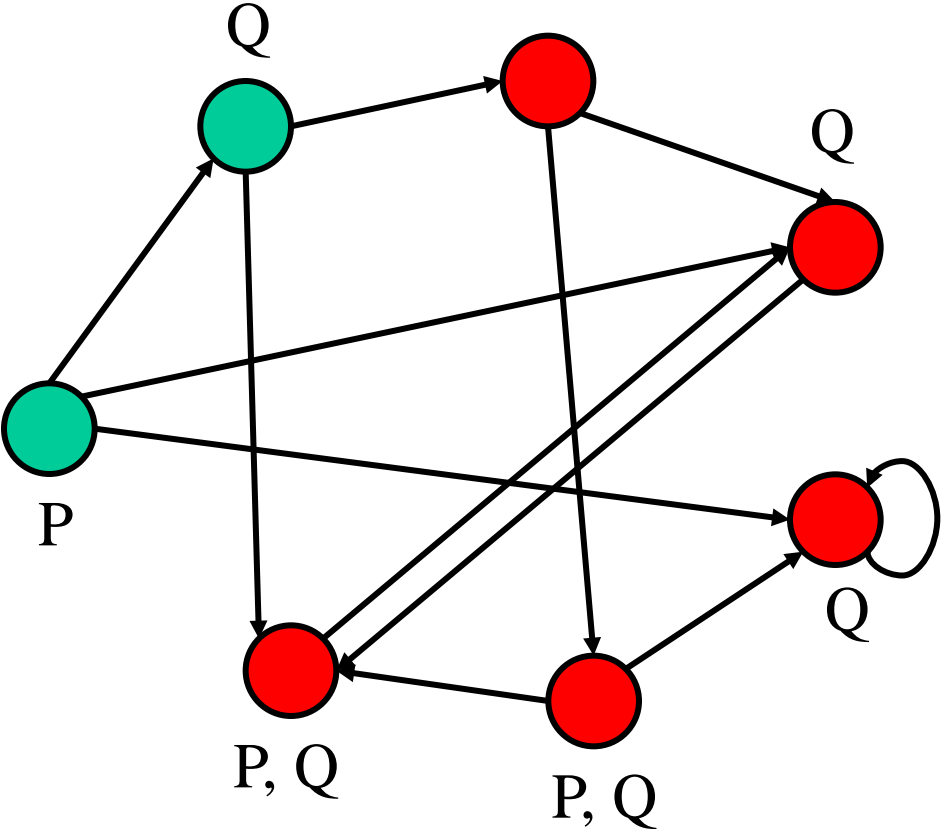
**end**

- Computing the minimal  $X$  such that  $X = f(X)$ , where  $f(X)$  denotes the R.H.S. of  $Y :=$

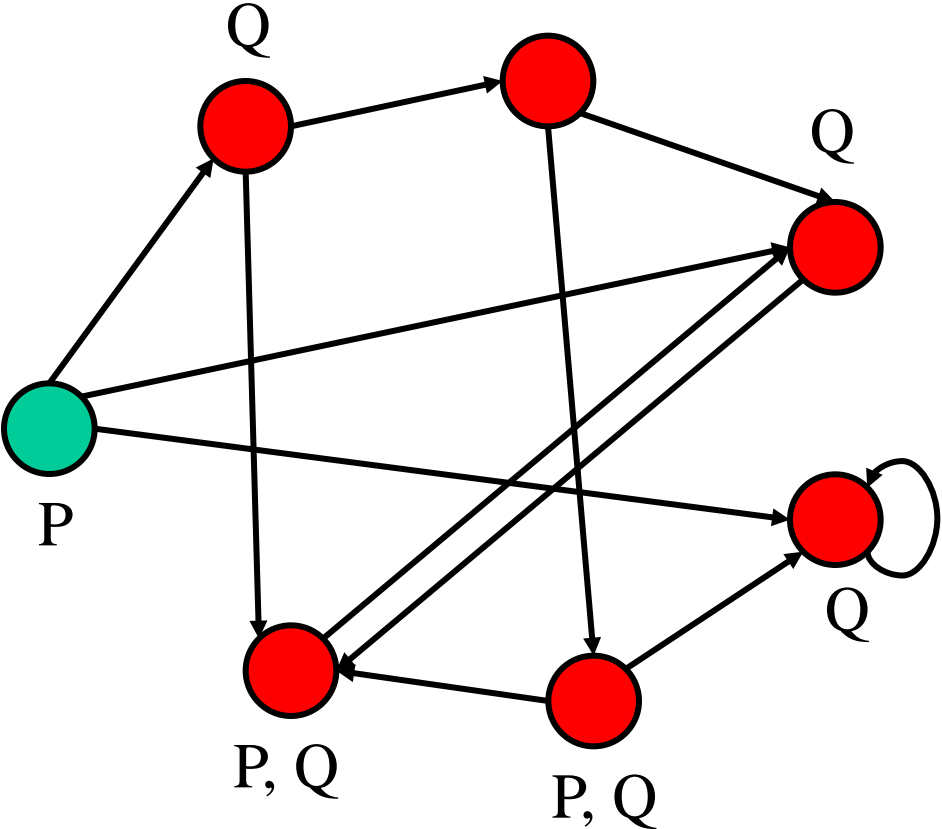
**AF AG Q**



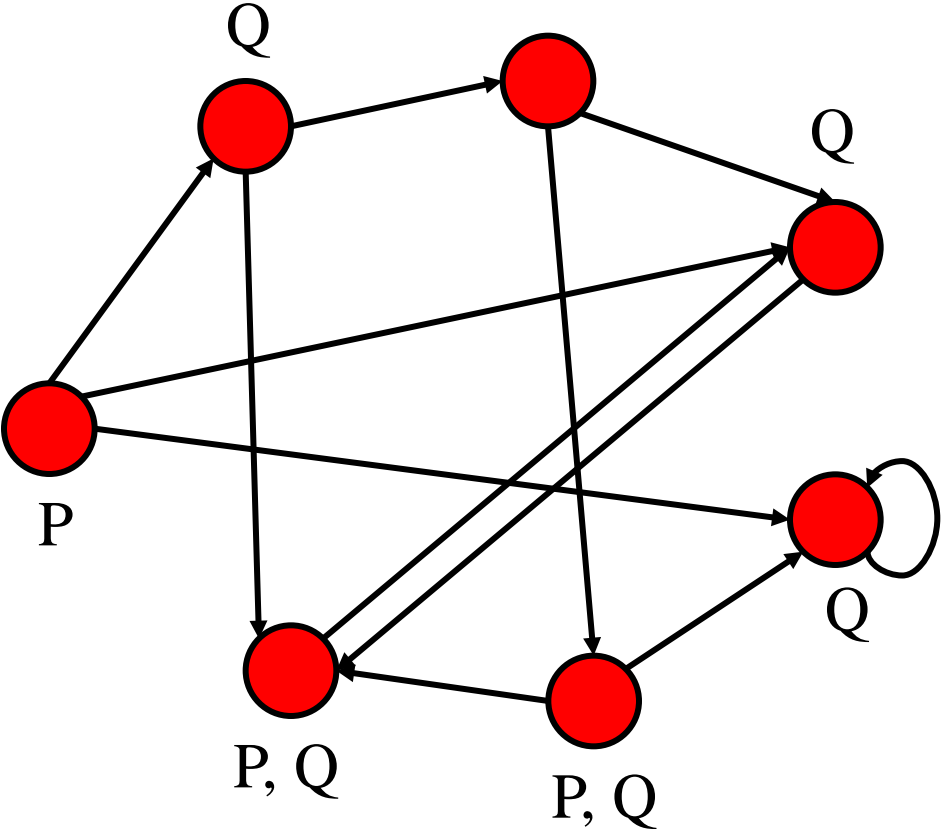
**AF AG Q**



**AF AG Q**



# AF AG Q





# Summary

- $[[\mathbf{EF}\varphi]] = [[\varphi \vee \Diamond \mathbf{EF}\varphi]]$ 
  - minimal
- $[[\mathbf{EG}\varphi]] = [[\varphi \wedge \Diamond \mathbf{EG}\varphi]]$ 
  - maximal
- $[[\mathbf{AG}\varphi]] = [[\varphi \wedge \Box \mathbf{AG}\varphi]]$ 
  - maximal
- $[[\mathbf{AF}\varphi]] = [[\varphi \vee \Box \mathbf{AF}\varphi]]$ 
  - minimal

# Summary

- $[[\mathbf{EF}\varphi]] = [[\varphi \vee \diamond \mathbf{EF}\varphi]]$   
– minimal
- $[[\mathbf{EG}\varphi]] = [[\varphi \wedge \diamond \mathbf{EG}\varphi]]$   
– maximal
- $[[\mathbf{AG}\varphi]] = [[\varphi \wedge \square \mathbf{AG}\varphi]]$   
– maximal
- $[[\mathbf{AF}\varphi]] = [[\varphi \vee \square \mathbf{AF}\varphi]]$   
– minimal

# Summary

- $[[\mathbf{EF}\varphi]] = [[\varphi \vee \diamond \mathbf{EF}\varphi]]$ 
  - minimal
- $[[\mathbf{EG}\varphi]] = [[\varphi \wedge \diamond \mathbf{EG}\varphi]]$ 
  - maximal
- $[[\mathbf{AG}\varphi]] = [[\varphi \wedge \square \mathbf{AG}\varphi]]$ 
  - maximal
- $[[\mathbf{AF}\varphi]] = [[\varphi \vee \square \mathbf{AF}\varphi]]$ 
  - minimal

# Modal $\mu$ -calculus

- Inductive definitions on propositional var.

$$X = \varphi \vee \Diamond X$$

- Minimal fixed point or maximal fixed point?

- In the case of a minimal fixed point

$$\mu X. \varphi \vee \Diamond X \text{ --- coincides with } \mathbf{EF}\varphi$$

- In the case of a maximal fixed point

true everywhere

- Example of a maximal fixed point

$$\nu X. \varphi \wedge \Box X \text{ --- coincides with } \mathbf{AG}\varphi$$

Which denotes  $\mathbf{AF}\varphi$ ?

1.  $\mu X. \varphi \wedge \Diamond X$

2.  $\mu X. \varphi \vee \Diamond X$

3.  $\nu X. \varphi \wedge \Diamond X$

4.  $\nu X. \varphi \vee \Diamond X$

5.  $\mu X. \varphi \wedge \Box X$

6.  $\mu X. \varphi \vee \Box X$

7.  $\nu X. \varphi \wedge \Box X$

8.  $\nu X. \varphi \vee \Box X$

# Symbolic Model Checking and BDD

# Representation of States

- States are often represented by bit vectors

# Example: Peterson's Algorithm

```
0: flags[me] = true;  
1: turn = you;  
2: if (flags[you] != true) goto 4;  
3: if (turn != you) goto 4; else goto 2;  
4: critical section;  
5: flags[me] = false;  
6: either goto 6 or goto 0;
```

- state: (pc0, pc1, flags[0], flags[1], turn)

pc0, pc1: 0..6

flags[0], flags[1]: {true, false}

turn: {0, 1}

Each pc can be represented by three bits

These are already bits



# Representation of States

- States are often represented by bit vectors
- So, the set  $S$  of states is (a subset of)  $\{0,1\}^n$
- Each state is regarded as an assignment mapping  $n$  boolean variables  $x_1, \dots, x_n$  to 0 or 1
- In the case of Peterson's algorithm
  - pc00, pc01, pc02 --- pc0
  - pc10, pc11, pc12 --- pc1
  - flags0, flags1 --- flags[0], flags[1]
  - turn

# Representation of State Sets

- A set of states can be represented by a boolean formula over  $x_1, \dots, x_n$
- For example,  $x_1 \wedge \neg x_2$  represents the set of states of the form  $\langle 1, 0, \dots \rangle \in \{0, 1\}^n$
- Remember that  $[[\varphi]]$  denotes a set of states defined as  $\{ s \in S \mid s \models \varphi \}$
- So,  $[[\varphi]]$  is represented by a boolean formula over  $x_1, \dots, x_n$
- For example
  - $pc0=4 \text{ --- } pc00 \wedge \neg pc01 \wedge \neg pc02$

# Representation of Transitions

- The transition relation  $R$  is represented by a boolean formula over  $x_1, \dots, x_n$  and  $x'_1, \dots, x'_n$ 
  - $x_1, \dots, x_n$  --- state before transition
  - $x'_1, \dots, x'_n$  --- state after transition
- The formula is denoted as

$$R(x_1, \dots, x_n, x'_1, \dots, x'_n)$$

$$R(\text{pc}00, \text{pc}01, \text{pc}02, \text{pc}10, \text{pc}11, \text{pc}12, \text{flags}0, \text{flags}1, \text{turn},$$

$$\text{pc}00', \text{pc}01', \text{pc}02', \text{pc}10', \text{pc}11', \text{pc}12',$$

$$\text{flags}0', \text{flags}1', \text{turn}') := \dots \vee$$

$$\neg \text{pc}00 \wedge \text{pc}01 \wedge \neg \text{pc}02 \wedge \neg \text{flags}1 \wedge$$

$$\text{pc}00' \wedge \neg \text{pc}01' \wedge \neg \text{pc}02' \wedge$$

$$(\text{pc}10 = \text{pc}10') \wedge (\text{pc}11 = \text{pc}11') \wedge (\text{pc}12 = \text{pc}12') \wedge$$

$$(\text{flags}0 = \text{flags}0') \wedge (\text{flags}1 = \text{flags}1') \wedge (\text{turn} = \text{turn}')$$

$\vee \dots$

0: flags[me] = true;

1: turn = you;

2: if (flags[you] != true) goto 4;

3: if (turn != you) goto 4; else goto 2;

4: critical section;

5: flags[me] = false;

6: either goto 6 or goto 0;

# Which is true?

1.  $R(0,1,0,0,1,0,1,1,1,0,1,0,1,0,0,1,1,1)$
2.  $R(0,1,0,0,1,0,1,1,1,0,1,1,0,1,0,1,1,1)$
3.  $R(0,1,0,0,1,0,1,1,1,1,0,0,0,1,0,1,1,1)$

# Semantics of Propositional Symbols

- $L : S \rightarrow 2^{\mathbf{Atom}}$

- It suffices to define

$$[[P]] = \{ s \mid P \in L(s) \}$$

for each  $P \in \mathbf{Atom}$

- Since  $\{ s \mid P \in L(s) \}$  is a state set, it can be defined by a formula  $P(x_1, \dots, x_n)$  over  $x_1, \dots, x_n$

# Summing Up...

- $K = \langle S, R, L \rangle$  (Kripke structure)

- $S = \{0,1\}^n$

- $R$  is represented by a boolean formula

$$R(x_1, \dots, x_n, x'_1, \dots, x'_n)$$

- $L : S \rightarrow 2^{\text{Atom}}$  is defined in terms of

$$P(x_1, \dots, x_n)$$

that represents

$$[[P]] = \{ s \mid P \in L(s) \} \text{ for each } P$$

# Computation of State Sets

- Computation of  $[[\mathbf{EF}\phi]]$

$X := \emptyset$

**loop**

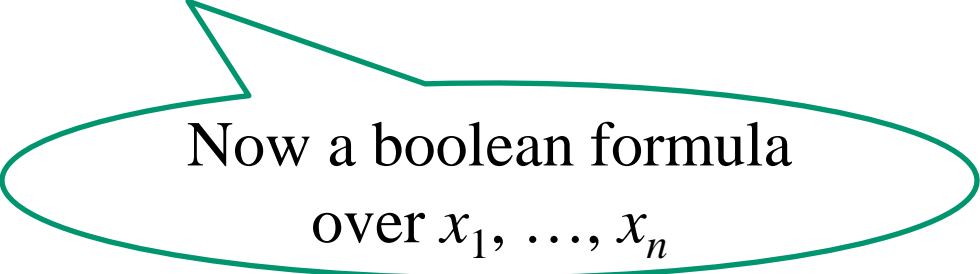
$Y := [[\phi]] \cup$

$\{ s \in S \mid \text{there exists } t \in S \text{ s.t. } R(s,t) \text{ and } t \in X \}$

**if**  $Y == X$  **then break**

$X := Y$

**end**



Now a boolean formula  
over  $x_1, \dots, x_n$

- Computing the minimal  $X$  such that  $X = f(X)$ , where  $f(X)$  denotes the R.H.S. of  $Y :=$



# Computation of State Sets

- Computation of  $[[\mathbf{EF}\varphi]](x_1, \dots, x_n)$

$X(x_1, \dots, x_n) := \mathbf{false}$

Begins with **false**

**loop**

$Y(x_1, \dots, x_n) := [[\varphi]](x_1, \dots, x_n) \vee$

Disjunction

$\{ s \in S \mid \text{there exists } t \in S \text{ s.t. } R(s, t) \text{ and } t \in X \}$

**if**  $Y == X$  **then break**

$X := Y$

**end**

Equality between  
boolean formulas

# Quantified Boolean Formulas

- Quantifiers are allowed in boolean formulas

$$\exists x f := f|_{x \leftarrow 0} \vee f|_{x \leftarrow 1}$$

$$\forall x f := f|_{x \leftarrow 0} \wedge f|_{x \leftarrow 1}$$

- $\{ s \in S \mid \text{there exists } t \in S \text{ s.t. } R(s, t) \text{ and } t \in X \}$

is represented by

$$\exists x'_1 \dots \exists x'_n (R(x_1, \dots, x_n, x'_1, \dots, x'_n) \wedge X(x'_1, \dots, x'_n))$$

$\exists x((\neg x \wedge y) \vee (x \wedge \neg z))$  is equivalent to?

1.  $\neg x \wedge y$

2.  $x \wedge \neg z$

3.  $y \wedge \neg z$

4.  $y \vee \neg z$

5.  $x$

6.  $\neg x$

7.  $y$

8.  $\neg z$

# Computation of State Sets

- Computation of  $[[\mathbf{EF}\varphi]](x_1, \dots, x_n)$

$X(x_1, \dots, x_n) := \mathbf{false}$

**loop**

$Y(x_1, \dots, x_n) := [[\varphi]](x_1, \dots, x_n) \vee$

$\exists x'_1 \dots \exists x'_n (R(x_1, \dots, x_n, x'_1, \dots, x'_n) \wedge$   
 $X(x'_1, \dots, x'_n))$

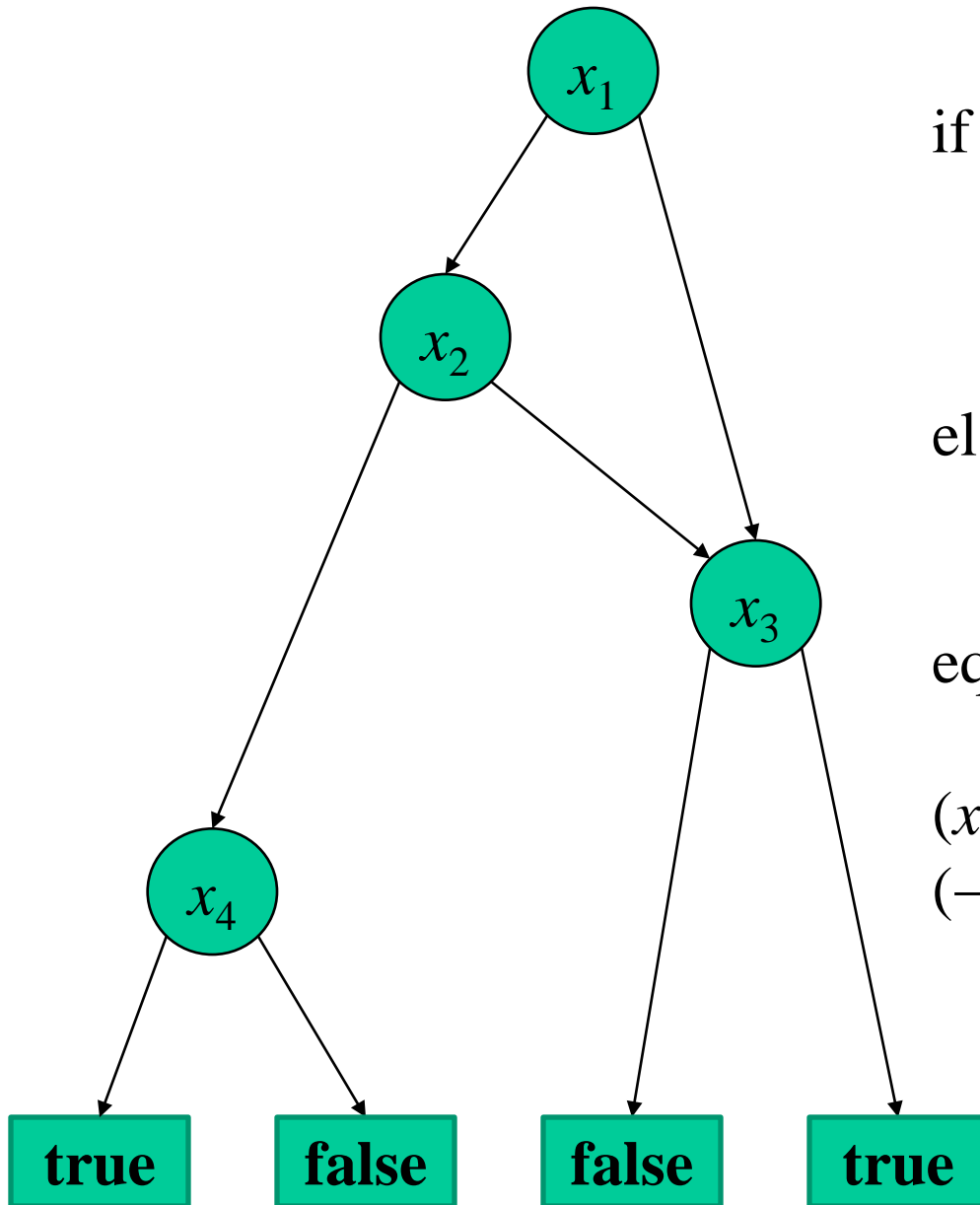
**if**  $Y == X$  **then break**

$X := Y$

**end**

# OBDD

- Ordered Binary Decision Diagram
  - Acyclic graph whose nodes are boolean variables
  - Each branch from a node represents “if variable then ... else ...”
  - Variables are ordered
  - Sub-graphs can be shared
- Compact representation and efficient computation of boolean formulas



if  $x_1$  then  
    if  $x_2$  then  
        if  $x_4$  then **true** else **false**  
        if  $x_3$  then **false** else **true**  
    else  
        if  $x_3$  then **false** else **true**

equivalent to

$$(x_1 \wedge (x_2 \wedge x_4) \vee (\neg x_2 \wedge \neg x_3)) \vee (\neg x_1 \wedge \neg x_3)$$

# Advantages of OBDD

- Equivalent formulas are represented by a unique OBDD (with a fixed order of variables)
- Operations on boolean formulas, including quantification, can be implemented as manipulations of OBDD
  - Quite efficient in general, but
  - Efficiency of operations (in particular, quantification) greatly depends on the order of variables

# SMV

- Symbolic Model Checker
- Originally developed for hardware verification
- Currently applied to various systems including software, protocols, etc.
- NuSMV is a reimplementation and extension of SMV
- <http://nusmv.fbk.eu/>



**LTL**

# Linear-time Temporal Logic

- Formulas are interpreted with respect to a path on a Kripke structure
- The concept of a path partially appears in CTL

$s \models \mathbf{EG}\varphi$  **iff**

there exists a path starting from  $s$  such that  
any state  $t$  on the path satisfies  $t \models \varphi$

- But in CTL formulas are interpreted at each state

# Kripke Structure

- $K = \langle S, R, L \rangle$

$S$ : set of states (may be infinite)

$R$ : transition relation between states

$$R \subseteq S \times S$$

$L$ : map from states to sets of prop. symbols

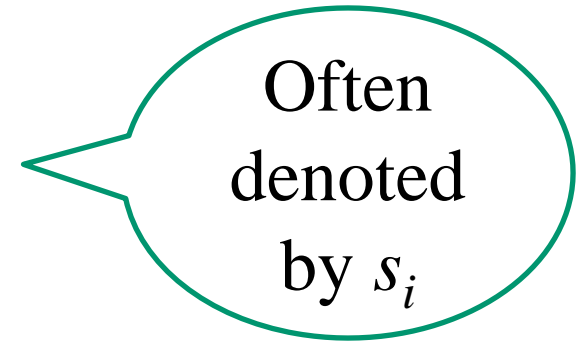
$L(s)$  denotes the set of prop. symbols  
that hold at state  $s \in S$

# Infinite Sequence of States --- Execution Path

- $\pi = \pi_0, \pi_1, \pi_2, \dots$

$$\pi_i \in S \quad (\forall i \geq 0)$$

$$R(\pi_i, \pi_{i+1}) \quad (\forall i \geq 0)$$



- suffix

$$\pi^i = \pi_i, \pi_{i+1}, \pi_{i+2}, \dots$$

# Formula

$\varphi, \psi$	$::=$	$P$	prop. symbol
		$\neg\varphi$	negation
		$\varphi \wedge \psi$	cojunction
		$\varphi \vee \psi$	disjunction
		$\bigcirc\varphi$	$(\mathbf{X}\varphi)$
		$\square\varphi$	$(\mathbf{G}\varphi)$
		$\diamond\varphi$	$(\mathbf{F}\varphi)$

**until** is not  
considered  
here

# Semantics

$\pi \models P$  **iff**  $P \in L(\pi_0)$

$\pi \models \neg\varphi$  **iff** **not**  $\pi \models \varphi$

$\pi \models \varphi \wedge \psi$  **iff**  $\pi \models \varphi$  **and**  $\pi \models \psi$

$\pi \models \varphi \vee \psi$  **iff**  $\pi \models \varphi$  **or**  $\pi \models \psi$

$\pi \models \bigcirc\varphi$  **iff**  $\pi^1 \models \varphi$

$\pi \models \Box\varphi$  **iff**  $\pi^i \models \varphi$  **for any**  $i \geq 0$

$\pi \models \Diamond\varphi$  **iff**  $\pi^i \models \varphi$  **for some**  $i \geq 0$

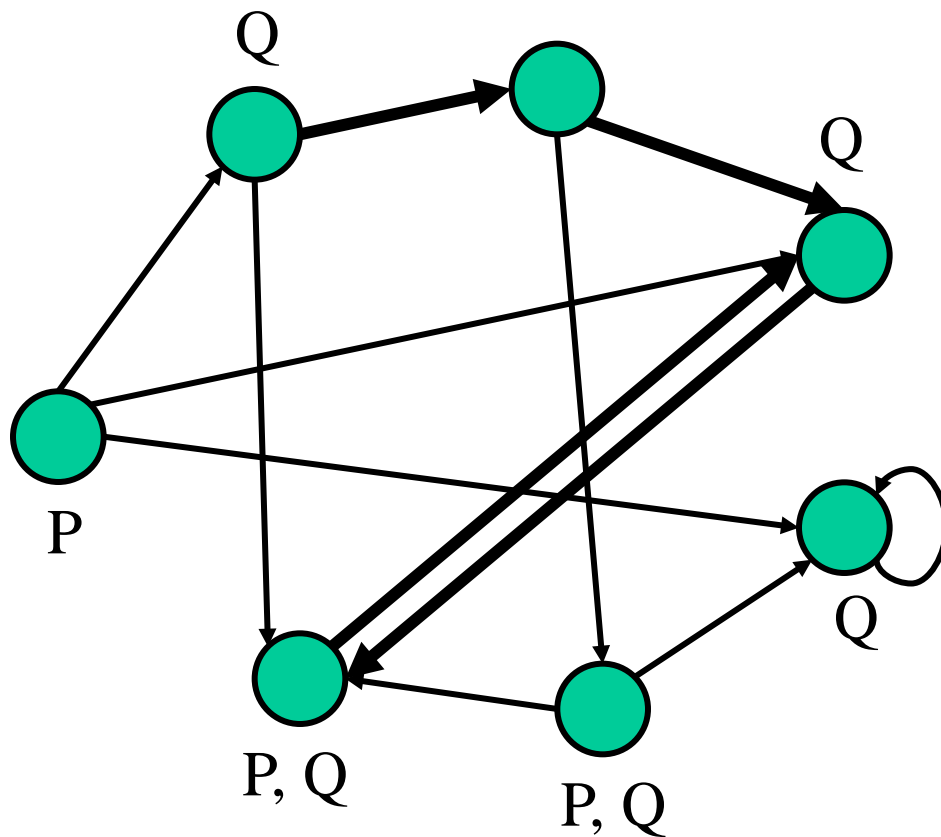
$\varphi$  holds in  $\pi$

$\pi \models \varphi$  ---  $\pi$  is a model of  $\varphi$

The semantics of  $\Box$  and  $\Diamond$  is different from that of CTL

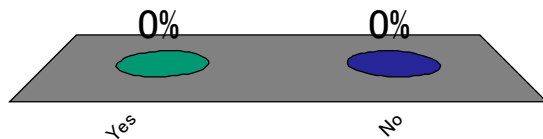
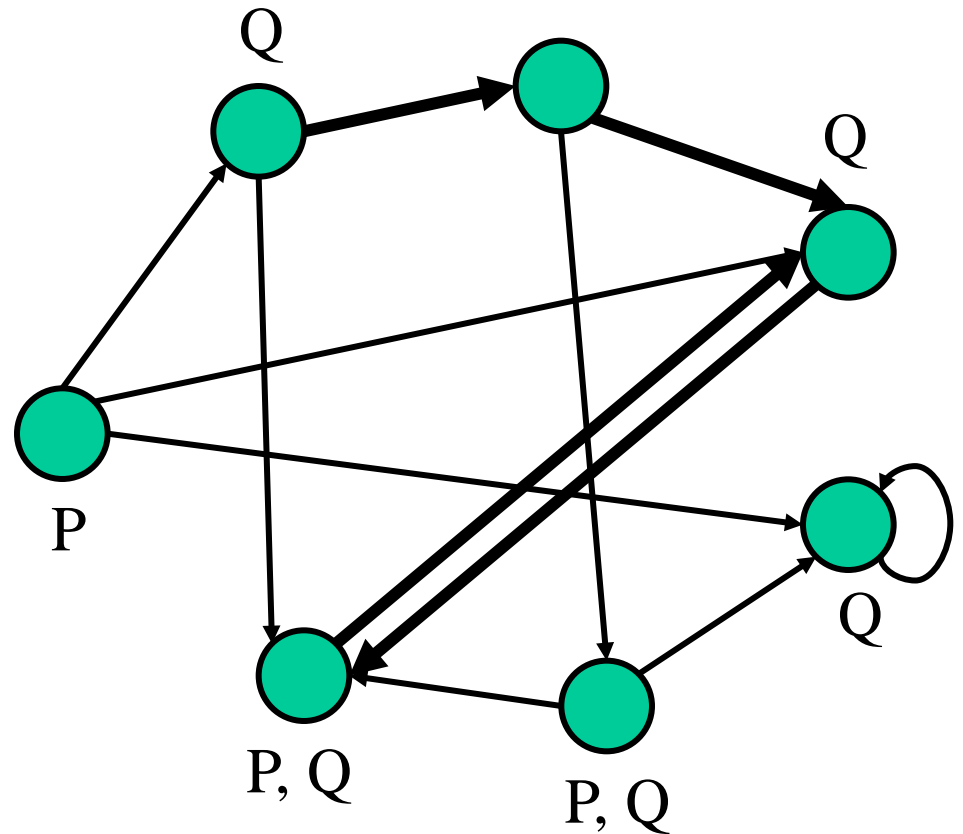
# Which formula holds in this path?

1.  $\Box P$
2.  $\Box \neg P$
3.  $\Diamond (P \wedge Q)$
4.  $\Diamond (P \wedge \neg Q)$



Does  $\diamond \square Q$  hold?

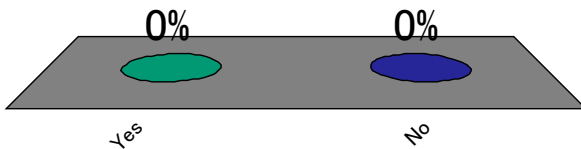
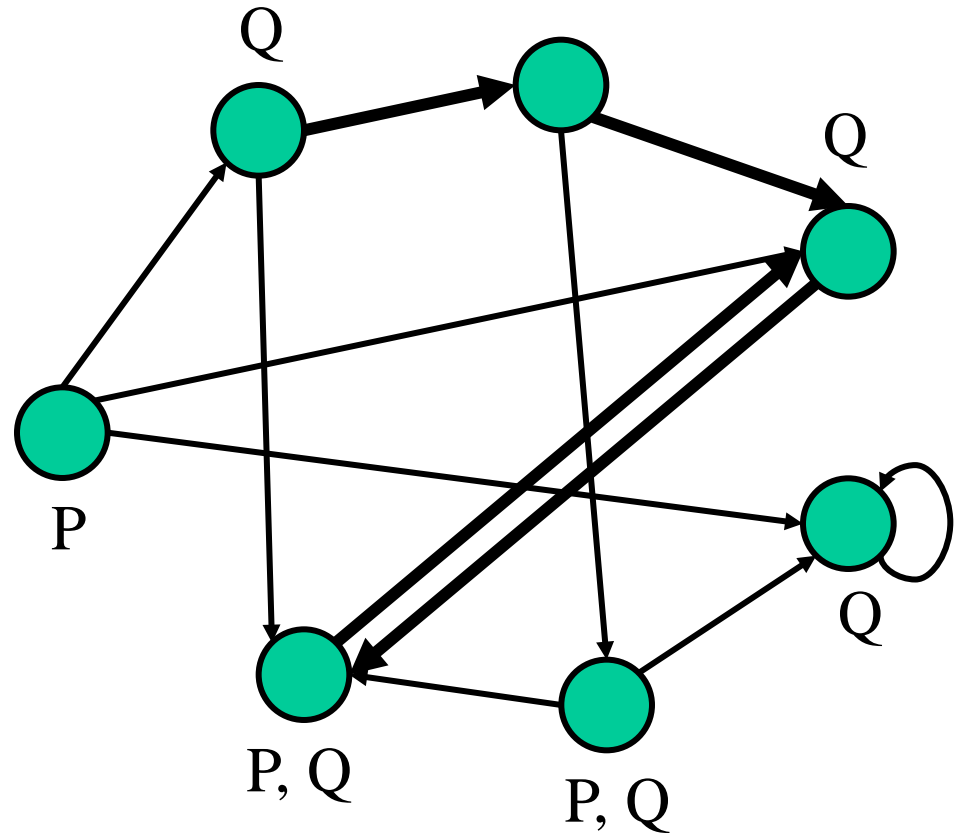
1. Yes
2. No





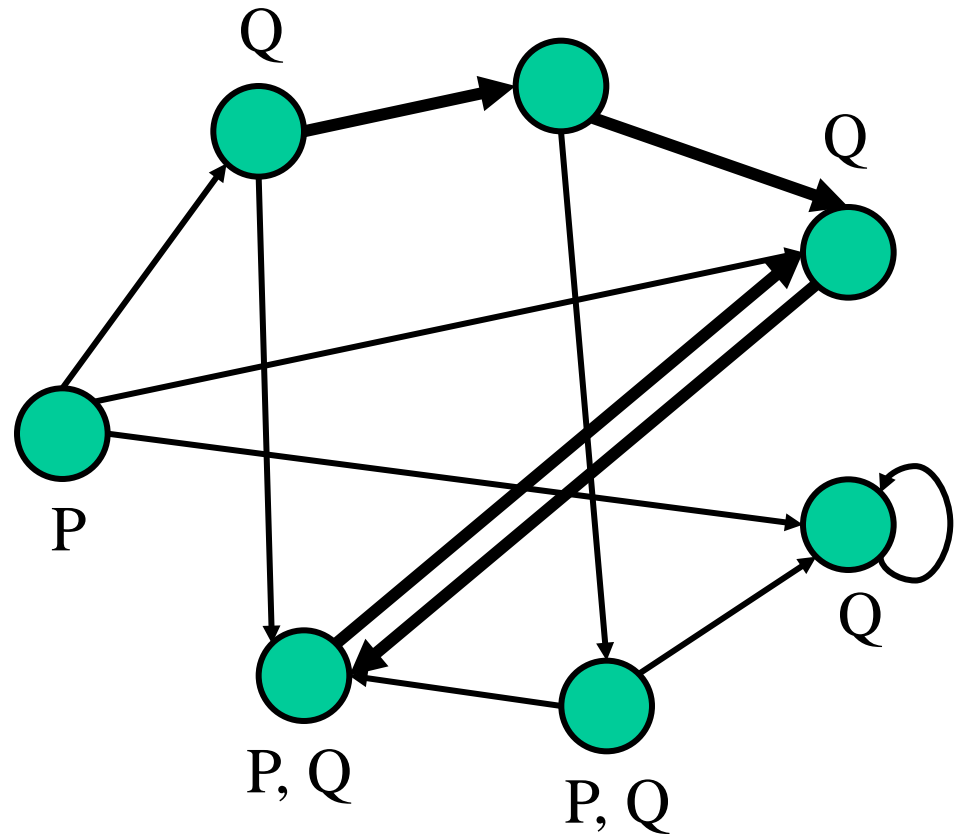
Does  $\diamond \square P$  hold?

1. Yes
2. No



Does  $\square \diamond P$  hold?

1. Yes
2. No



$$\Box \Diamond P$$

- $\pi \models \Box \Diamond P$  implies  $\pi \models \Diamond P$ ,  
so there exists  $i$  such that  $\pi^i \models P$
- $\pi \models \Box \Diamond P$  implies  $\pi^{i+1} \models \Diamond P$ ,  
so there exists  $j > i$  such that  $\pi^j \models P$
- Consequently, there exist an infinite number  
of  $i$  such that  $\pi^i \models P$
- Conversely, if there exist an infinite number  
of  $i$  such that  $\pi^i \models P$ , then  $\pi \models \Box \Diamond P$  holds

# Expressing Fairness

- Let E denote that a certain process is executable, and let R denote that the process has been executed

- Unconditional fairness

$$\Box \Diamond R$$

- Weak fairness

$$\Box \Diamond (\neg E \vee R)$$

$$\Box \Diamond (E \supset R)$$

- Strong fairness

$$\neg \Box \Diamond E \vee \Box \Diamond R$$

$$\Box \Diamond E \supset \Box \Diamond R$$

Consider their negation


$\square$  and  $\diamond$  and  $\neg$

$$\pi \models \neg \square \varphi \quad \mathbf{iff} \quad \pi \models \diamond \neg \varphi$$

$$\pi \models \neg \diamond \varphi \quad \mathbf{iff} \quad \pi \models \square \neg \varphi$$

$$\pi \models \neg \bigcirc \varphi \quad \mathbf{iff} \quad \pi \models \bigcirc \neg \varphi$$

# Model Checking in LTL

- Given a formula  $\varphi_0$ , a Kripke structure  $K$ , and its initial state  $s$ , does there exist a path  $\pi$  starting from  $s$  such that  $\pi \models \varphi_0$ ? 
  - If you want to verify that  $\varphi_0$  holds w.r.t. any path  $\pi$  starting from  $s$ , then you should negate  $\varphi_0$  and solve the model checking problem on  $\neg\varphi_0$
- Not so easy as in CTL
  - It is necessary to characterize paths that satisfy the given formula  $\varphi$
  - Use  $\omega$ -automata

# Working Example

- In the following, the formula  $\Box(a \supset \Diamond b)$  is used as a working example
- It is a typical formula expressing liveness
  - If a file is opened, it is eventually closed

# Formula (Negation Normal Form)

$$\begin{array}{l} \varphi, \psi \quad ::= \quad P \\ | \quad \neg P \\ | \quad \varphi \wedge \psi \\ | \quad \varphi \vee \psi \\ | \quad \bigcirc \varphi \\ | \quad \square \varphi \\ | \quad \diamond \varphi \end{array}$$

For any formula,  
there exists an equivalent  
normal form



Which is the normal form equivalent  
to  $\neg \Box(a \supset \Diamond b)$ ?

1.  $\Box(a \wedge \Diamond \neg b)$

2.  $\Box(a \vee \Diamond \neg b)$

3.  $\Diamond(a \wedge \Box \neg b)$

4.  $\Diamond(a \vee \Box \neg b)$

5.  $\Box(\neg a \wedge \Diamond b)$

6.  $\Box(\neg a \vee \Diamond b)$

7.  $\Diamond(\neg a \wedge \Box b)$

8.  $\Diamond(\neg a \vee \Box b)$

$\square$  and  $\diamond$  and  $\bigcirc$

$\pi \models \square\varphi$     **iff**     $\pi \models \varphi \wedge \bigcirc\square\varphi$

$\pi \models \diamond\varphi$     **iff**     $\pi \models \varphi \vee \bigcirc\diamond\varphi$

# $cl(\varphi_0)$ : the closure of $\varphi_0$

The minimum set of formulas satisfying

- $\varphi_0 \in cl(\varphi_0)$
- If  $\varphi_1 \wedge \varphi_2 \in cl(\varphi_0)$ , then  $\varphi_1 \in cl(\varphi_0)$  and  $\varphi_2 \in cl(\varphi_0)$
- If  $\varphi_1 \vee \varphi_2 \in cl(\varphi_0)$ , then  $\varphi_1 \in cl(\varphi_0)$  and  $\varphi_2 \in cl(\varphi_0)$
- If  $\bigcirc \varphi \in cl(\varphi_0)$ , then  $\varphi \in cl(\varphi_0)$
- If  $\square \varphi \in cl(\varphi_0)$ , then  $\varphi \wedge \bigcirc \square \varphi \in cl(\varphi_0)$
- If  $\diamond \varphi \in cl(\varphi_0)$ , then  $\varphi \vee \bigcirc \diamond \varphi \in cl(\varphi_0)$
- The condition “If  $\neg P \in cl(\varphi_0)$ , then  $P \in cl(\varphi_0)$ ” is not required

- $\varphi_0 = \Diamond(a \wedge \Box\neg b)$

- $cl(\varphi_0) :$

$$\Diamond(a \wedge \Box\neg b)$$

$$(a \wedge \Box\neg b) \vee \bigcirc\Diamond(a \wedge \Box\neg b)$$

$$a \wedge \Box\neg b$$

$$\bigcirc\Diamond(a \wedge \Box\neg b)$$

$$a$$

$$\Box\neg b$$

$$\neg b \wedge \bigcirc\Box\neg b$$

$$\neg b$$

$$\bigcirc\Box\neg b$$

# $\varphi_0$ -type

- $\Gamma \subseteq cl(\varphi_0)$
- If  $\varphi_1 \wedge \varphi_2 \in \Gamma$ , then  $\varphi_1 \in \Gamma$  and  $\varphi_2 \in \Gamma$
- If  $\varphi_1 \vee \varphi_2 \in \Gamma$ , then  $\varphi_1 \in \Gamma$  or  $\varphi_2 \in \Gamma$
- It is not the case that  $P \in \Gamma$  and  $\neg P \in \Gamma$
- If  $\Box \varphi \in \Gamma$ , then  $\varphi \wedge \bigcirc \Box \varphi \in \Gamma$
- If  $\Diamond \varphi \in \Gamma$ , then  $\varphi \vee \bigcirc \Diamond \varphi \in \Gamma$

# Selection of $\varphi_0$ -types and Transitions between $\varphi_0$ -types

- Select all the minimal  $\varphi_0$ -types that contain  $\varphi_0$
- If a  $\varphi_0$ -type  $\Gamma \subseteq cl(\varphi_0)$  is selected, select all the minimal  $\varphi_0$ -types  $\Gamma'$  that contain  $\{\varphi \in cl(\varphi_0) \mid \bigcirc \varphi \in \Gamma\}$
- Define a transition  $\Gamma \rightarrow \Gamma'$
- Repeat the above process

- $\Gamma_I :$   
 $\diamond(a \wedge \square \neg b)$

•  $\Gamma_I :$

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$



- $\Gamma_I :$ 
  - $\diamond(a \wedge \square \neg b)$
  - $(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$
  - $a \wedge \square \neg b$

- $\Gamma_{II} :$ 
  - $\diamond(a \wedge \square \neg b)$
  - $(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$
  - $\bigcirc \diamond(a \wedge \square \neg b)$

•  $\Gamma_I :$

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$a \wedge \square \neg b$$

$$a$$

$$\square \neg b$$

•  $\Gamma_{II} :$

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$\bigcirc \diamond(a \wedge \square \neg b)$$

- $\Gamma_I :$ 
  - $\diamond(a \wedge \square \neg b)$
  - $(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$
  - $a \wedge \square \neg b$
  - $a$
  - $\square \neg b$
  - $\neg b \wedge \bigcirc \square \neg b$

- $\Gamma_{II} :$ 
  - $\diamond(a \wedge \square \neg b)$
  - $(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$
  - $\bigcirc \diamond(a \wedge \square \neg b)$

•  $\Gamma_I :$

$$\diamond(a \wedge \square\neg b)$$

$$(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$$

$$a \wedge \square\neg b$$

$$a$$

$$\square\neg b$$

$$\neg b \wedge \bigcirc\square\neg b$$

$$\neg b$$

$$\bigcirc\square\neg b$$

•  $\Gamma_{II} :$

$$\diamond(a \wedge \square\neg b)$$

$$(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$$

$$\bigcirc\diamond(a \wedge \square\neg b)$$

# Which is in a successor of $\Gamma_I$ ?

•  $\Gamma_I$  :

$$\diamond(a \wedge \square\neg b)$$

$$(a \wedge \square\neg b) \vee \bigcirc \diamond(a \wedge \square\neg b)$$

$$a \wedge \square\neg b$$

$$a$$

$$\square\neg b$$

$$\neg b \wedge \bigcirc \square\neg b$$

$$\neg b$$

$$\bigcirc \square\neg b$$

1.  $a$

2.  $\square\neg b$

3.  $\bigcirc \square\neg b$

4.  $a \wedge \square\neg b$

5.  $\diamond(a \wedge \square\neg b)$

•  $\Gamma_I :$

$$\diamond(a \wedge \square\neg b)$$

$$(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$$

$$a \wedge \square\neg b$$

$$a$$

$$\square\neg b$$

$$\neg b \wedge \bigcirc\square\neg b$$

$$\neg b$$

$$\bigcirc\square\neg b$$

•  $\Gamma_{III} :$

$$\square\neg b$$

•  $\Gamma_{II} :$

$$\diamond(a \wedge \square\neg b)$$

$$(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$$

$$\bigcirc\diamond(a \wedge \square\neg b)$$

•  $\Gamma_I :$

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$a \wedge \square \neg b$$

$$a$$

$$\square \neg b$$

$$\neg b \wedge \bigcirc \square \neg b$$

$$\neg b$$

$$\bigcirc \square \neg b$$

•  $\Gamma_{II} :$

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$\bigcirc \diamond(a \wedge \square \neg b)$$

•  $\Gamma_{III} :$

$$\square \neg b$$

$$\neg b \wedge \bigcirc \square \neg b$$

- $\Gamma_I :$   
 $\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$

$$a \wedge \Box \neg b$$

$$a$$

$$\Box \neg b$$

$$\neg b \wedge \bigcirc \Box \neg b$$

$$\neg b$$

$$\bigcirc \Box \neg b$$

- $\Gamma_{II} :$   
 $\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $\bigcirc \diamond(a \wedge \Box \neg b)$

- $\Gamma_{III} :$   
 $\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$



# What are successors of I ?

I :

- $\diamond(a \wedge \Box \neg b)$
- $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$
- $a \wedge \Box \neg b$
- $a$
- $\Box \neg b$
- $\neg b \wedge \bigcirc \Box \neg b$
- $\neg b$
- $\bigcirc \Box \neg b$

1. none

2. I

3. I and II

4. I and III

II :

- $\diamond(a \wedge \Box \neg b)$
- $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$
- $\bigcirc \diamond(a \wedge \Box \neg b)$

III :

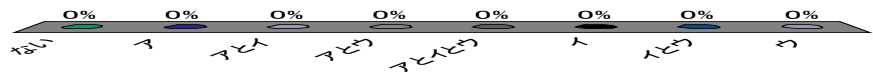
- $\Box \neg b$
- $\neg b \wedge \bigcirc \Box \neg b$
- $\neg b$
- $\bigcirc \Box \neg b$

5. I and II and III

6. II

7. II and III

8. III



# What are successors of $\Pi$ ?

1. none

2. I

3. I and  $\Pi$

4. I and  $\text{III}$

5. I and  $\Pi$  and  $\text{III}$

6.  $\Pi$

7.  $\Pi$  and  $\text{III}$

8.  $\text{III}$

I :

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$a \wedge \square \neg b$$

$$a$$

$$\square \neg b$$

$$\neg b \wedge \bigcirc \square \neg b$$

$$\neg b$$

$$\bigcirc \square \neg b$$

$\Pi$  :

$$\diamond(a \wedge \square \neg b)$$

$$(a \wedge \square \neg b) \vee \bigcirc \diamond(a \wedge \square \neg b)$$

$$\bigcirc \diamond(a \wedge \square \neg b)$$

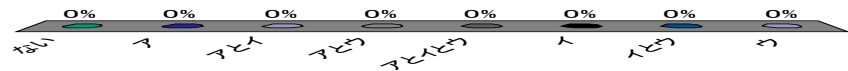
$\text{III}$  :

$$\square \neg b$$

$$\neg b \wedge \bigcirc \square \neg b$$

$$\neg b$$

$$\bigcirc \square \neg b$$



# What are successors of III?

I :

- $\diamond(a \wedge \Box \neg b)$
- $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$
- $a \wedge \Box \neg b$
- $a$
- $\Box \neg b$
- $\neg b \wedge \bigcirc \Box \neg b$
- $\neg b$
- $\bigcirc \Box \neg b$

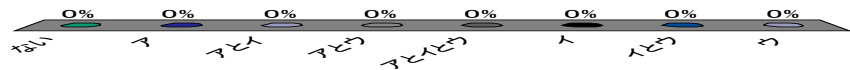
II :

- $\diamond(a \wedge \Box \neg b)$
- $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$
- $\bigcirc \diamond(a \wedge \Box \neg b)$

III :

- $\Box \neg b$
- $\neg b \wedge \bigcirc \Box \neg b$
- $\neg b$
- $\bigcirc \Box \neg b$

1. none
2. I
3. I and II
4. I and III
5. I and II and III
6. II
7. II and III
8. III



I :

$$\begin{aligned} & \diamond(a \wedge \Box\neg b) \\ & (a \wedge \Box\neg b) \vee \bigcirc \diamond(a \wedge \Box\neg b) \\ & a \wedge \Box\neg b \\ & a \\ & \Box\neg b \\ & \neg b \wedge \bigcirc \Box\neg b \\ & \neg b \\ & \bigcirc \Box\neg b \end{aligned}$$

II :

$$\begin{aligned} & \diamond(a \wedge \Box\neg b) \\ & (a \wedge \Box\neg b) \vee \bigcirc \diamond(a \wedge \Box\neg b) \\ & \bigcirc \diamond(a \wedge \Box\neg b) \end{aligned}$$

III :

$$\begin{aligned} & \Box\neg b \\ & \neg b \wedge \bigcirc \Box\neg b \\ & \neg b \\ & \bigcirc \Box\neg b \end{aligned}$$

I :

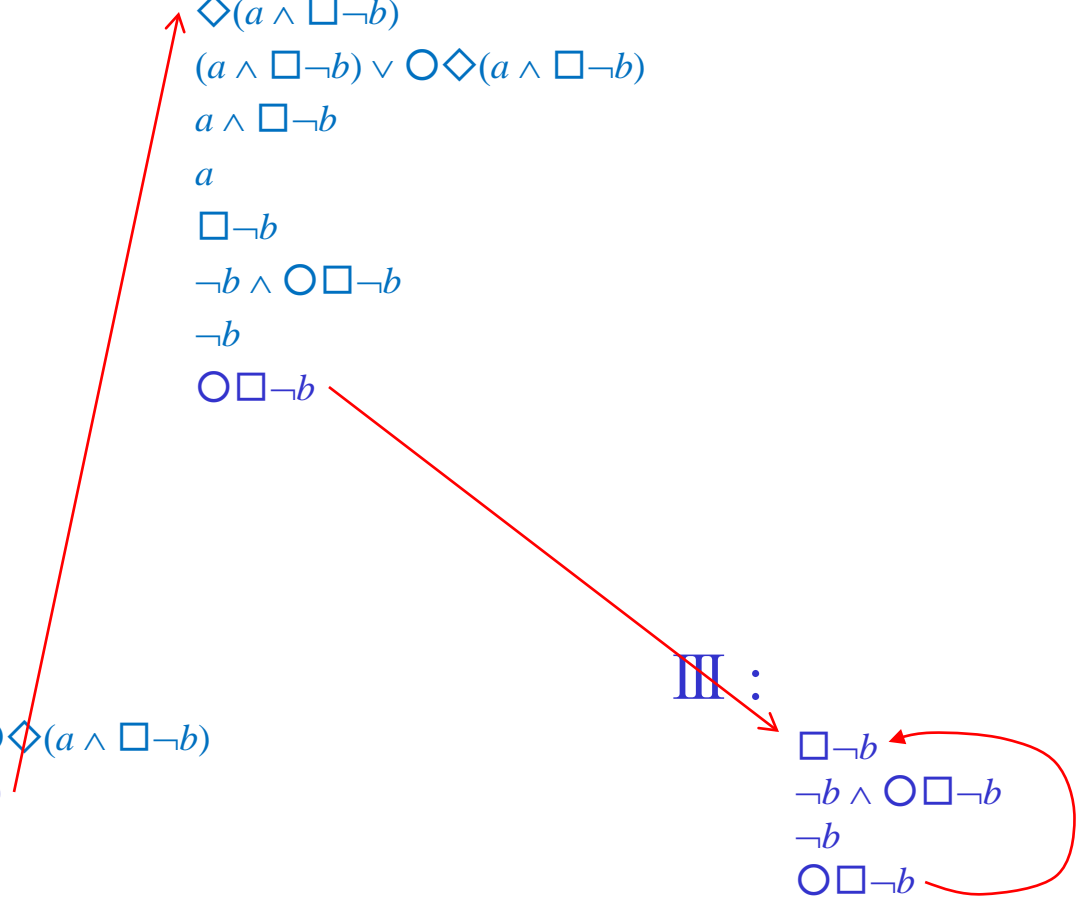
$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $a \wedge \Box \neg b$   
 $a$   
 $\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$

II :

$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $\bigcirc \diamond(a \wedge \Box \neg b)$

III :

$\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$



I :

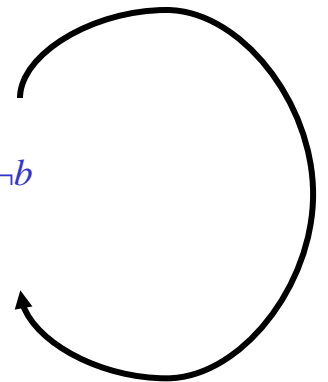
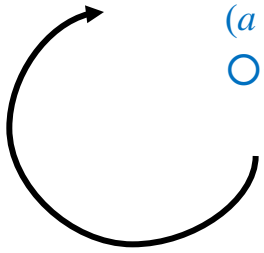
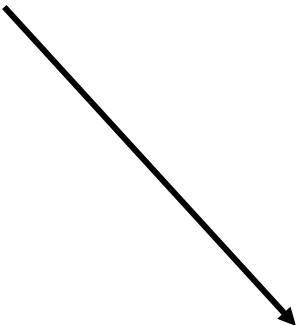
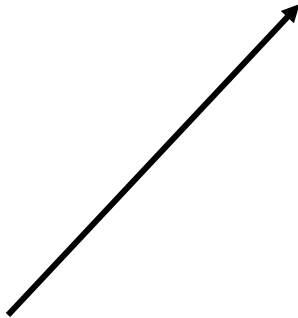
$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $a \wedge \square\neg b$   
 $a$   
 $\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$

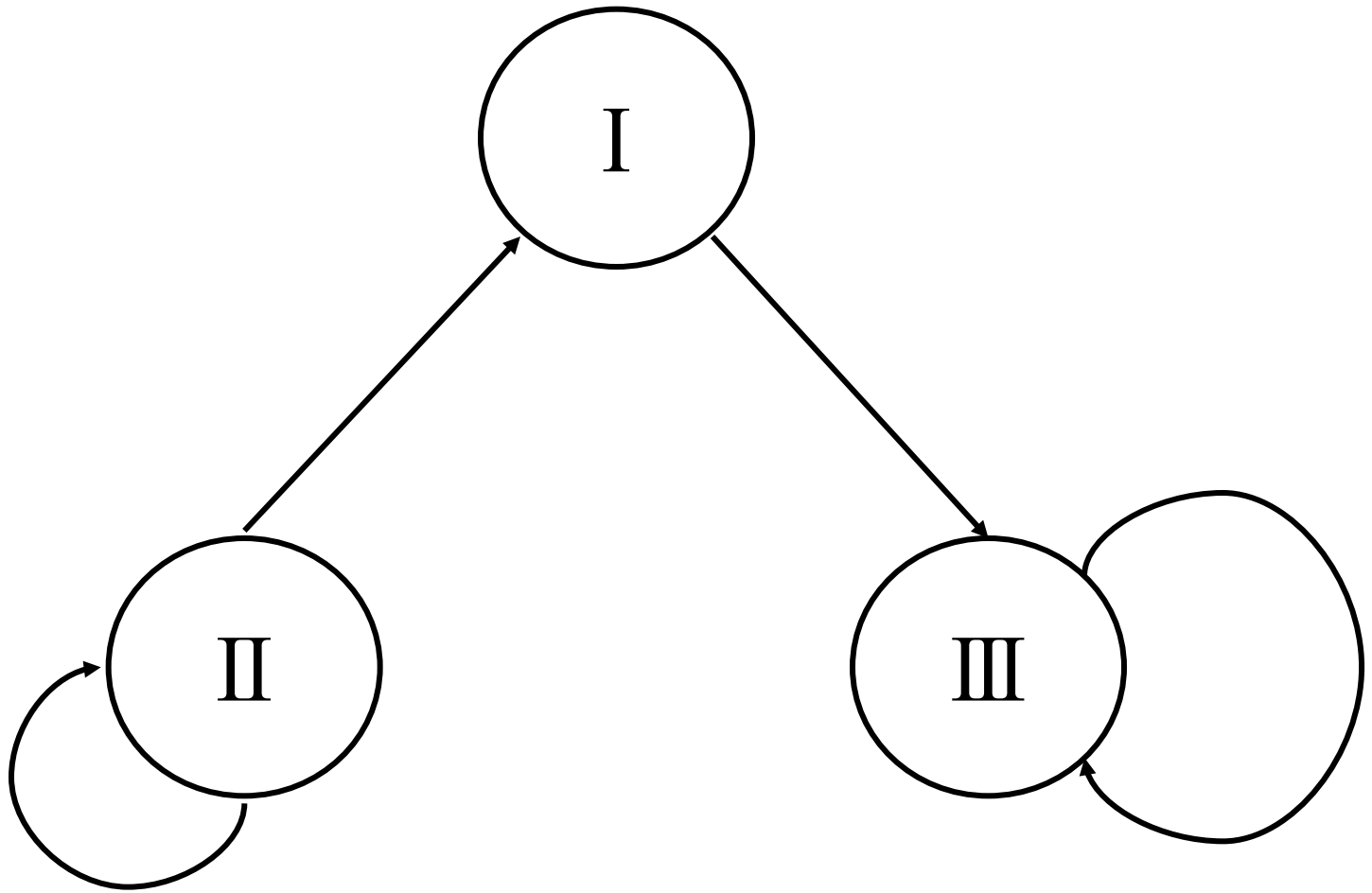
II :

$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $\bigcirc\diamond(a \wedge \square\neg b)$

III :

$\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$





# Making a “Symbolic” Model

- If  $\pi = \pi_0, \pi_1, \pi_2, \dots$  is a model of  $\varphi_0$ , then there exists an infinite sequence

$$\Pi = \Gamma_0, \Gamma_1, \Gamma_2, \dots$$

of  $\varphi_0$ -types such that

- $\varphi_0 \in \Gamma_0$
- $\pi^i \models \varphi$  for any  $\varphi \in \Gamma_i$
- $\Gamma_i \rightarrow \Gamma_{i+1}$
- If  $\Diamond \varphi \in \Gamma_i$ , then  $\varphi \in \Gamma_j$  for some  $j \geq i$



$$\pi = \pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \dots$$

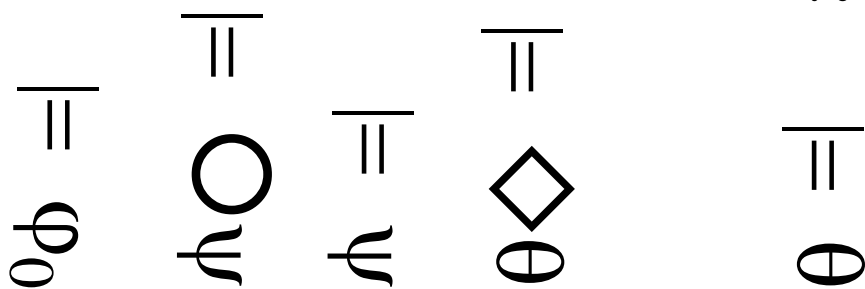
$$\pi^1 = \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \dots$$

$$\pi^2 = \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \dots$$

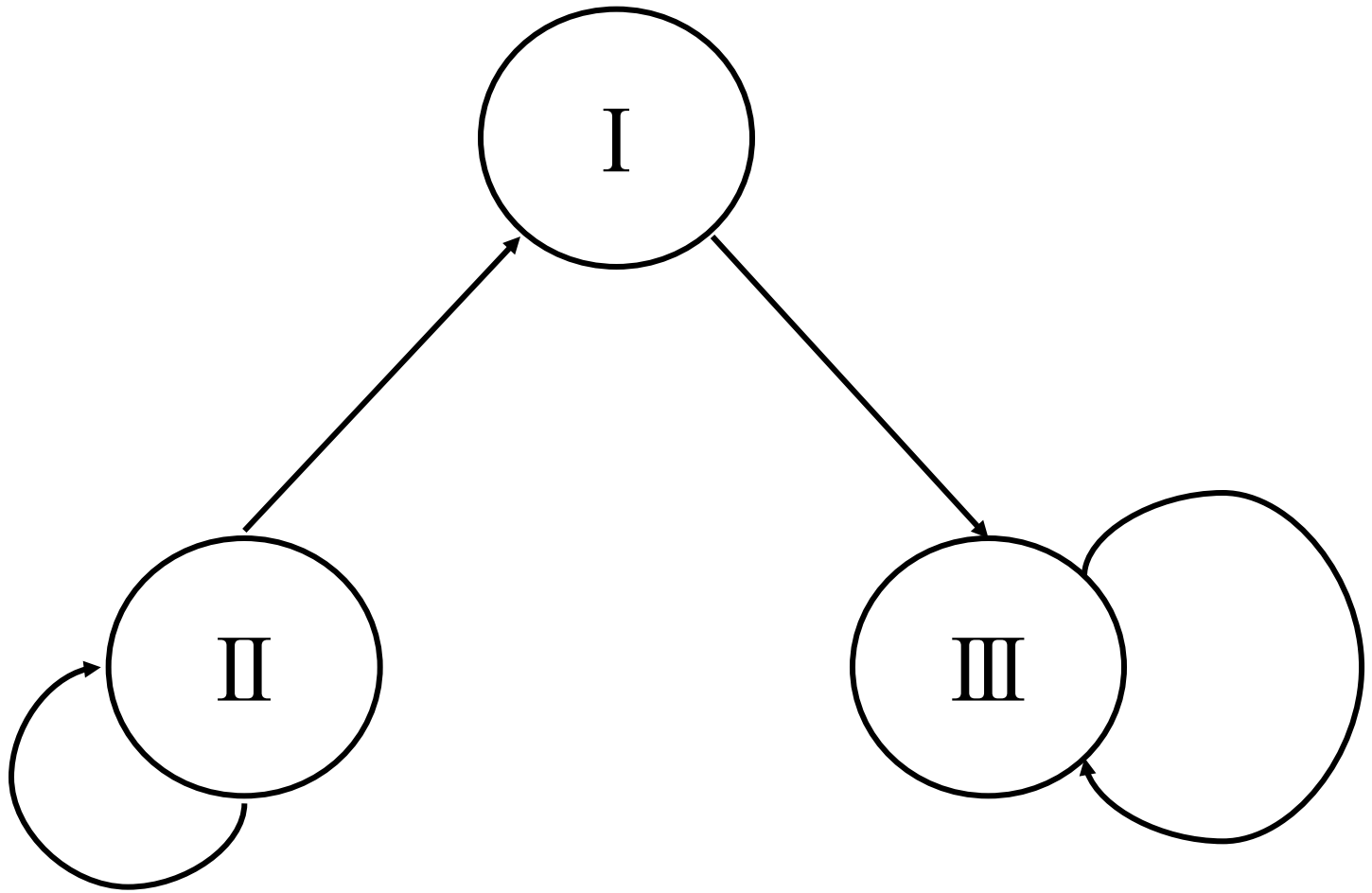
$$\pi^3 = \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \dots$$

$$\pi^4 = \pi_4, \pi_5, \pi_6, \pi_7, \dots$$

$$\pi^5 = \pi_5, \pi_6, \pi_7, \dots$$



$$\Pi = \Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, \dots$$



# Conversely ...

- If an infinite sequence  $\Pi = \Gamma_0, \Gamma_1, \Gamma_2, \dots$  of  $\varphi_0$ -types  $\Gamma_i \subseteq cl(\varphi_0)$  satisfies the above conditions and the following, then  $\pi^i \models \varphi$  for any  $\varphi \in \Gamma_i$  (in particular,  $\pi \models \varphi_0$ )
  - If  $P \in \Gamma_i$ , then  $P \in L(\pi_i)$
  - If  $\neg P \in \Gamma_i$ , then  $P \in L(\pi_i)$  does not hold

# $\omega$ -automaton

- In order to characterize an infinite sequence that satisfies the above conditions, construct an  $\omega$ -automaton whose states are  $\varphi_0$ -types  $\Gamma \subseteq cl(\varphi_0)$
- Its transitions are defined by  $\Gamma \rightarrow \Gamma'$
- Initial states  $\Gamma_0$  satisfy  $\varphi_0 \in \Gamma_0$

I :

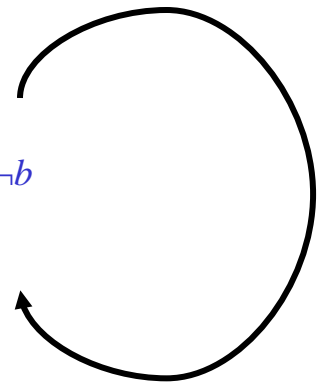
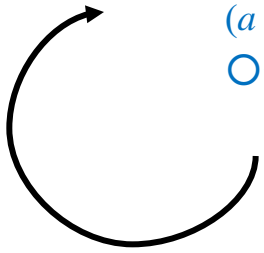
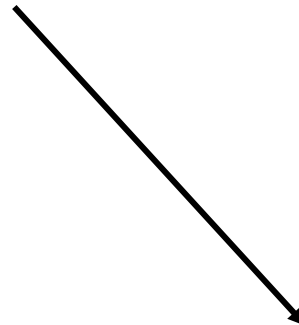
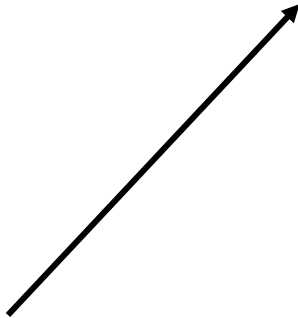
$\diamond(a \wedge \Box\neg b)$   
 $(a \wedge \Box\neg b) \vee \bigcirc\diamond(a \wedge \Box\neg b)$   
 $a \wedge \Box\neg b$   
 $a$   
 $\Box\neg b$   
 $\neg b \wedge \bigcirc\Box\neg b$   
 $\neg b$   
 $\bigcirc\Box\neg b$

II :

$\diamond(a \wedge \Box\neg b)$   
 $(a \wedge \Box\neg b) \vee \bigcirc\diamond(a \wedge \Box\neg b)$   
 $\bigcirc\diamond(a \wedge \Box\neg b)$

III :

$\Box\neg b$   
 $\neg b \wedge \bigcirc\Box\neg b$   
 $\neg b$   
 $\bigcirc\Box\neg b$



I :

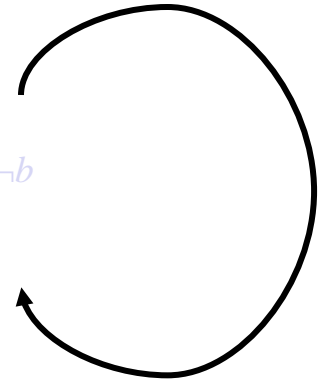
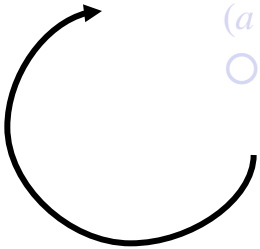
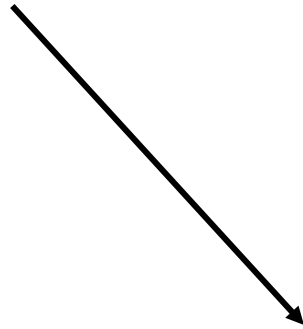
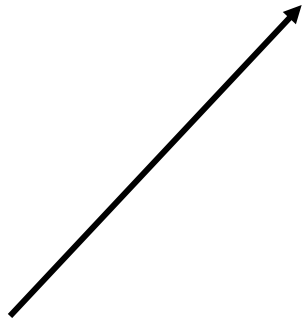
$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $a \wedge \square\neg b$   
 $a$   
 $\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$

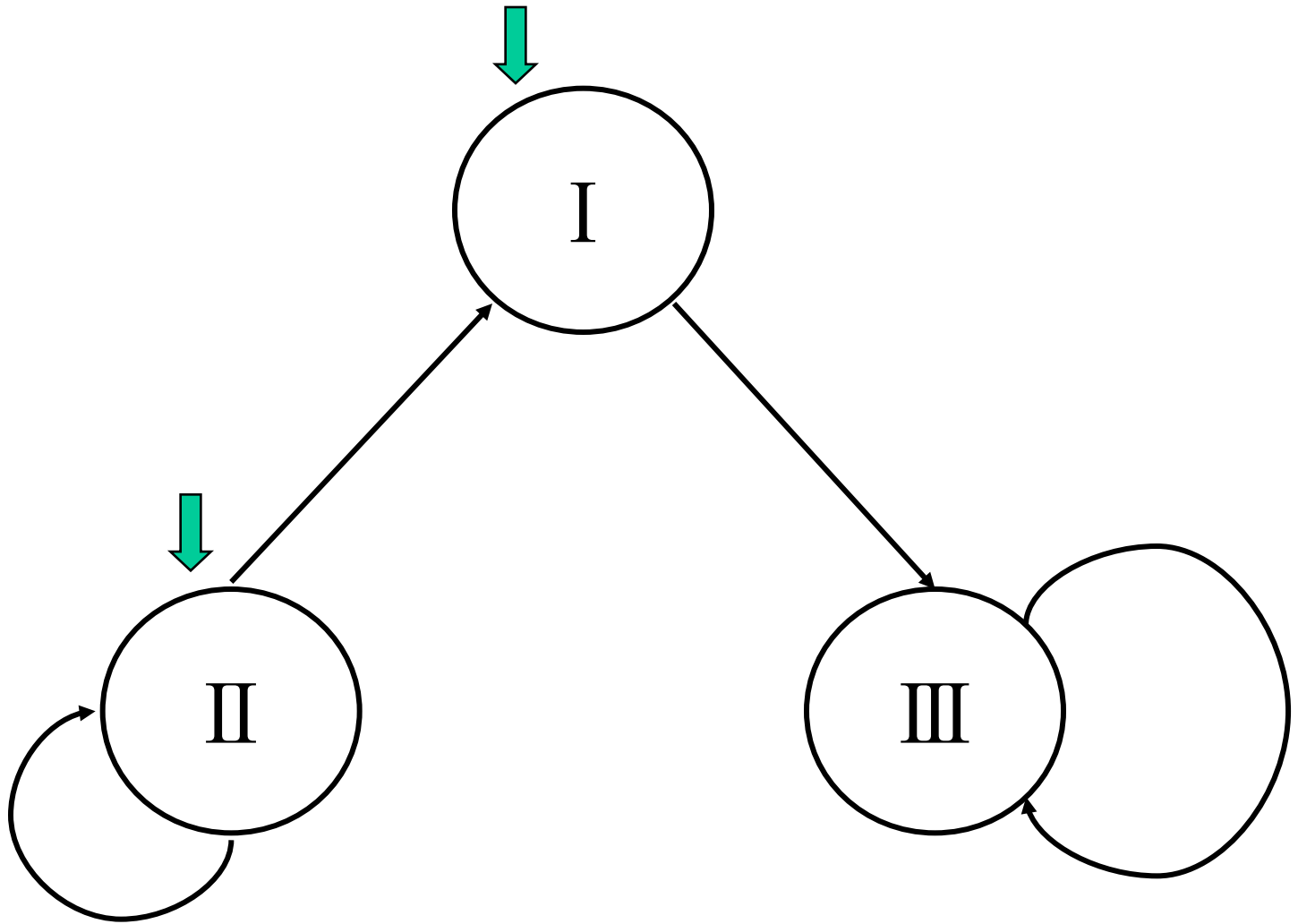
II :

$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $\bigcirc\diamond(a \wedge \square\neg b)$

III :

$\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$





# Labels

- In order to check the following conditions, propositional symbols and their negations are put as labels to states ( $\varphi_0$ -types)
- If  $P \in \Gamma_i$ , then  $P \in L(\pi_i)$
- If  $\neg P \in \Gamma_i$ , then  $P \in L(\pi_i)$  does not hold



I :

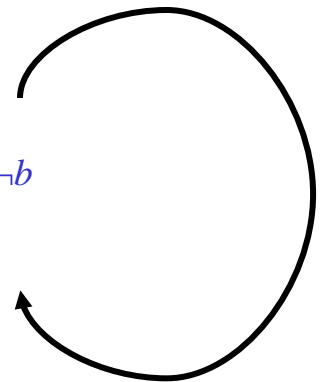
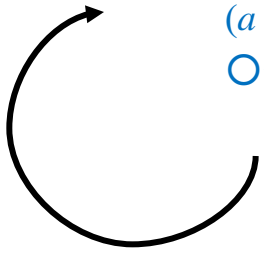
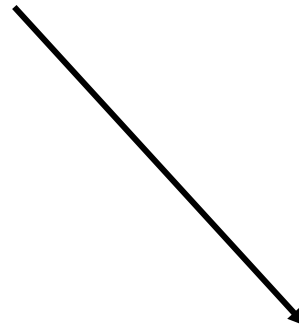
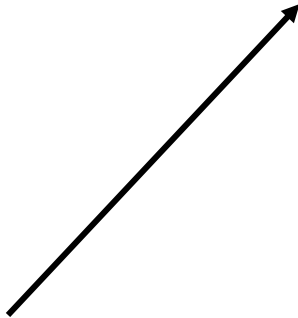
$\diamond(a \wedge \Box\neg b)$   
 $(a \wedge \Box\neg b) \vee \bigcirc\diamond(a \wedge \Box\neg b)$   
 $a \wedge \Box\neg b$   
 $a$   
 $\Box\neg b$   
 $\neg b \wedge \bigcirc\Box\neg b$   
 $\neg b$   
 $\bigcirc\Box\neg b$

II :

$\diamond(a \wedge \Box\neg b)$   
 $(a \wedge \Box\neg b) \vee \bigcirc\diamond(a \wedge \Box\neg b)$   
 $\bigcirc\diamond(a \wedge \Box\neg b)$

III :

$\Box\neg b$   
 $\neg b \wedge \bigcirc\Box\neg b$   
 $\neg b$   
 $\bigcirc\Box\neg b$



I :

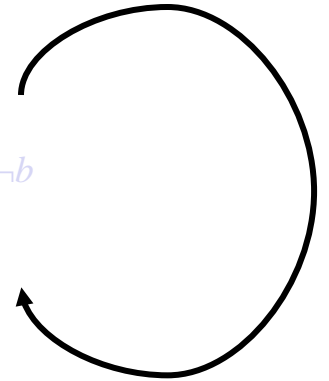
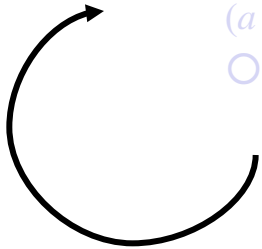
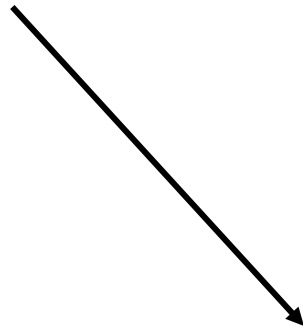
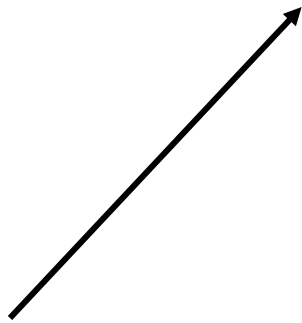
$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $a \wedge \square\neg b$   
 $a$   
 $\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$

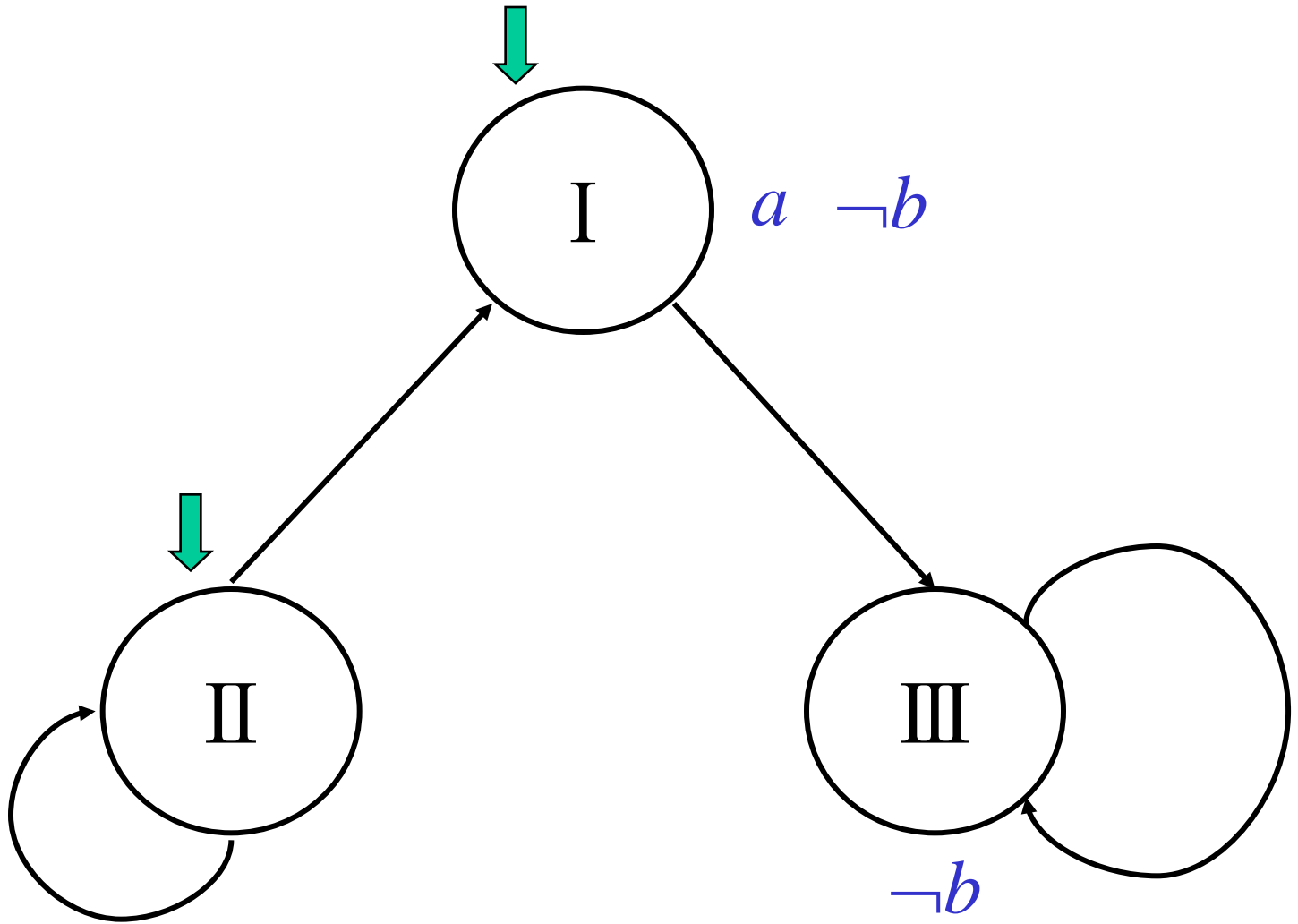
II :

$\diamond(a \wedge \square\neg b)$   
 $(a \wedge \square\neg b) \vee \bigcirc\diamond(a \wedge \square\neg b)$   
 $\bigcirc\diamond(a \wedge \square\neg b)$

III :

$\square\neg b$   
 $\neg b \wedge \bigcirc\square\neg b$   
 $\neg b$   
 $\bigcirc\square\neg b$





# Condition on Infinite Paths

- We want to characterize infinite paths  $\Pi = \Gamma_0, \Gamma_1, \Gamma_2, \dots$  over the  $\omega$ -automaton that satisfy the following conditions

If  $\Diamond \varphi \in \Gamma_i$ , then  $\varphi \in \Gamma_j$  for some  $j \geq i$

- This is equivalent to

For each  $\Diamond \varphi \in cl(\varphi_0)$ , elements of  $F(\Diamond \varphi)$  occur infinite times

- Where

$$F(\Diamond \varphi) = \{ \Gamma \mid \Diamond \varphi \notin \Gamma \text{ or } \varphi \in \Gamma \}$$

# What is $F(\Diamond(a \wedge \Box\neg b))$ ?

I :

- $\Diamond(a \wedge \Box\neg b)$
- $(a \wedge \Box\neg b) \vee \bigcirc\Diamond(a \wedge \Box\neg b)$
- $a \wedge \Box\neg b$
- $a$
- $\Box\neg b$
- $\neg b \wedge \bigcirc\Box\neg b$
- $\neg b$
- $\bigcirc\Box\neg b$

1. none

2. I

3. I and II

4. I and III

II :

- $\Diamond(a \wedge \Box\neg b)$
- $(a \wedge \Box\neg b) \vee \bigcirc\Diamond(a \wedge \Box\neg b)$
- $\bigcirc\Diamond(a \wedge \Box\neg b)$

III :

- $\Box\neg b$
- $\neg b \wedge \bigcirc\Box\neg b$
- $\neg b$
- $\bigcirc\Box\neg b$

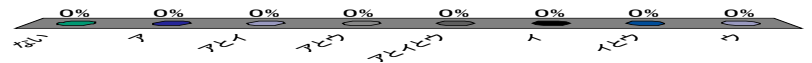
5. I and II and III

6. II

7. II and III

8. III

$$F(\Diamond\phi) = \{\Gamma \mid \Diamond\phi \notin \Gamma \text{ または } \phi \in \Gamma\}$$



I :

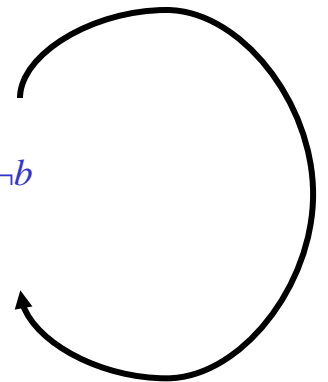
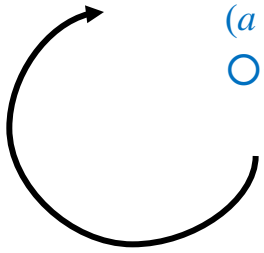
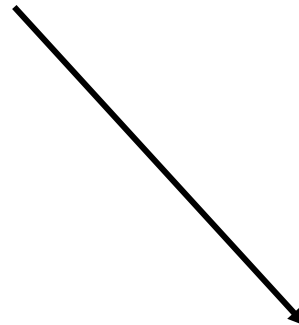
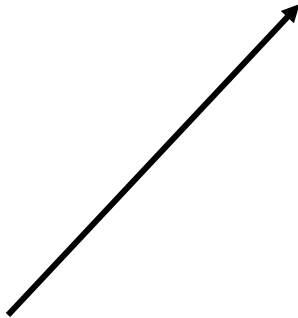
$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $a \wedge \Box \neg b$   
 $a$   
 $\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$

II :

$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $\bigcirc \diamond(a \wedge \Box \neg b)$

III :

$\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$



I :

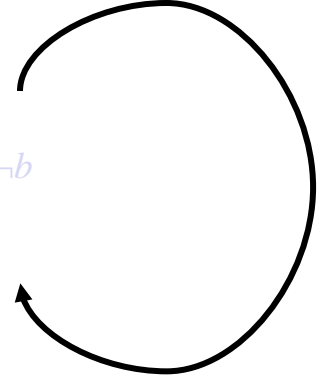
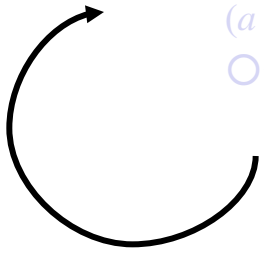
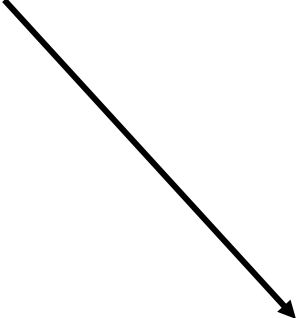
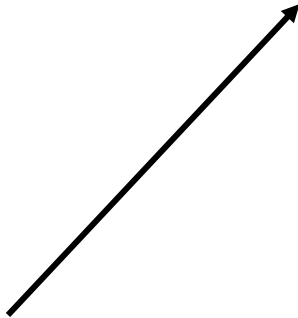
$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $a \wedge \Box \neg b$   
 $a$   
 $\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$

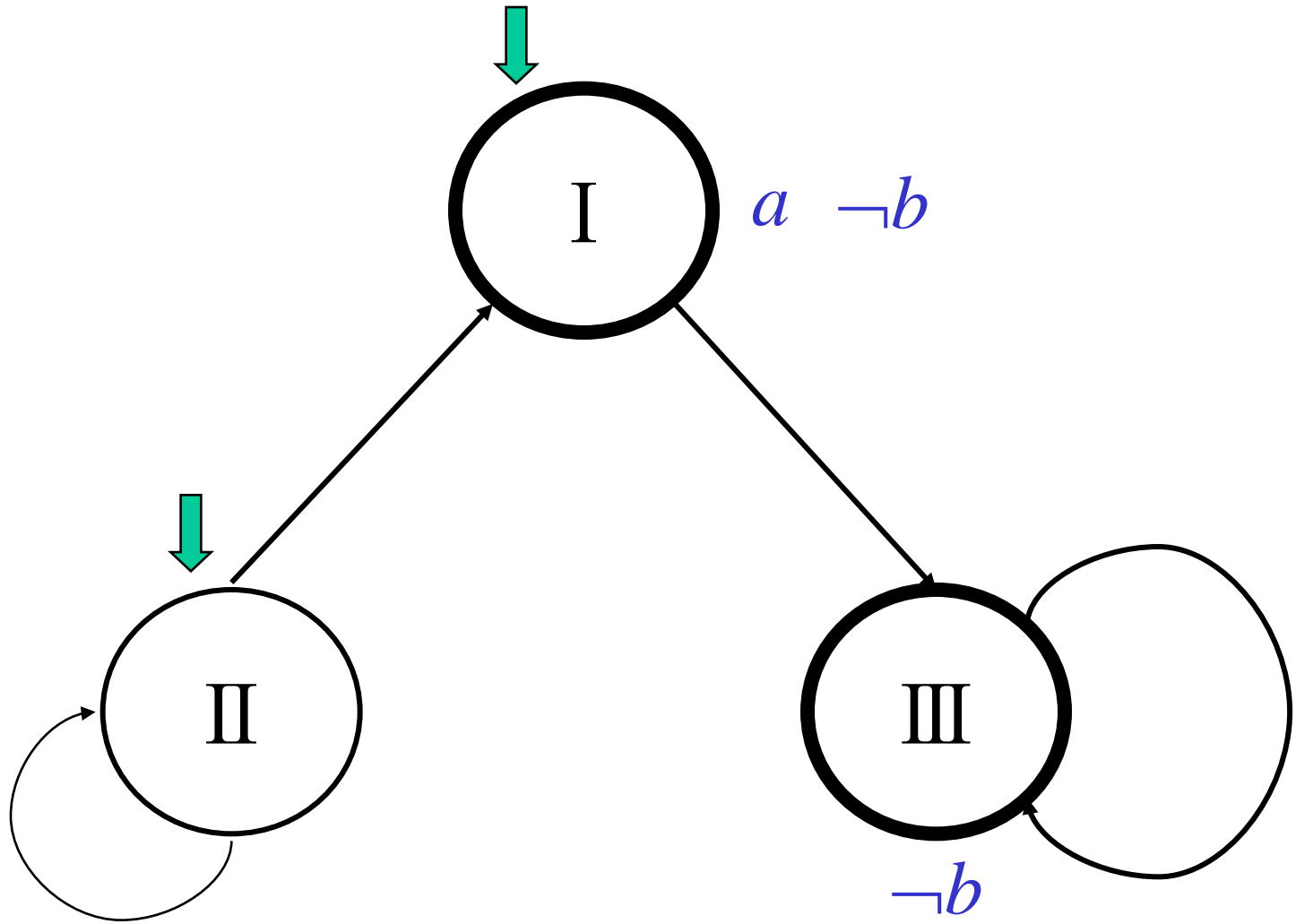
II :

$\diamond(a \wedge \Box \neg b)$   
 $(a \wedge \Box \neg b) \vee \bigcirc \diamond(a \wedge \Box \neg b)$   
 $\bigcirc \diamond(a \wedge \Box \neg b)$

III :

$\Box \neg b$   
 $\neg b \wedge \bigcirc \Box \neg b$   
 $\neg b$   
 $\bigcirc \Box \neg b$





$$F(\Diamond(a \wedge \Box \neg b)) = \{ I, III \}$$



# Model Checking

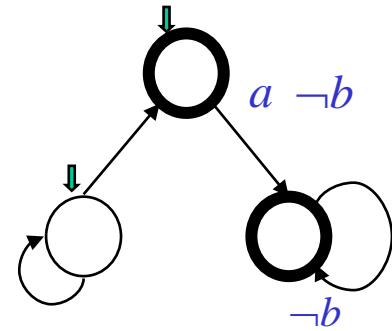
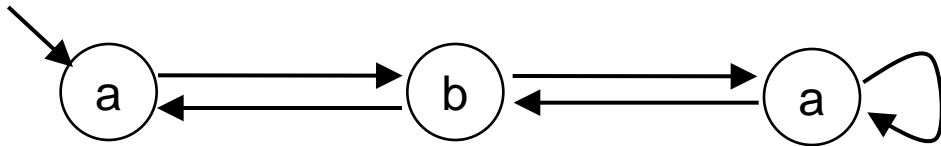
- For a Kripke structure  $K = \langle S, R, L \rangle$ ,  
an initial state  $s_0 \in S$ , and  
a formula  $\varphi_0$ ,  
Is there any model  $\pi = \pi_0, \pi_1, \pi_2, \dots$   
of  $\varphi_0$  that satisfies  $\pi_0 = s_0$ ?

# Equivalent to the Following

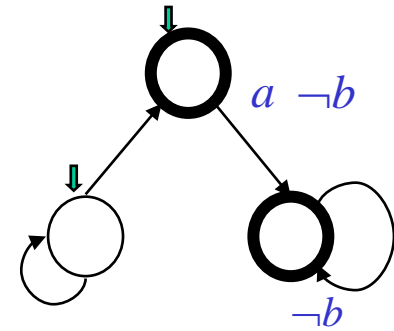
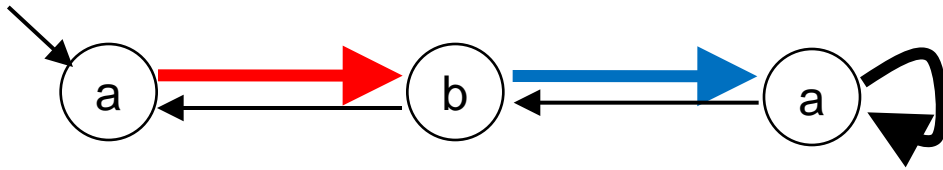
Is there an infinite path  $\Pi = \Gamma_0, \Gamma_1, \Gamma_2, \dots$  over the  $\omega$ -automaton such that

- For each  $\Diamond \varphi \in cl(\varphi_0)$ , elements of  $F(\Diamond \varphi)$  occur infinite times
- If  $P \in \Gamma_i$ , then  $P \in L(\pi_i)$
- If  $\neg P \in \Gamma_i$ , then  $P \in L(\pi_i)$  does not hold

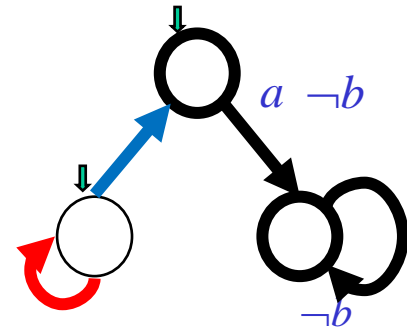
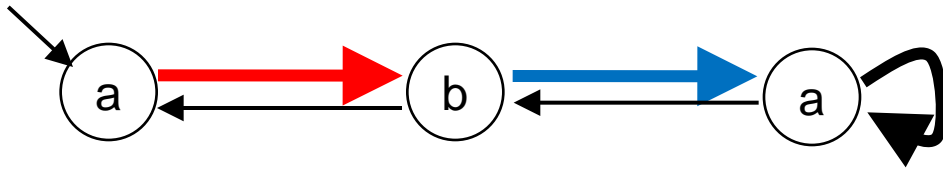
# Example



# Example



# Example



# Synchronous Product

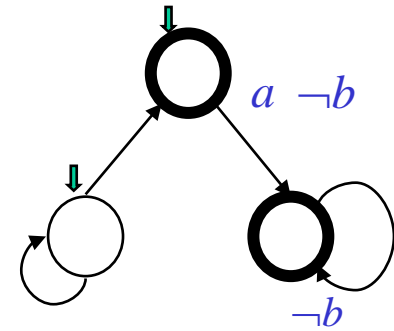
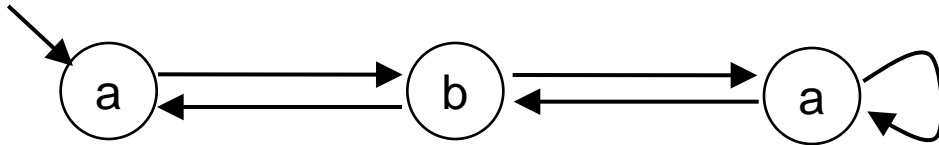
- In order to simultaneously check the existence of  $\pi$  and that of  $\Pi$ , construct the “synchronous product” of the  $\omega$ -automaton and  $K$

States:  $\langle s, \Gamma \rangle$  **where**  $\{P \mid P \in \Gamma\} \subseteq L(s)$   
 $\{P \mid \neg P \in \Gamma\} \cap L(s) = \emptyset$

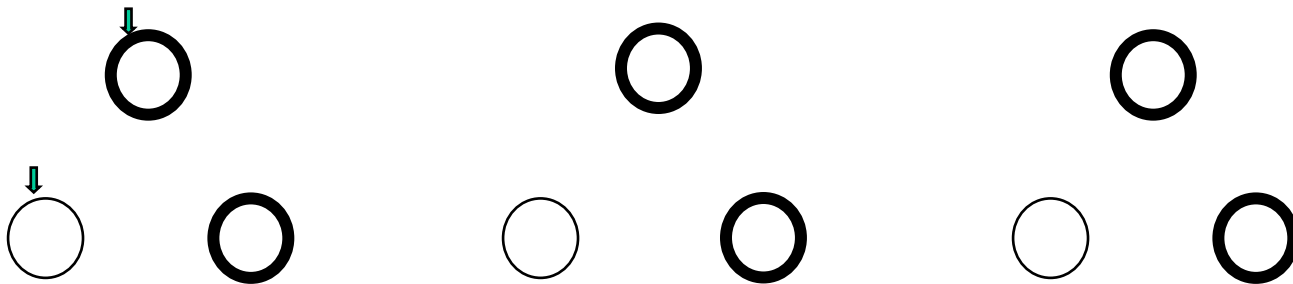
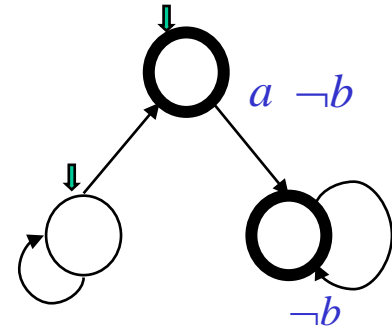
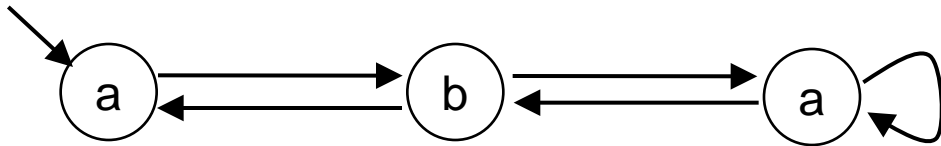
Initial states:  $\langle s_0, \Gamma_0 \rangle$  pair of initial states

Transitions:  $\langle s, \Gamma \rangle \rightarrow \langle s', \Gamma' \rangle$  **iff**  $R(s, s')$  **and**  $\Gamma \rightarrow \Gamma'$

# Synchronous Product

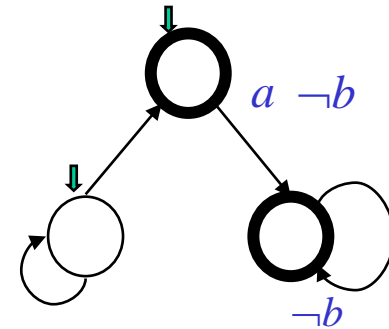
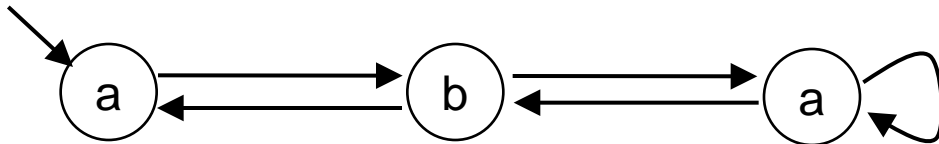


# Synchronous Product





# Which state does not exist?

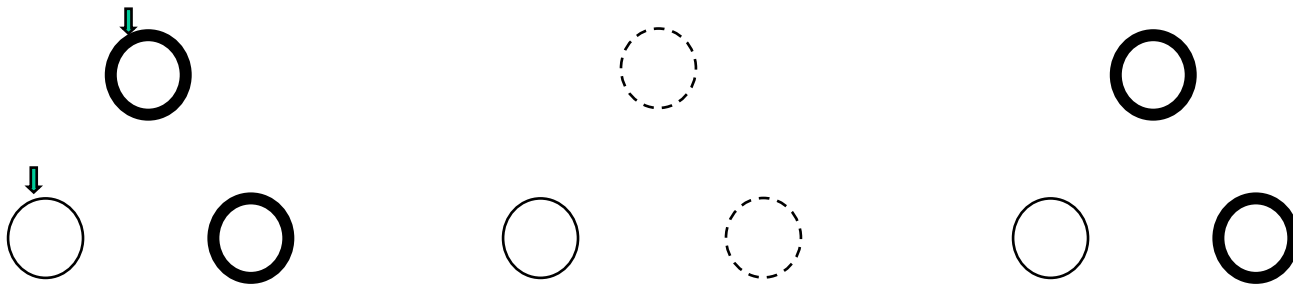
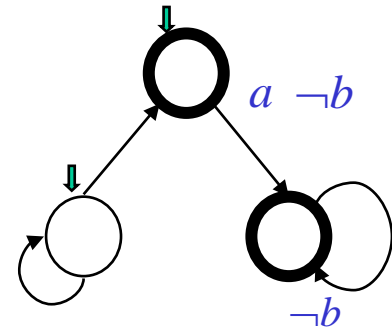
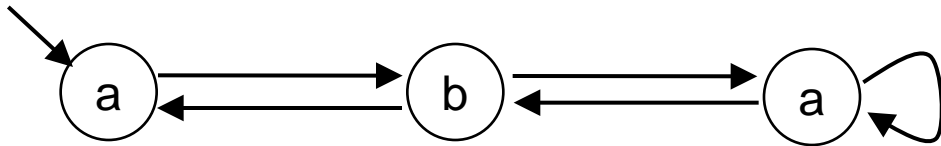


Five options for the missing state are shown, each with a percentage below it:

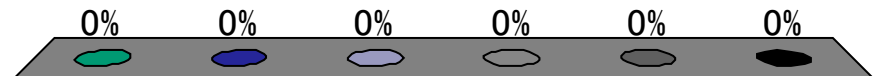
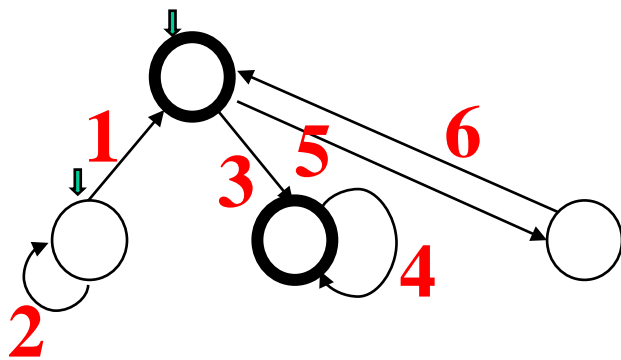
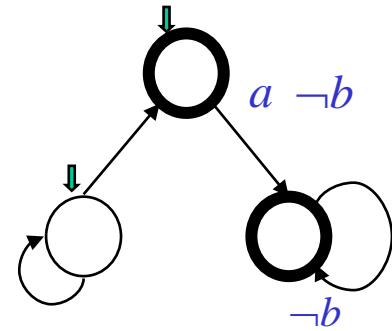
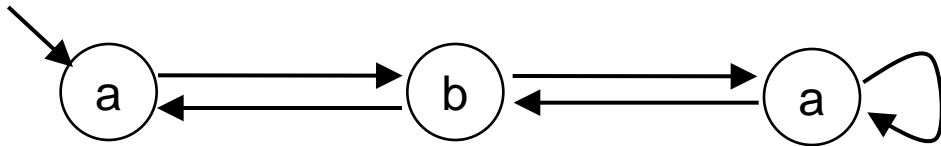
- Option 1: A double circle with a green arrow pointing down. 0%
- Option 2: A double circle. 0%
- Option 3: A double circle with a green arrow pointing down. 0%
- Option 4: A single circle with a green arrow pointing down. 0%
- Option 5: A single circle. 0%

Below the options is a horizontal bar with five colored ovals: green, blue, light blue, grey, and dark grey.

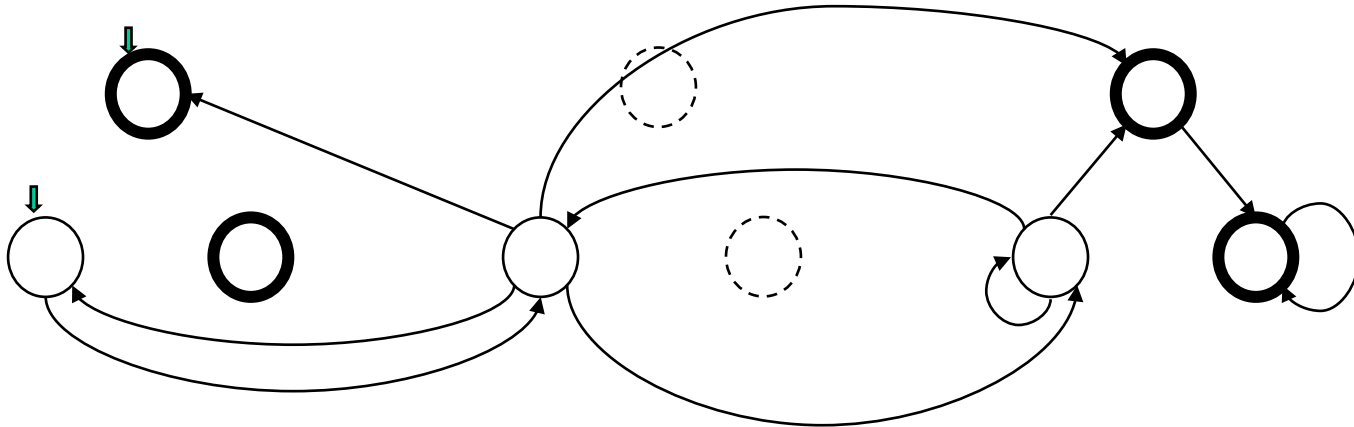
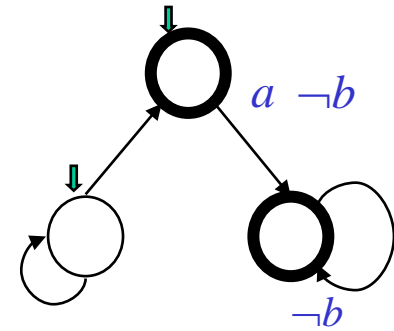
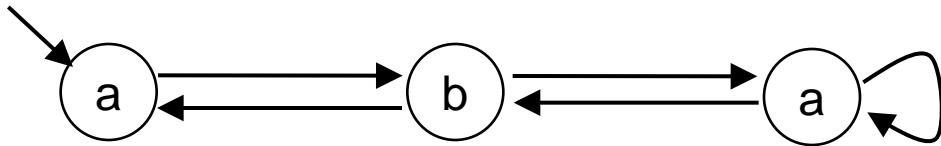
# Synchronous Product



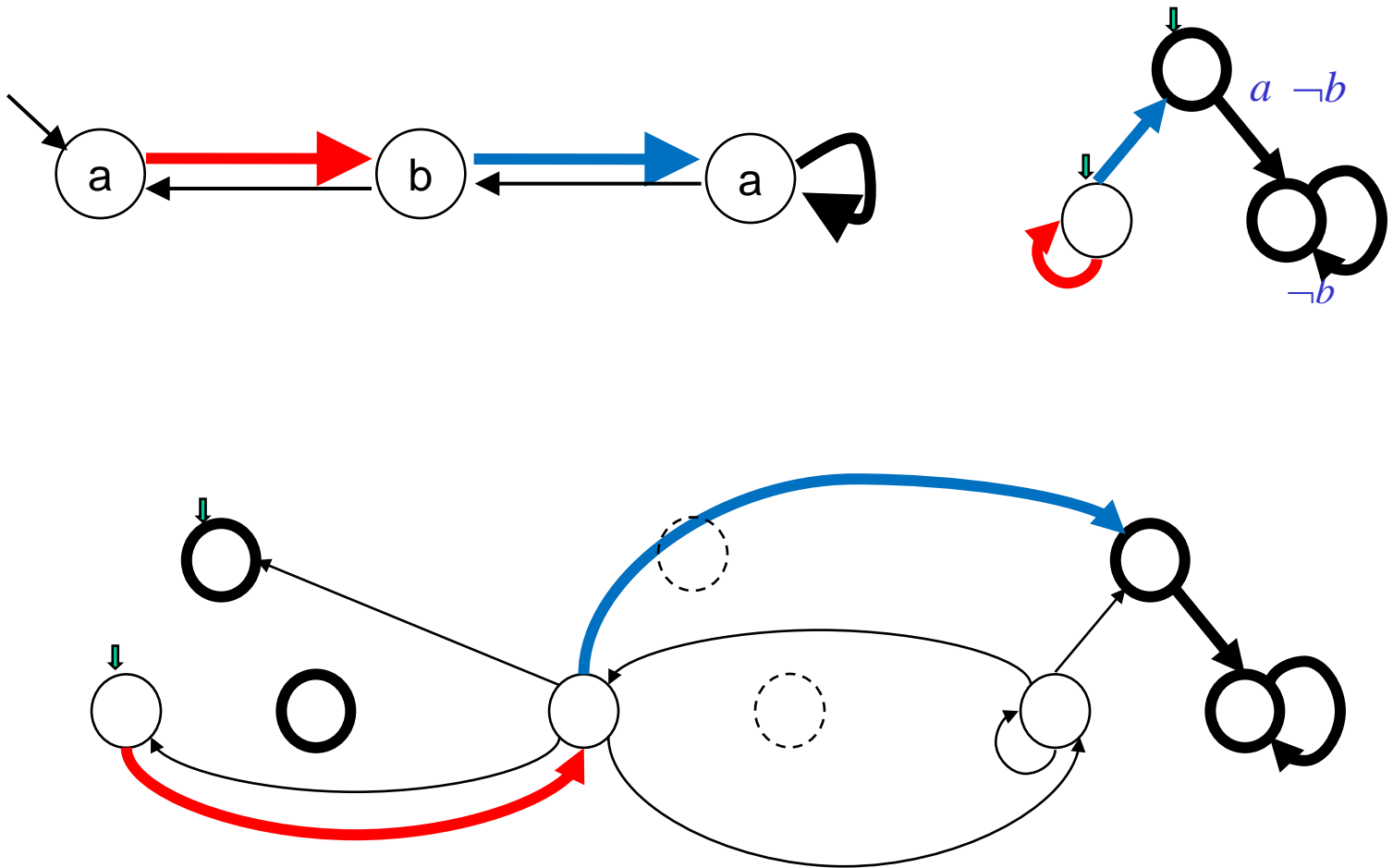
# Which transition exists?



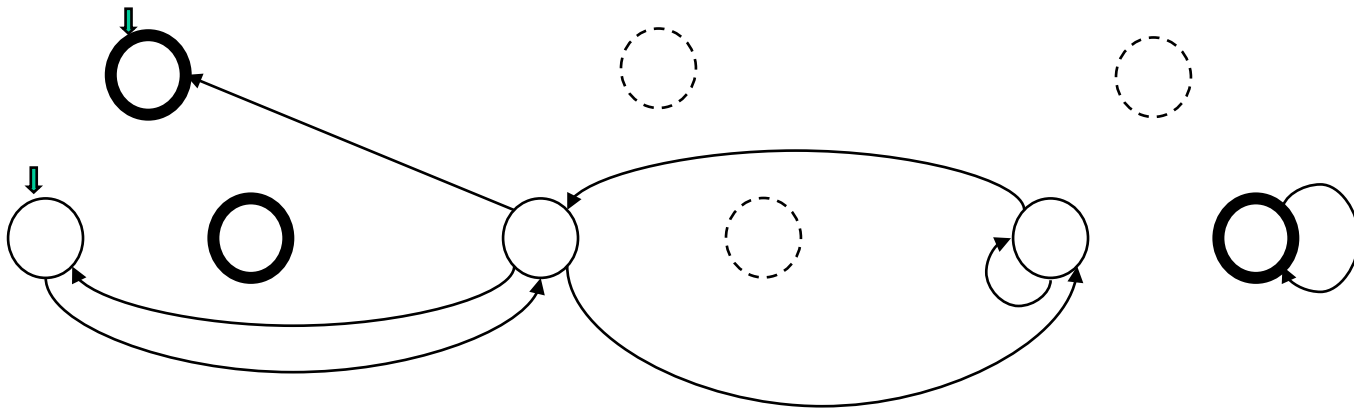
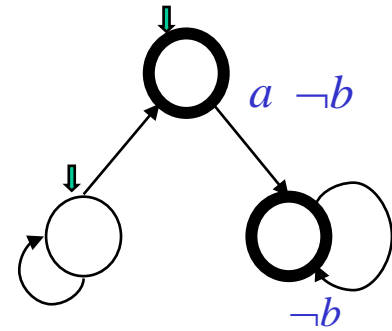
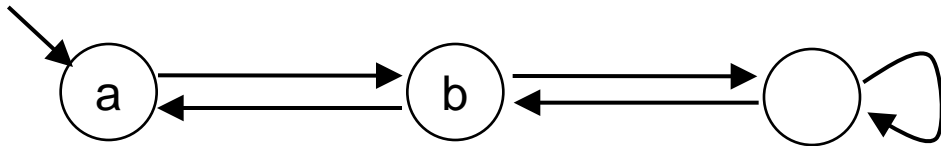
# Synchronous Product



# Synchronous Product



# Synchronous Product



# Condition for Existence of a Model

- There exists a model  $\pi = \pi_0, \pi_1, \pi_2, \dots$  of  $\varphi_0$  iff

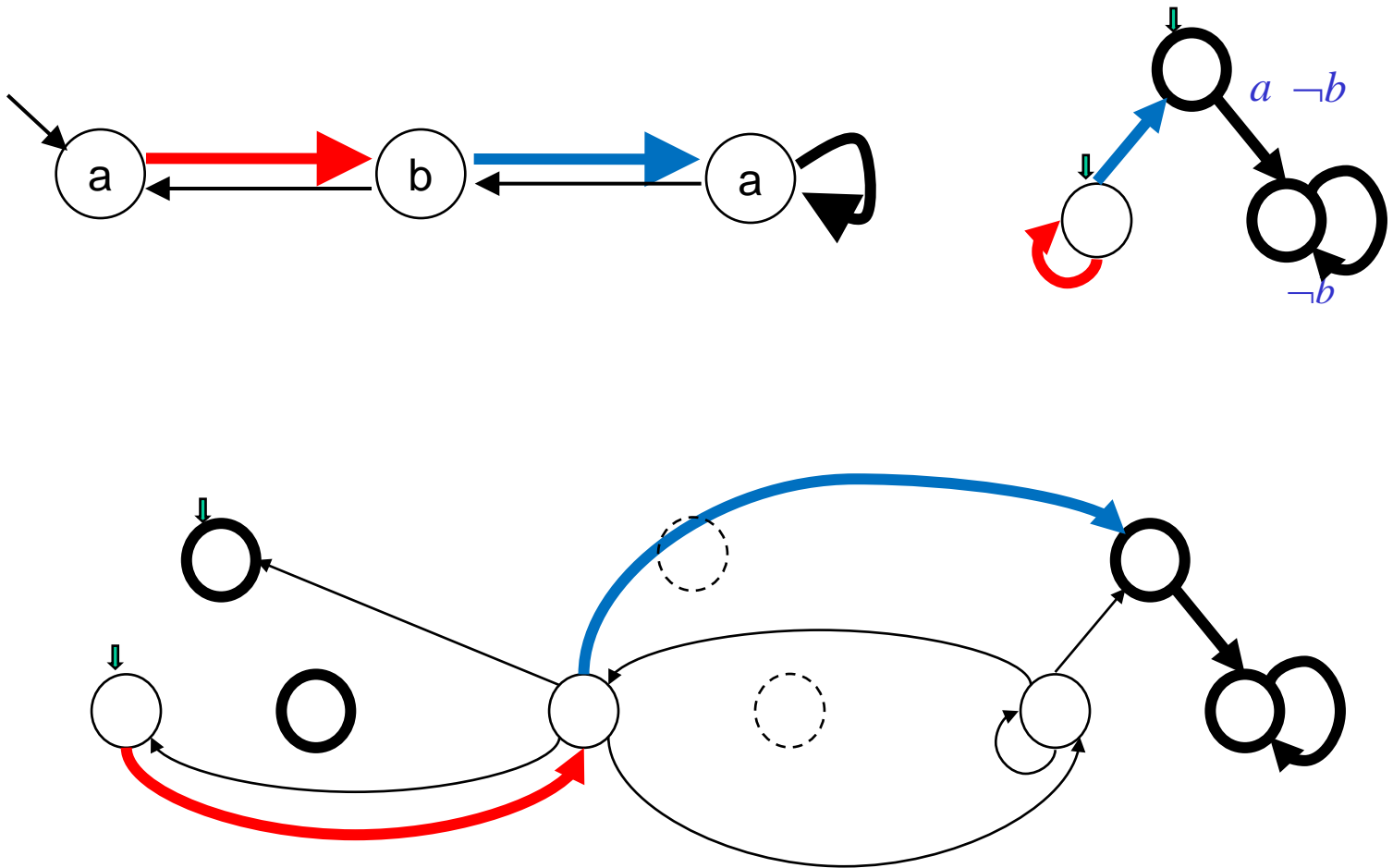
There exists an infinite path

$$\langle \pi_0, \Gamma_0 \rangle \rightarrow \langle \pi_1, \Gamma_1 \rangle \rightarrow \dots$$

in the synchronous product such that for each  $\Diamond\varphi \in cl(\varphi_0)$ , elements of  $F(\Diamond\varphi)$  occur infinite times

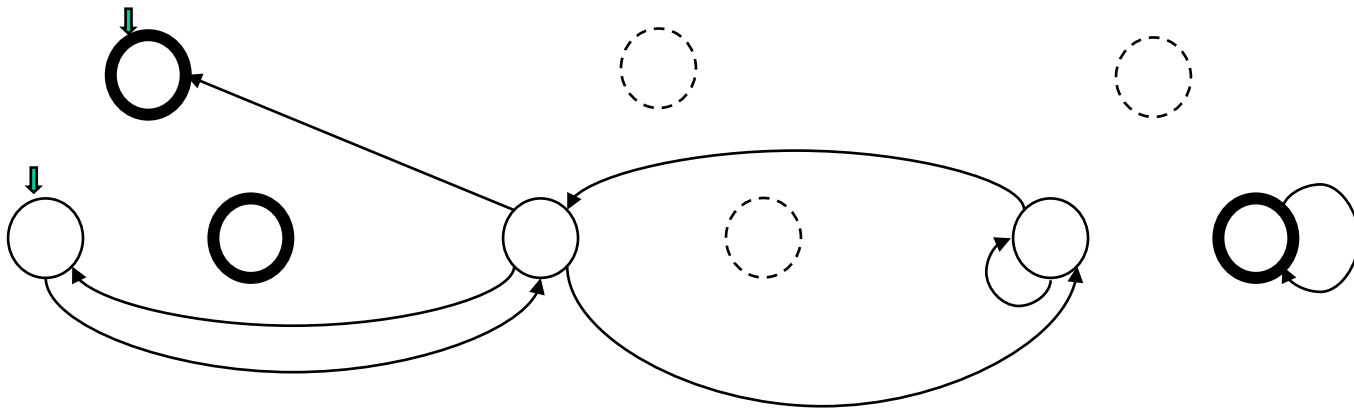
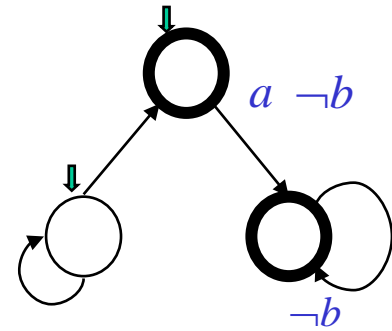
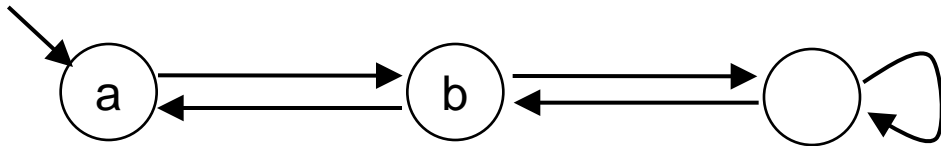
$\Leftrightarrow$  Existence of a kind of loop

# Synchronous Product





# Synchronous Product



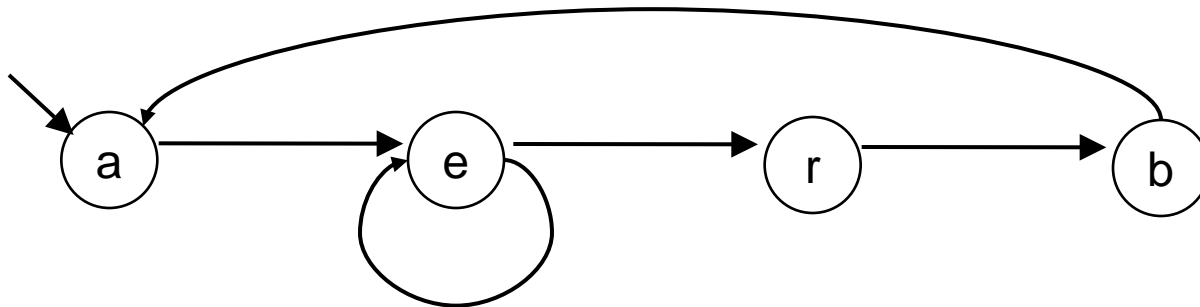
# Condition on a Loop

- Reachable from an initial state
- For each  $\Diamond\varphi \in cl(\varphi_0)$ , elements of  $F(\Diamond\varphi)$  occur infinite times
- Existence of such a loop can be decided by checking strongly connected components in the synchronous product

# Report

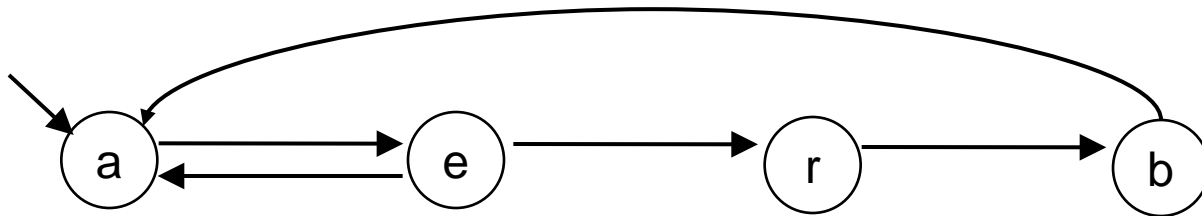
Don't forget  
to negate it

- Construct an  $\omega$ -automaton for verifying  $\Box \Diamond (e \supset r) \supset \Box (a \supset \Diamond b)$
- Verify that the formula holds in each path of the following Kripke structure



# Report'

- Construct an  $\omega$ -automaton for verifying  $(\Box \Diamond e \supset \Box \Diamond r) \supset \Box (a \supset \Diamond b)$
- Verify that the formula holds in each path of the following Kripke structure



**until**

$\pi \models \varphi$  **until**  $\psi$

**iff**  $\pi^i \models \psi$  **for some**  $i \geq 0$

**and**

$\pi^j \models \varphi$  **for any**  $j < i$

# release

$\pi \models \varphi$  **release**  $\psi$

**iff**  $\pi^i \models \varphi$  **for some**  $i \geq 0$

**and**

$\pi^j \models \psi$  **for any**  $j \leq i$

**or**  $\pi^j \models \psi$  **for any**  $j$

**until and release and  $\neg$**

$\pi \models \neg(\varphi \text{ **until** } \psi)$

**iff**  $\pi \models \neg\varphi \text{ **release** } \neg\psi$

$\pi \models \neg(\varphi \text{ **release** } \psi)$

**iff**  $\pi \models \neg\varphi \text{ **until** } \neg\psi$

# until and release and $\bigcirc$

$\pi \models \varphi$  **until**  $\psi$

**iff**  $\pi \models \psi \vee \bigcirc(\varphi \text{ until } \psi)$

$\pi \models \varphi$  **release**  $\psi$

**iff**  $\pi \models \psi \wedge \bigcirc(\varphi \text{ release } \psi)$



# SPIN

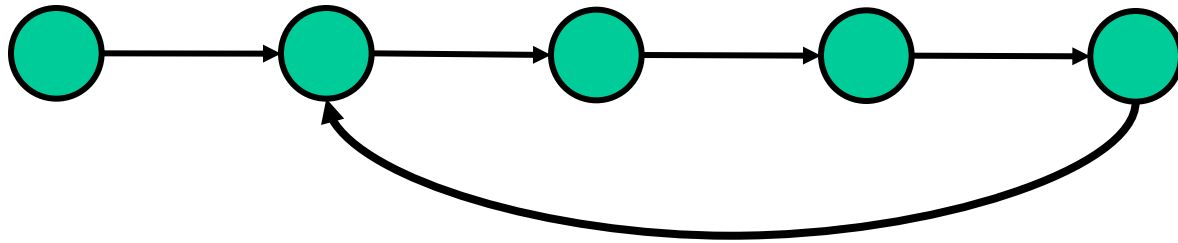
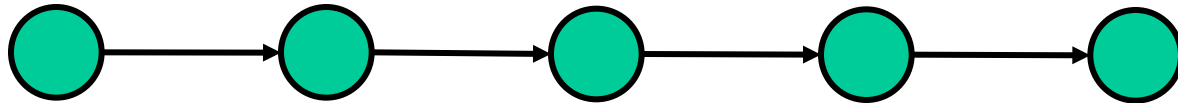
- One of the most popular model checkers
- The target system is described in Promela, a CSP-like concurrent language
- The property is defined in LTL and translated into a NEVER clause of Promela
- The synchronous product is verified
- Applied to verify protocols, algorithms, (software) designs, etc.
- <http://spinroot.com/spin/whatispin.html>

# Bounded Model Checking and SAT/SMT

# Bounded Model Checking

- If a verified property is refuted by a finite prefix of a path or by a path with a simple loop, then it gives a counter example
- Verification with respect to a finite prefix or a simple loop can be translated into a SAT problem
- SAT solvers are continuously improved
- Why not use SAT solvers?

# Bounded Paths



$(4,1)$ -loop

4-loop

# Bounded Semantics

$\pi \models_k \varphi$

if  $\pi$  is a  $k$ -loop and  $\pi \models \varphi$ , or

if  $\pi \models \varphi$  can be decided only

by looking at  $\pi_0, \pi_1, \dots, \pi_k$

$\models_k \mathbf{E}\varphi$  iff  $\pi \models_k \varphi$  for some  $\pi$

$\models \mathbf{E}\varphi$  iff  $\pi \models \varphi$  for some  $\pi$

# Theorem

$\models \mathbf{E}\varphi$  **iff**  $\models_k \mathbf{E}\varphi$  for some  $k$

Therefore, if  $\varphi$  is the negation of a verified property,  $\models_k \mathbf{E}\varphi$  gives an counter-example

# Theorem

$\models \mathbf{E}\varphi$  **iff**  $\models_k \mathbf{E}\varphi$  for some  $k$

Therefore if  $\varphi$  is a formula of a verification condition system, then  $\varphi$  is satisfiable if and only if it is satisfiable in some finite model. In fact,  $k$  can be found in the range up to  $|M| \times 2^{|\varphi|}$ , where  $|M|$  is the number of states and  $|\varphi|$  is the length of  $\varphi$ .

# Translation to SAT

$\pi_i$  is represented by  $x_{i1}, \dots, x_{in}$

$\pi \models_k \varphi$  is translated into

a big boolean formula over

$x_{01}, \dots, x_{0n}, \dots, x_{k1}, \dots, x_{kn}$



# Translation for a $(k,l)$ -loop

$$\text{Trl}(P, l, i, k) := P(x_{i1}, \dots, x_{in})$$

$$\text{Trl}(\neg P, l, i, k) := \neg P(x_{i1}, \dots, x_{in})$$

$$\text{Trl}(\varphi \wedge \psi, l, i, k) := \text{Trl}(\varphi, l, i, k) \wedge \text{Trl}(\psi, l, i, k)$$

$$\text{Trl}(\bigcirc \varphi, l, i, k) := \text{Trl}(\varphi, l, \text{succ}(i), k)$$

$$\text{succ}(i) := \text{if } i=k \text{ then } l \text{ else } i+1$$

$$\text{Trl}(\Box \varphi, l, i, k) := \bigwedge_{j=\min(i,l)}^k \text{Trl}(\varphi, l, j, k)$$

# Introduction of New Variables

- A new boolean variable is introduced for each subformula and  $i$

$$(a \wedge b) \vee (c \rightarrow (a \wedge b))$$



$$(x \vee (c \rightarrow x)) \wedge (x = (a \wedge b))$$

# Example

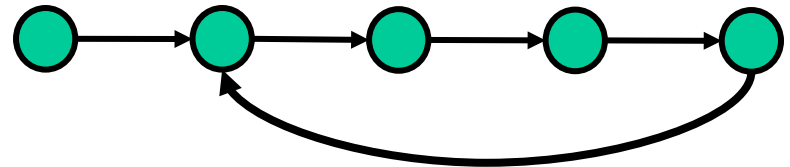
$$\text{Trl}(\Diamond(a \wedge \Box\neg b), 1, 0, 4) = (a_0 \wedge (\neg b_0 \wedge y)) \vee x$$

$$x = \text{Trl}(\Diamond(a \wedge \Box\neg b), 1, 1, 4) =$$

$$(a_1 \wedge y) \vee (a_2 \wedge y) \vee (a_3 \wedge y) \vee (a_4 \wedge y)$$

$$y = \text{Trl}(\Box\neg b, 1, 1, 4) =$$

$$\neg b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4$$



- In addition,  $(a_i, b_i, \dots)$  and  $(a_{i+1}, b_{i+1}, \dots)$  should satisfy the transition relation  $R$
- $(a_0, b_0, \dots)$  should be an initial state

# SAT (Satisfiability)

- The problem of deciding whether there exists an assignment that makes a given boolean formula true
  - Assignment --- mapping each boolean variable to 0 or 1

# SAT Solvers

- Please refer to other lectures on algorithms
- <http://www.satcompetition.org>
- DPLL
  - Davis, Putnam, Logemann, and Loveland
  - Assumes conjunctive normal forms
  - To satisfy  $\varphi \wedge rest$ , tentatively chooses one literal from  $\varphi$  and recursively attacks *rest* ... while backtracking ...

# SMT (Satisfiability Modulo Theories)

- Satisfiability under a specific theory (or combination of theories)
  - See the next slide
- <http://www.smtcomp.org>
- DLPP( $T$ )
  - Works as DLPP while calling the  $T$ -solver, which decides satisfiability of conjunctions, and validity of implications between conjunctions

# Examples of Theories

- Equality and uninterpreted functions
- Peano arithmetic
- Linear integer arithmetic with constant multiplication
- Real **number** arithmetic with multiplication
- Rational number arithmetic **with**out constant multiplication
- (Acyclic) recursive types
- Arrays

# Nelson-Oppen

- Method to handle combinations of theories
- Split a formula with symbols from  $T_1$  and  $T_2$  into a formula in  $T_1$  and a formula in  $T_2$  (purification)

$$x \leq y \wedge y \leq x + f(x)$$

⇓

$$x \leq y \wedge y \leq x + z \wedge z = f(x)$$



# Abstract Model Checking and CEGAR

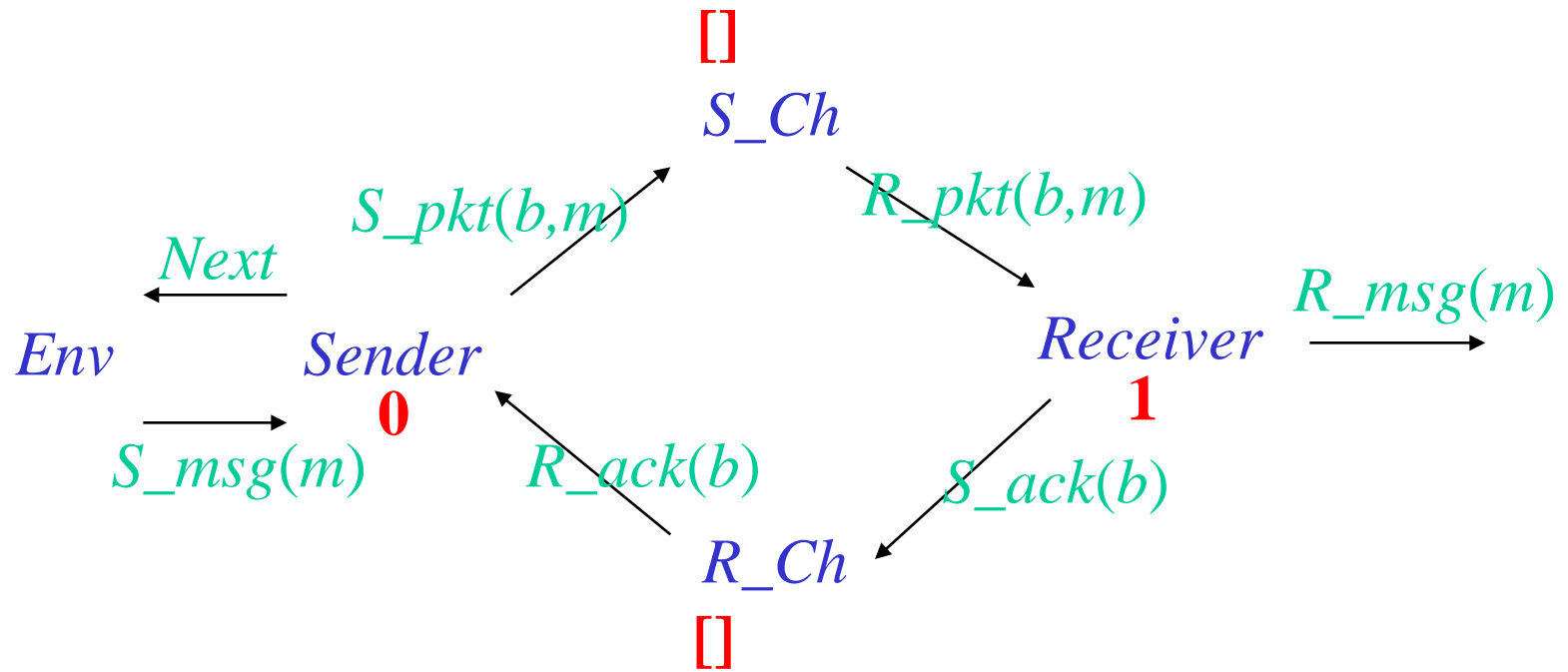
# Abstraction

- Abstraction of states  $\rightarrow$  abstract states
- A mapping  $\alpha : S \rightarrow A$
- Usually,  $A$  is much simpler than  $S$ 
  - Typically,  $A$  is a finite set while  $S$  is infinite
- Example: alternating bit protocol
  - State is a tuple of
    - Sender's state
    - Receiver's state
    - Channels' states --- infinite  $\leftarrow$  apply abstraction

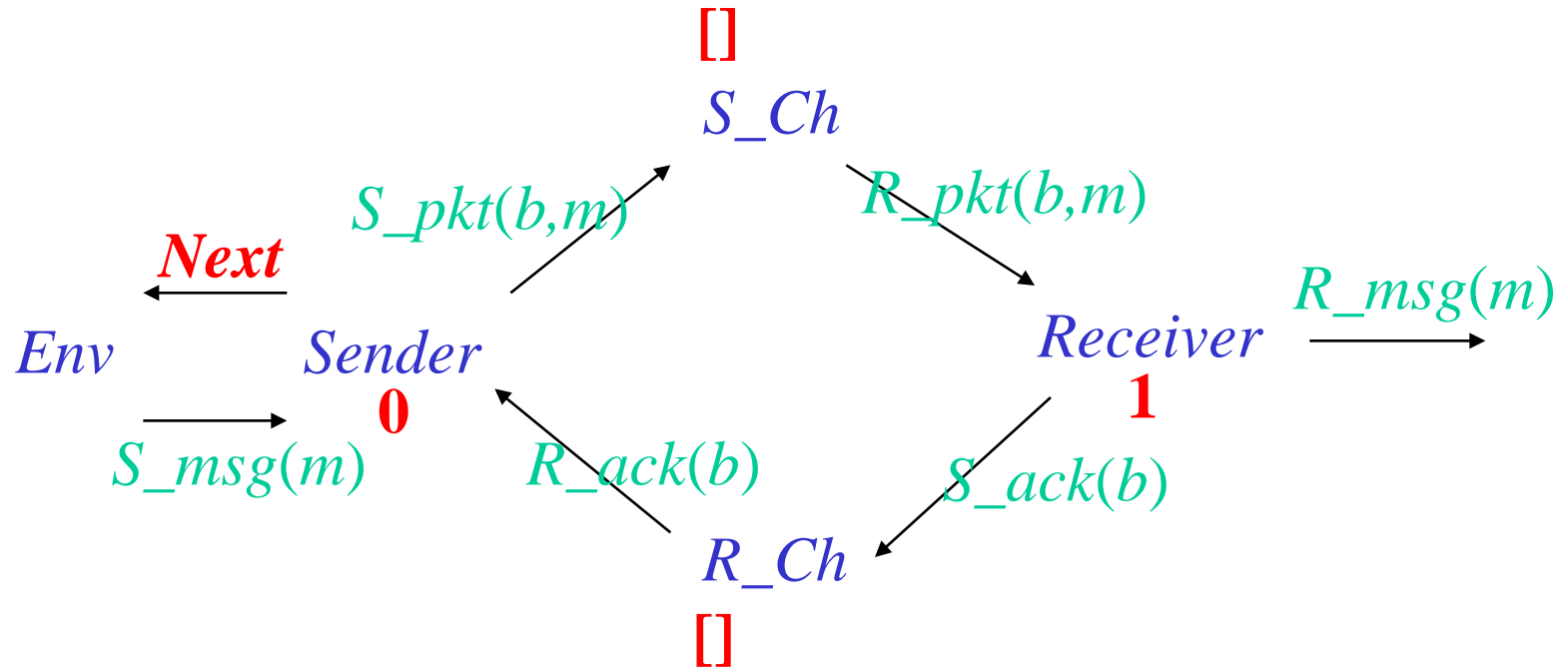
# Alternating Bit Protocol

- *Sender*
- *Receiver*
- *S\_Ch*: channel of message packets
  - Each packet is a pair of a message and a header
  - Modeled as a queue of packets
  - Messages may be duplicated or lost
- *R\_Ch*: channel of acknowledgements
  - Ditto

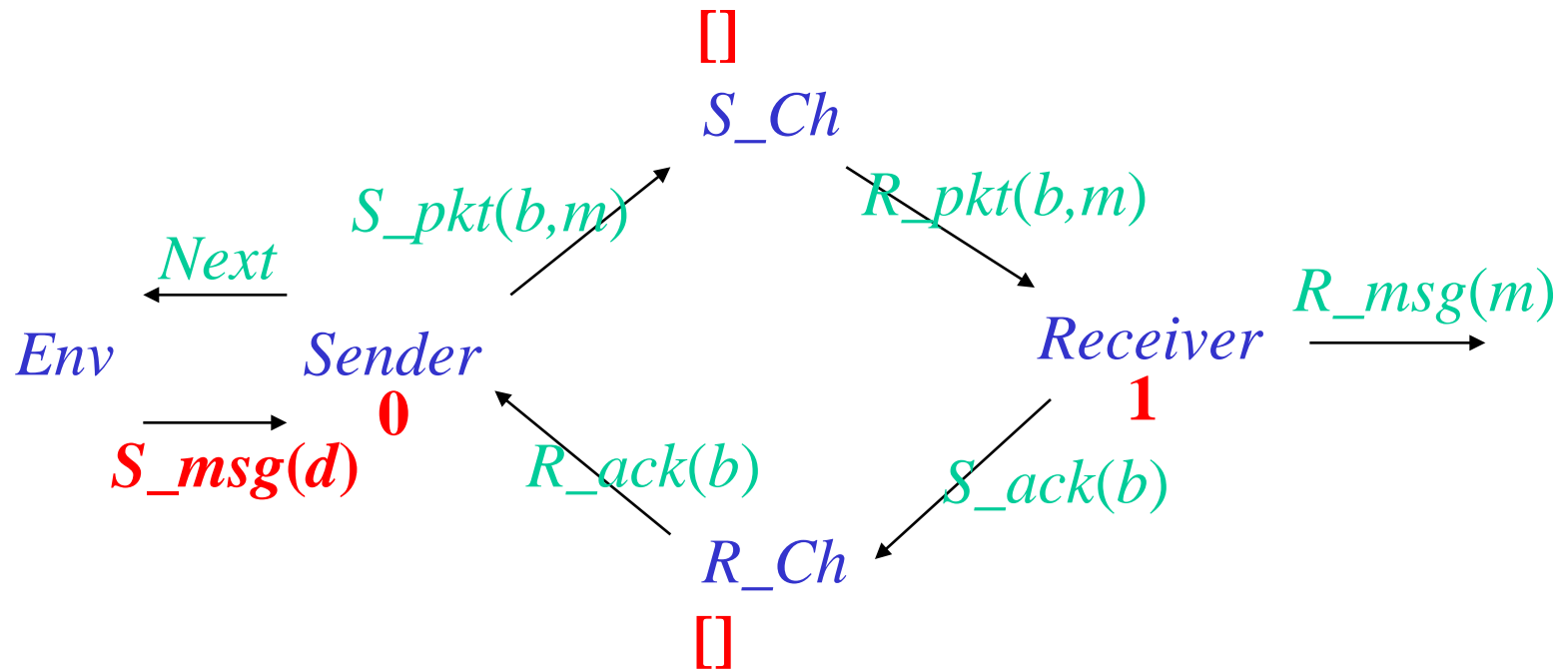
# Alternating Bit Protocol



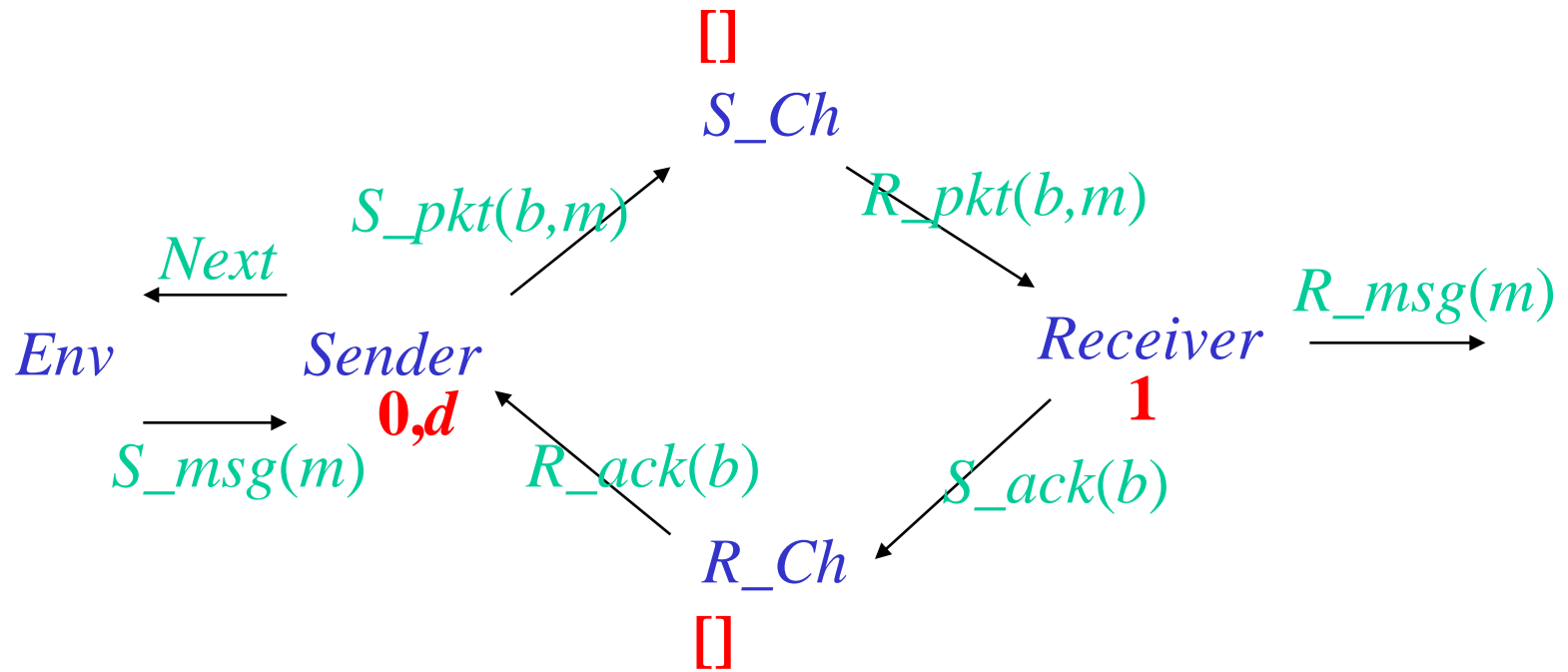
# Alternating Bit Protocol



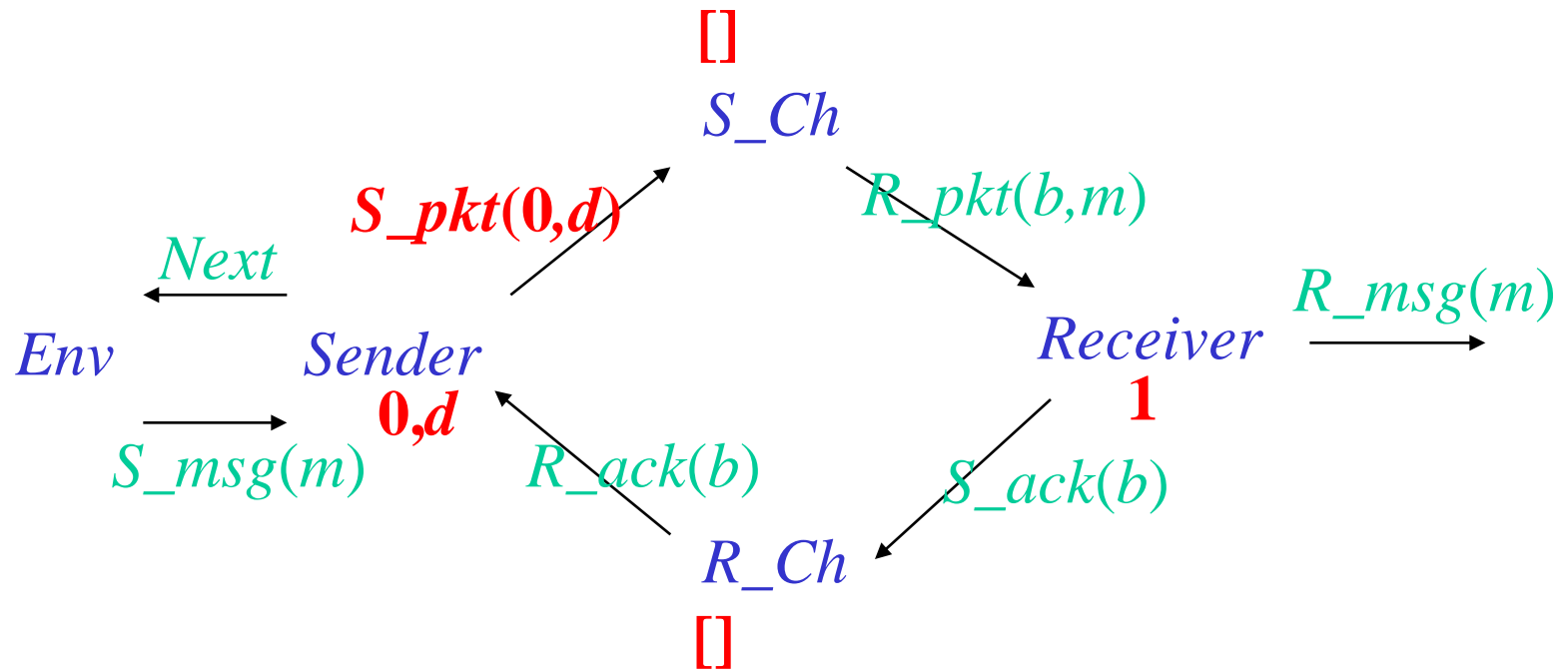
# Alternating Bit Protocol



# Alternating Bit Protocol

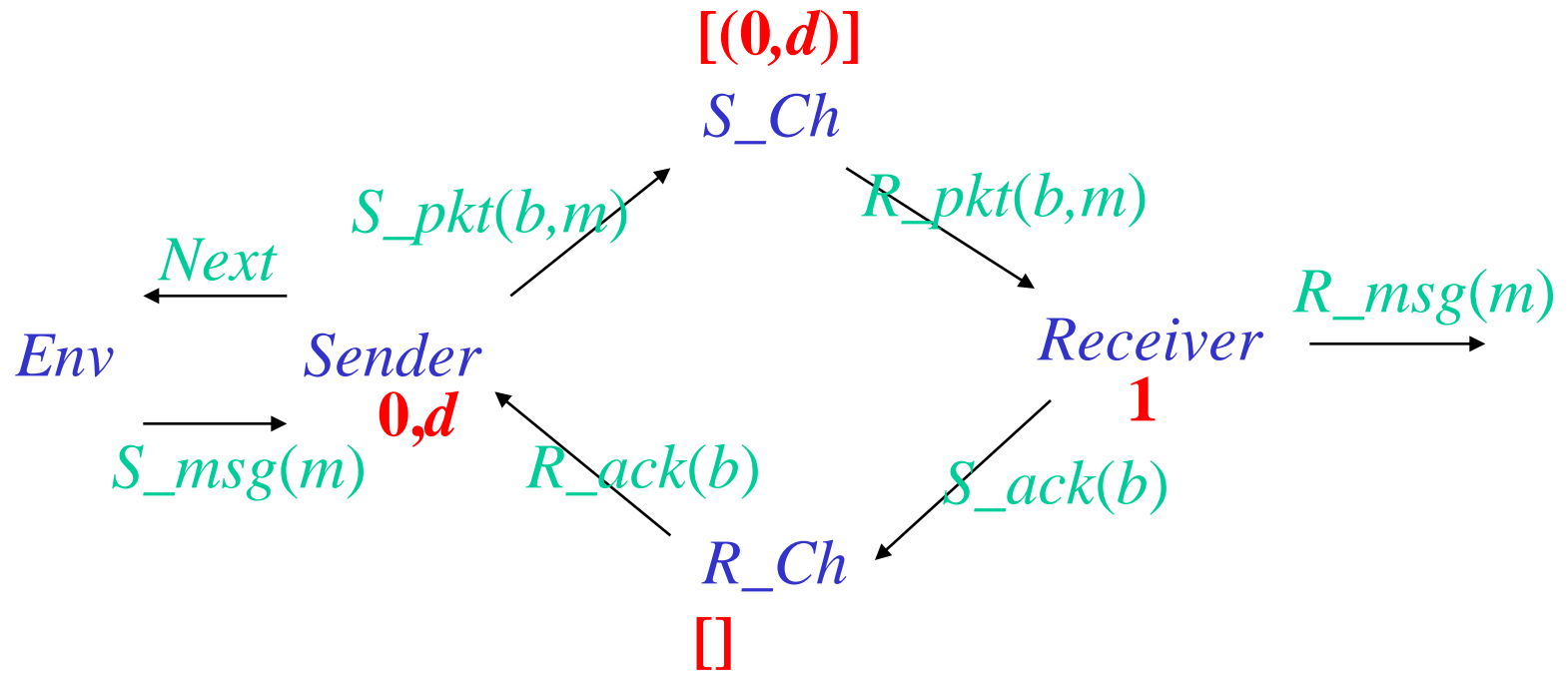


# Alternating Bit Protocol

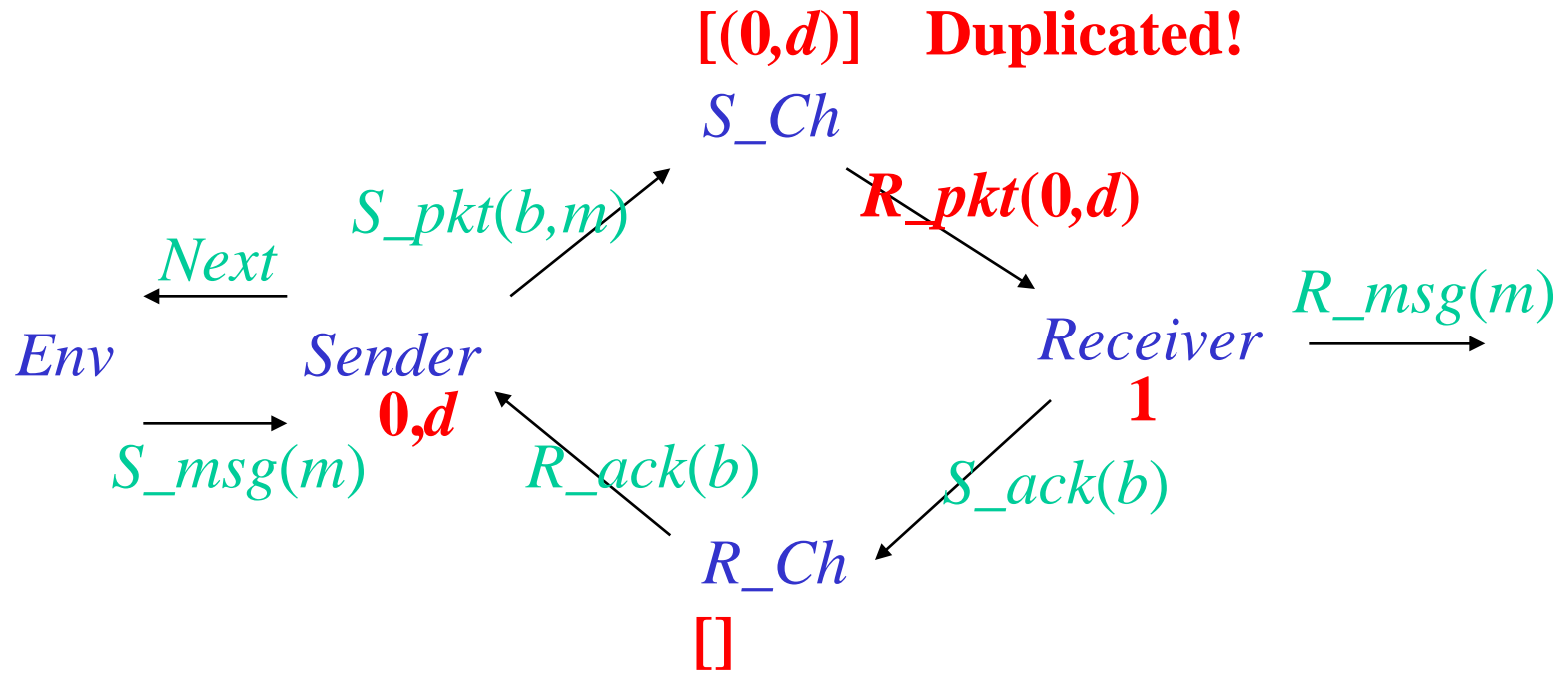




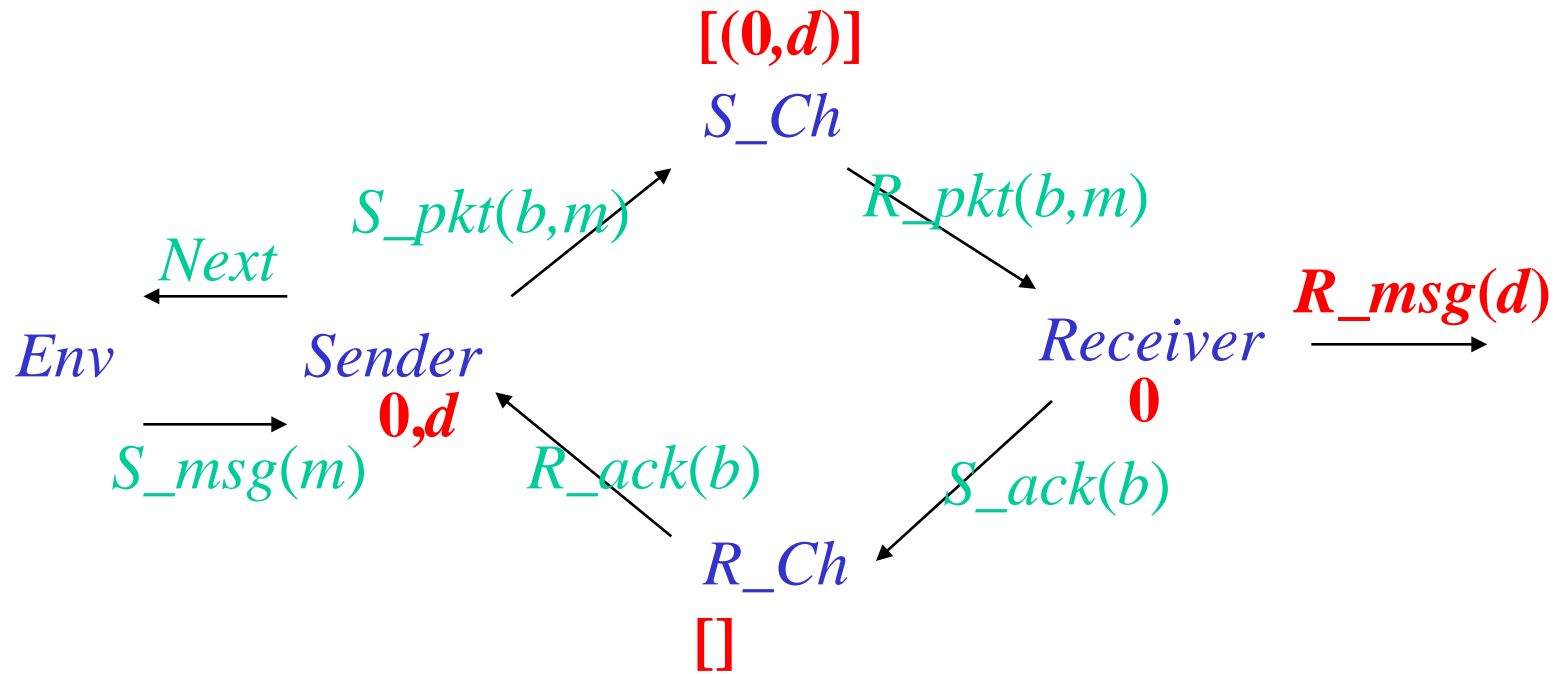
# Alternating Bit Protocol



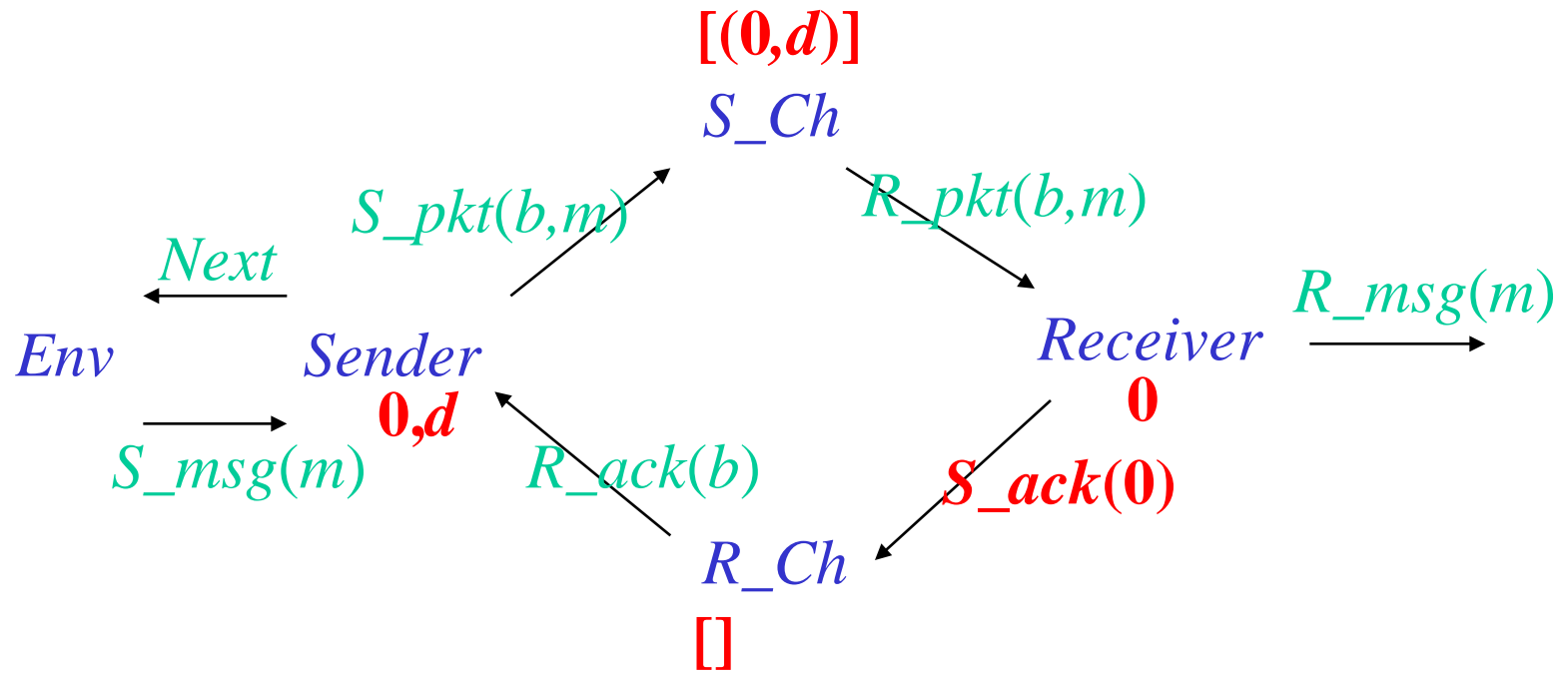
# Alternating Bit Protocol



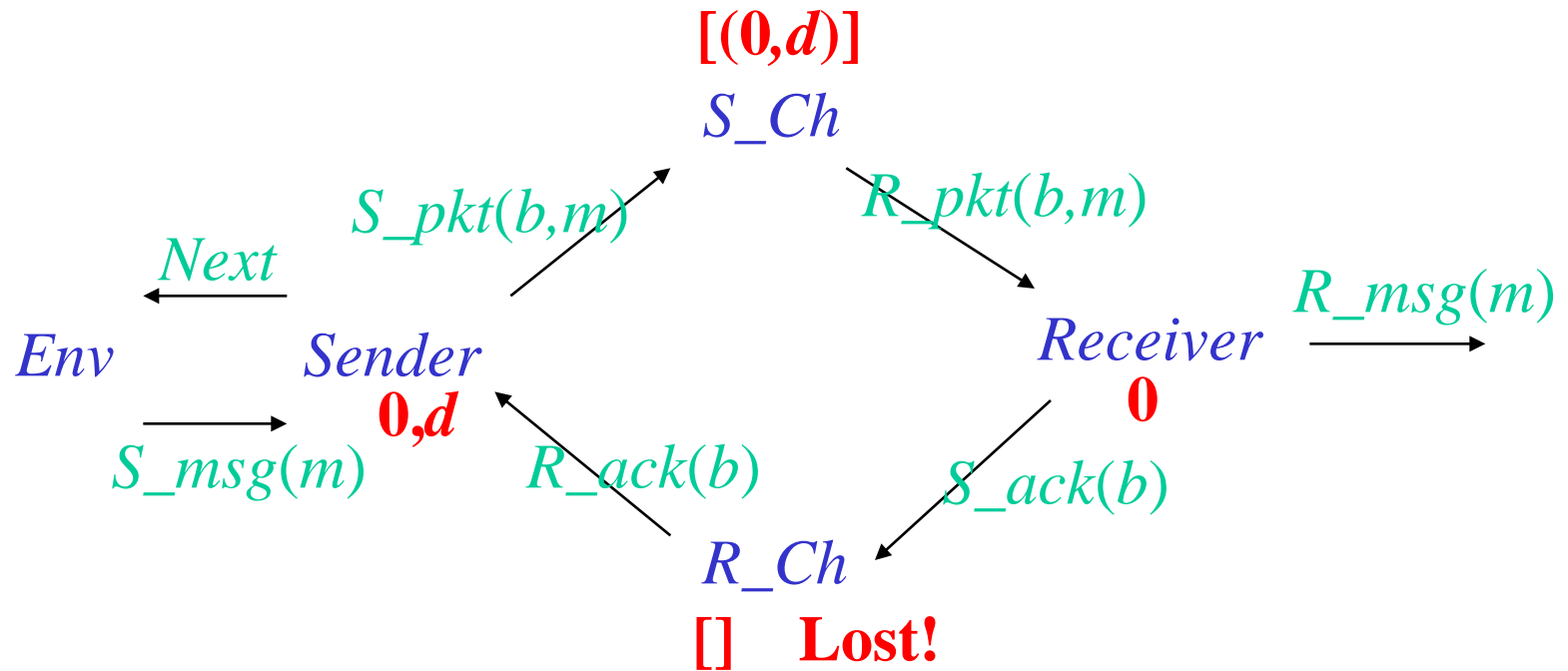
# Alternating Bit Protocol



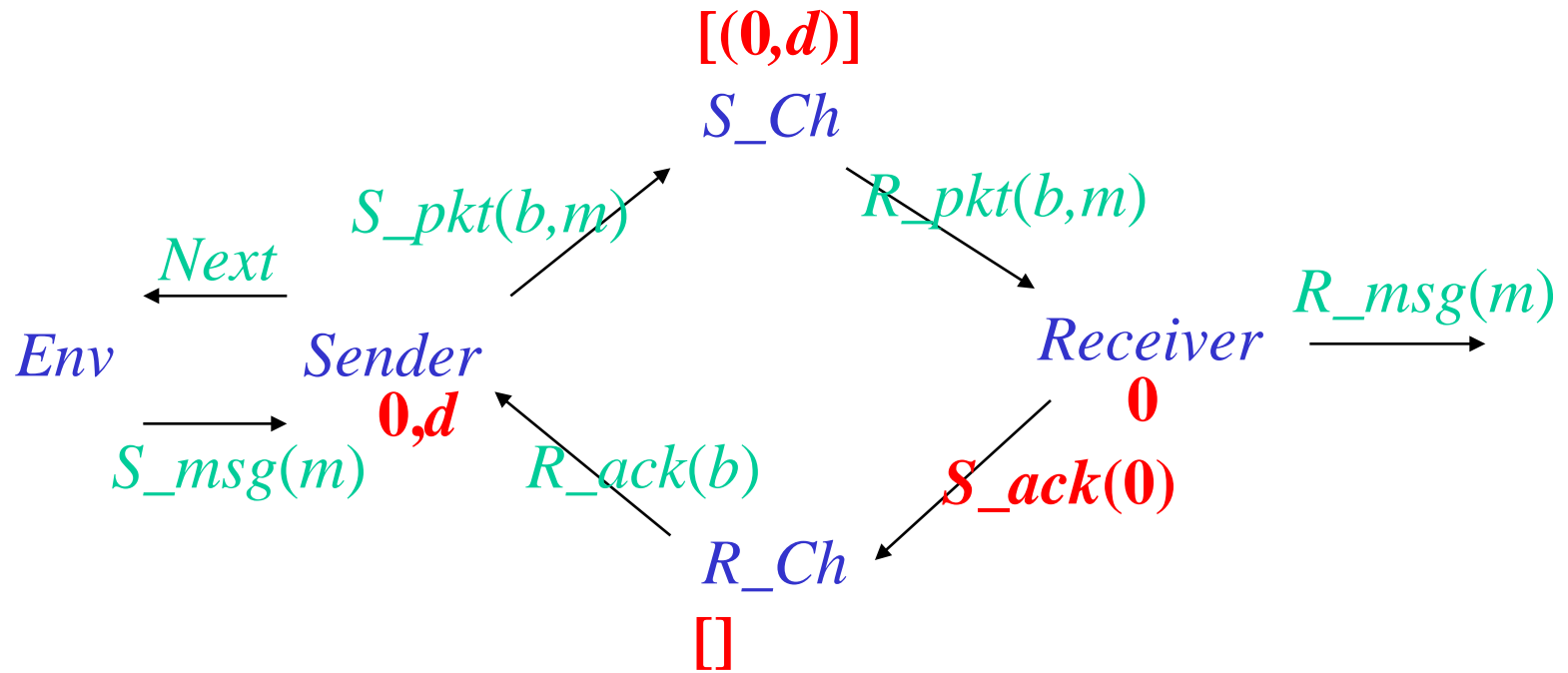
# Alternating Bit Protocol



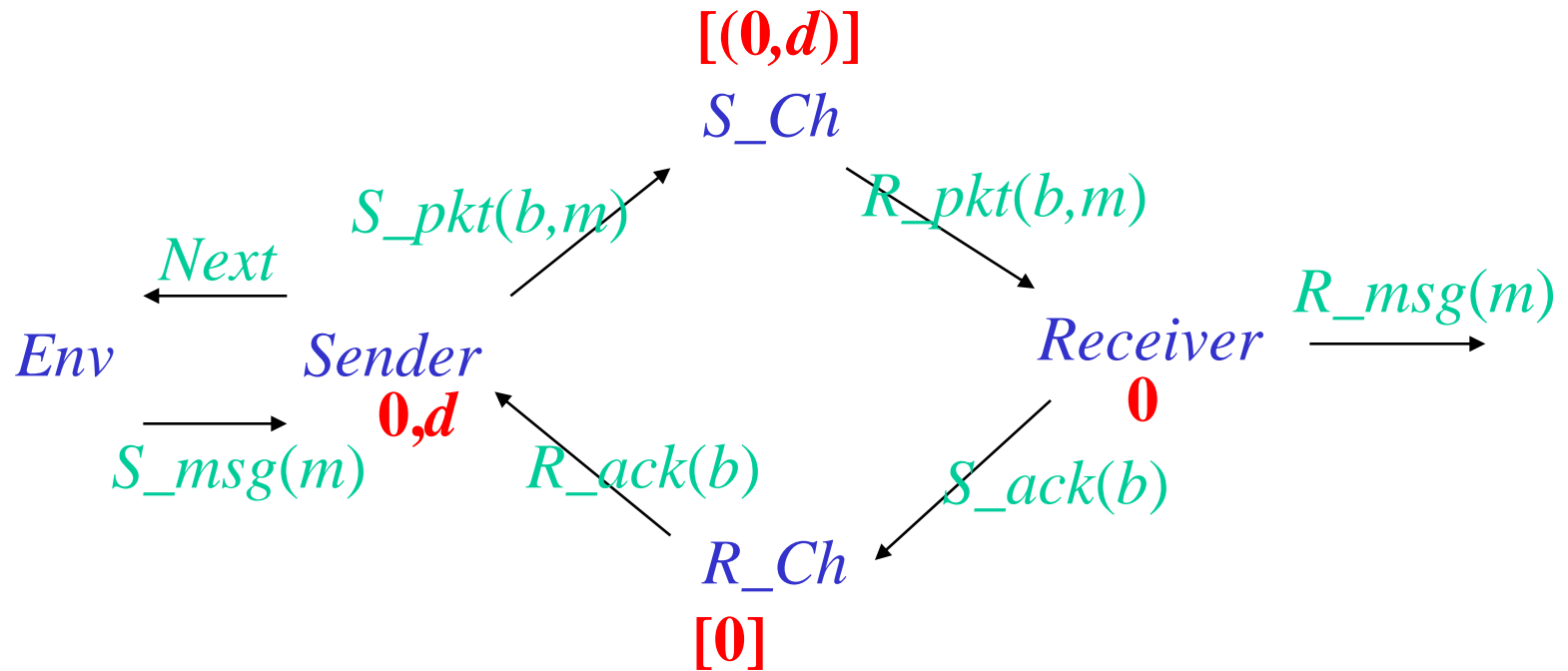
# Alternating Bit Protocol



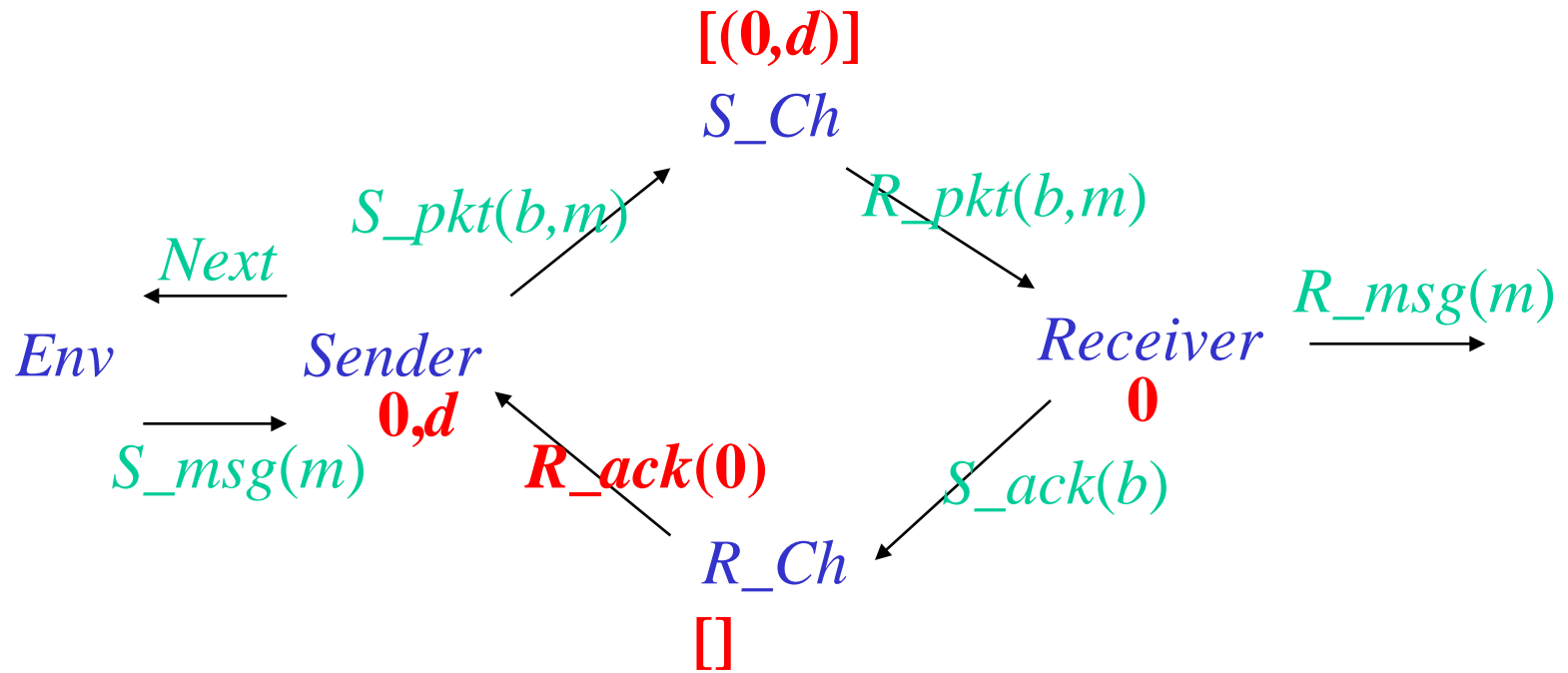
# Alternating Bit Protocol



# Alternating Bit Protocol

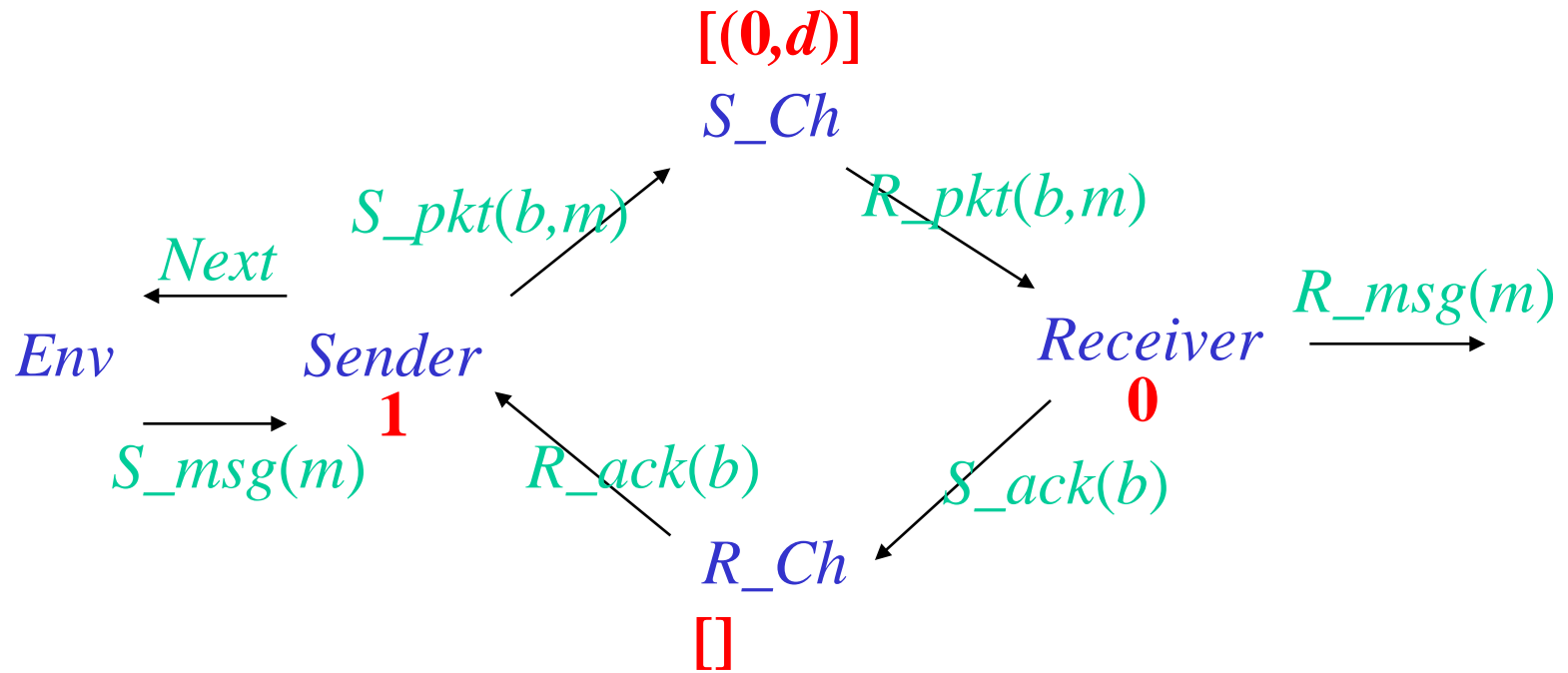


# Alternating Bit Protocol

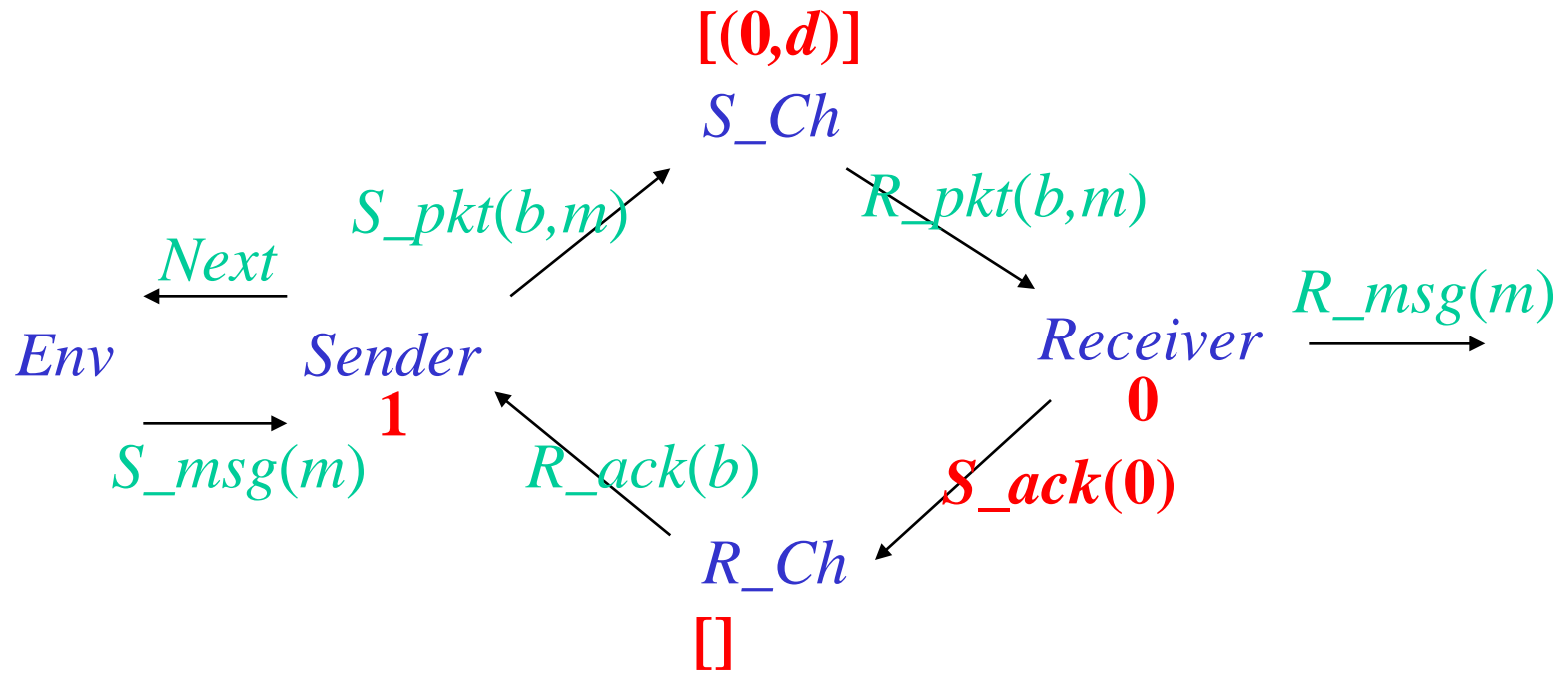




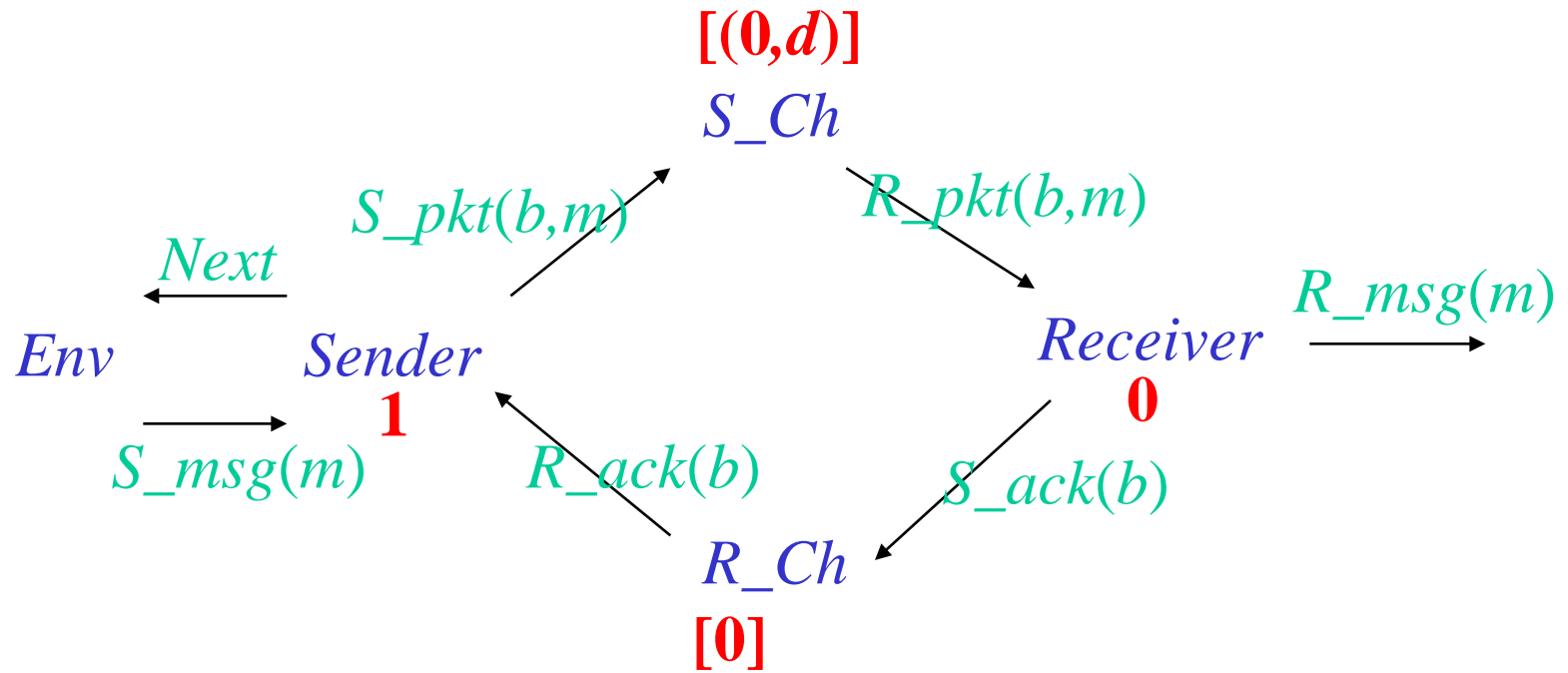
# Alternating Bit Protocol



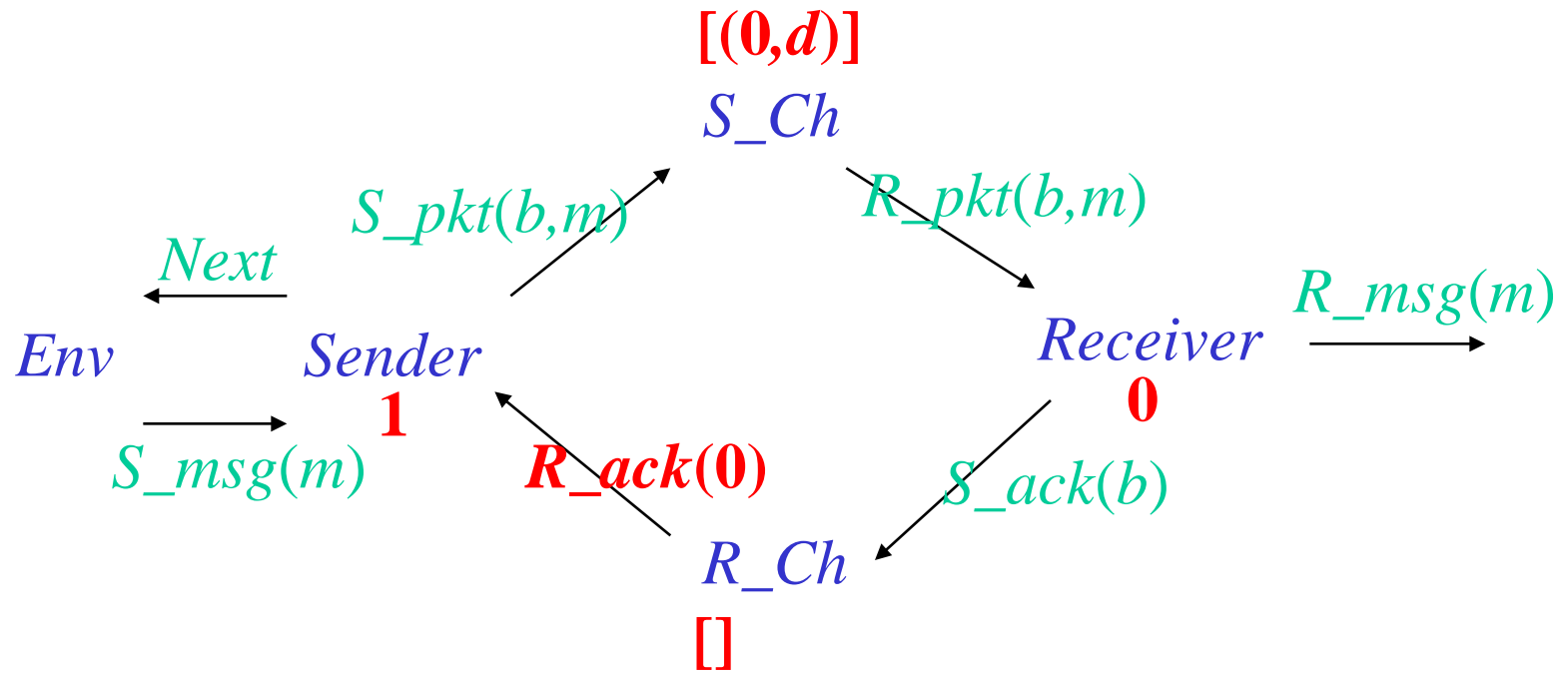
# Alternating Bit Protocol



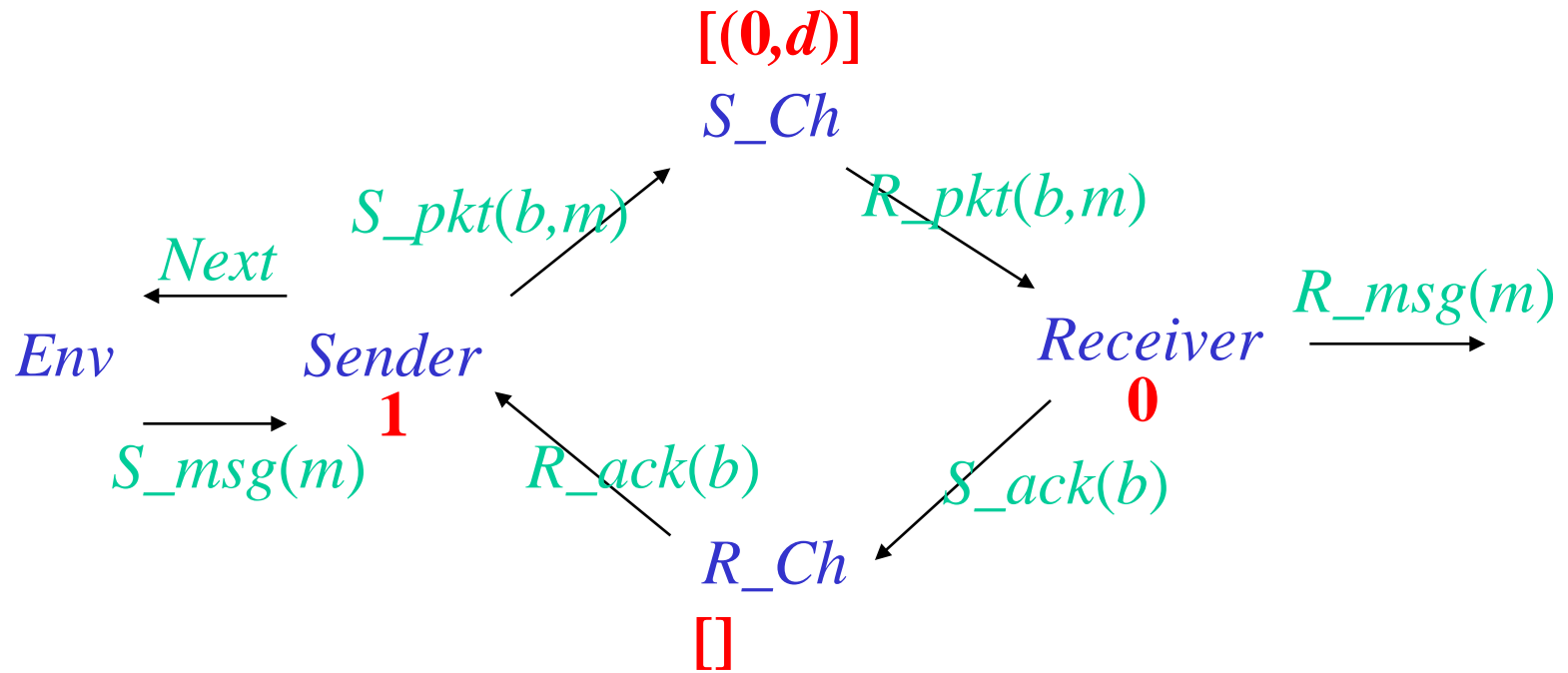
# Alternating Bit Protocol



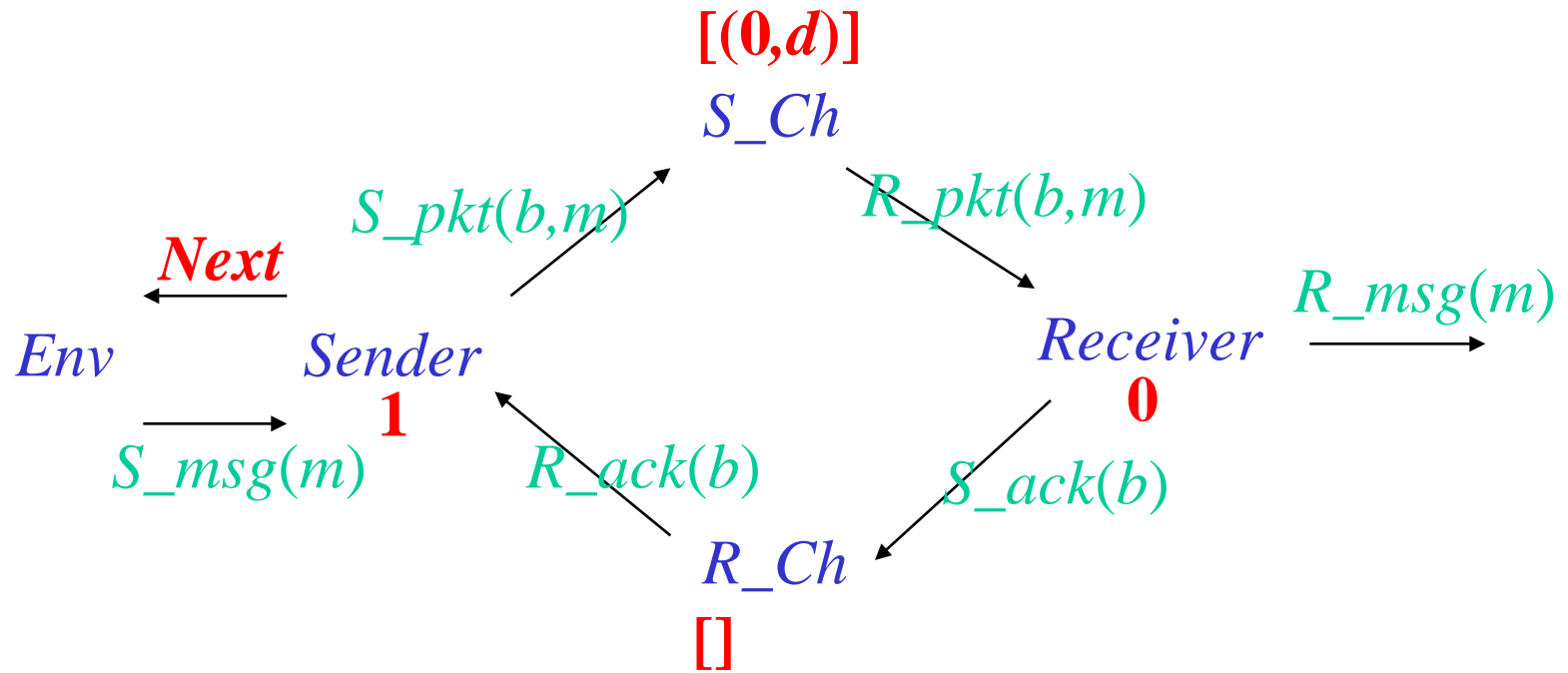
# Alternating Bit Protocol



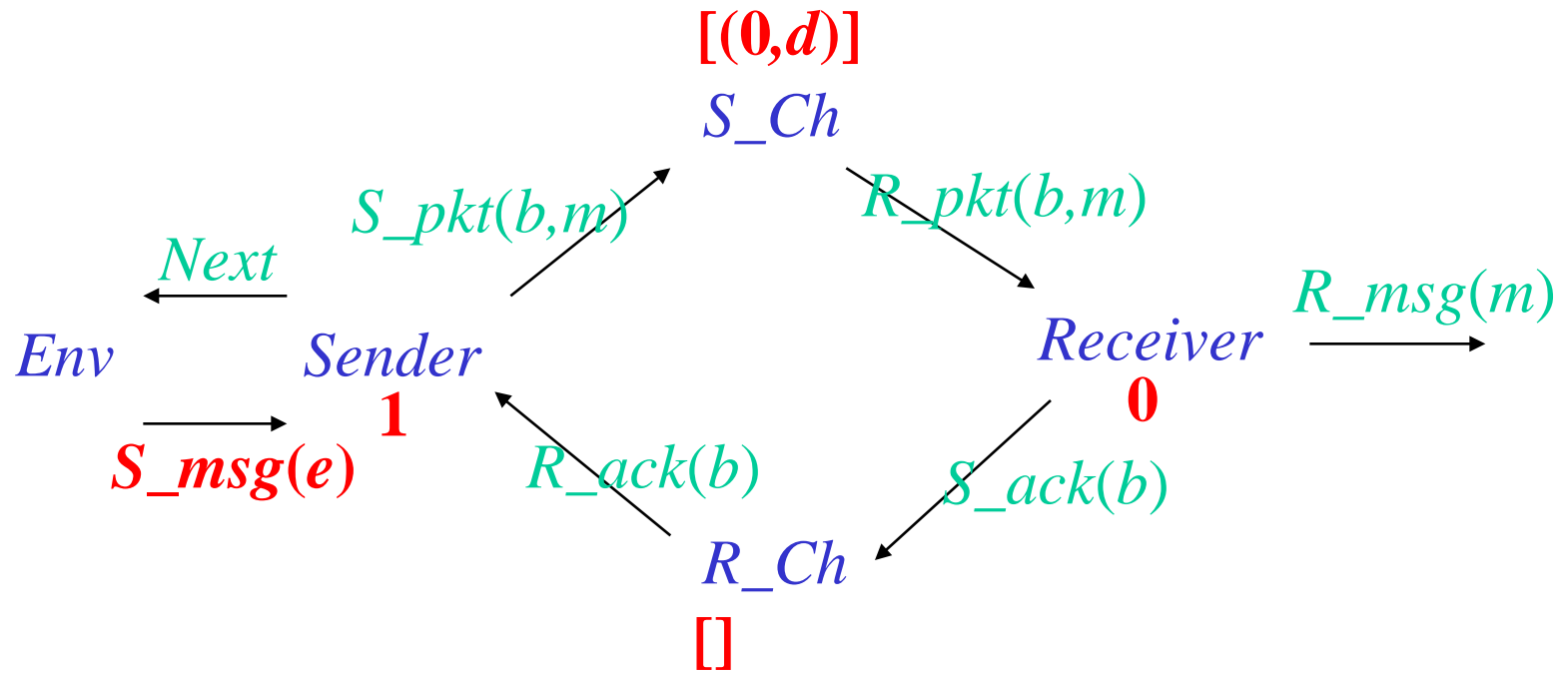
# Alternating Bit Protocol



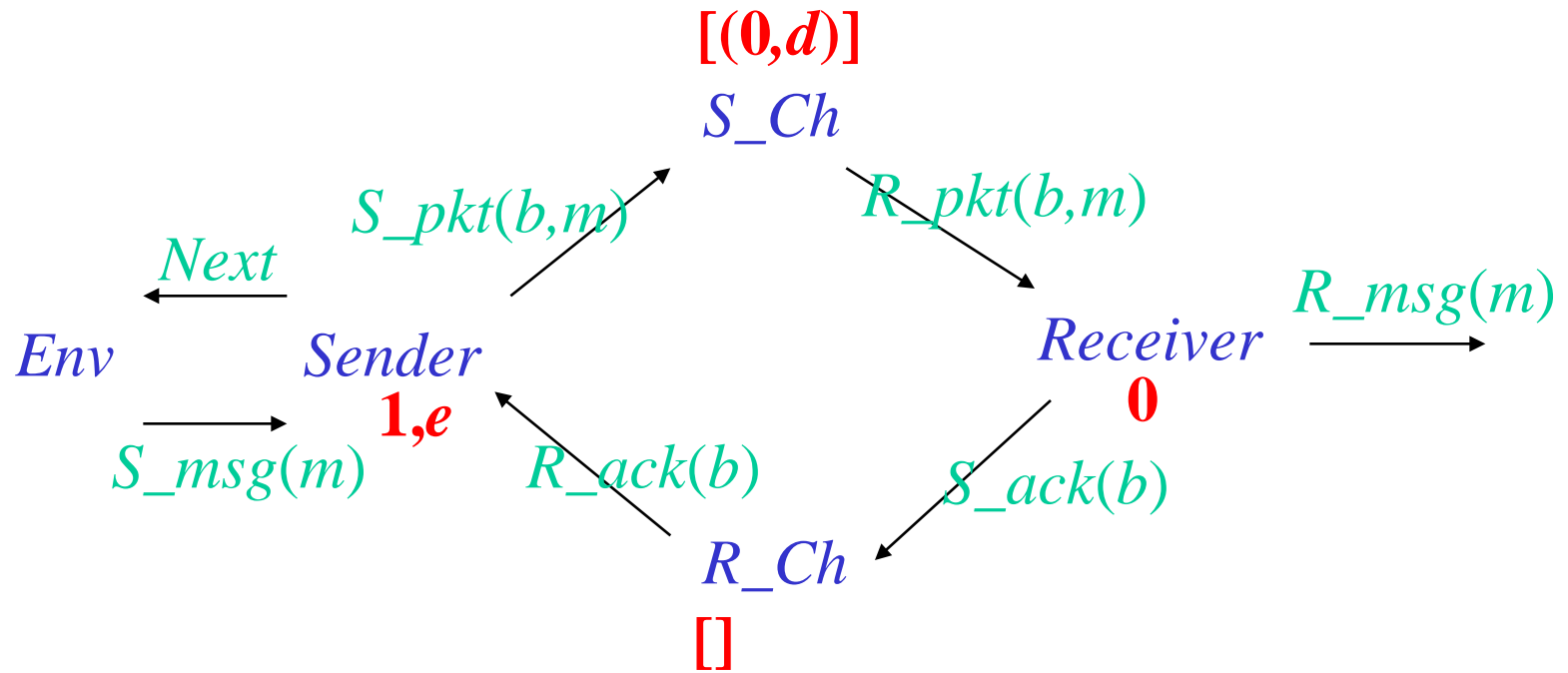
# Alternating Bit Protocol



# Alternating Bit Protocol

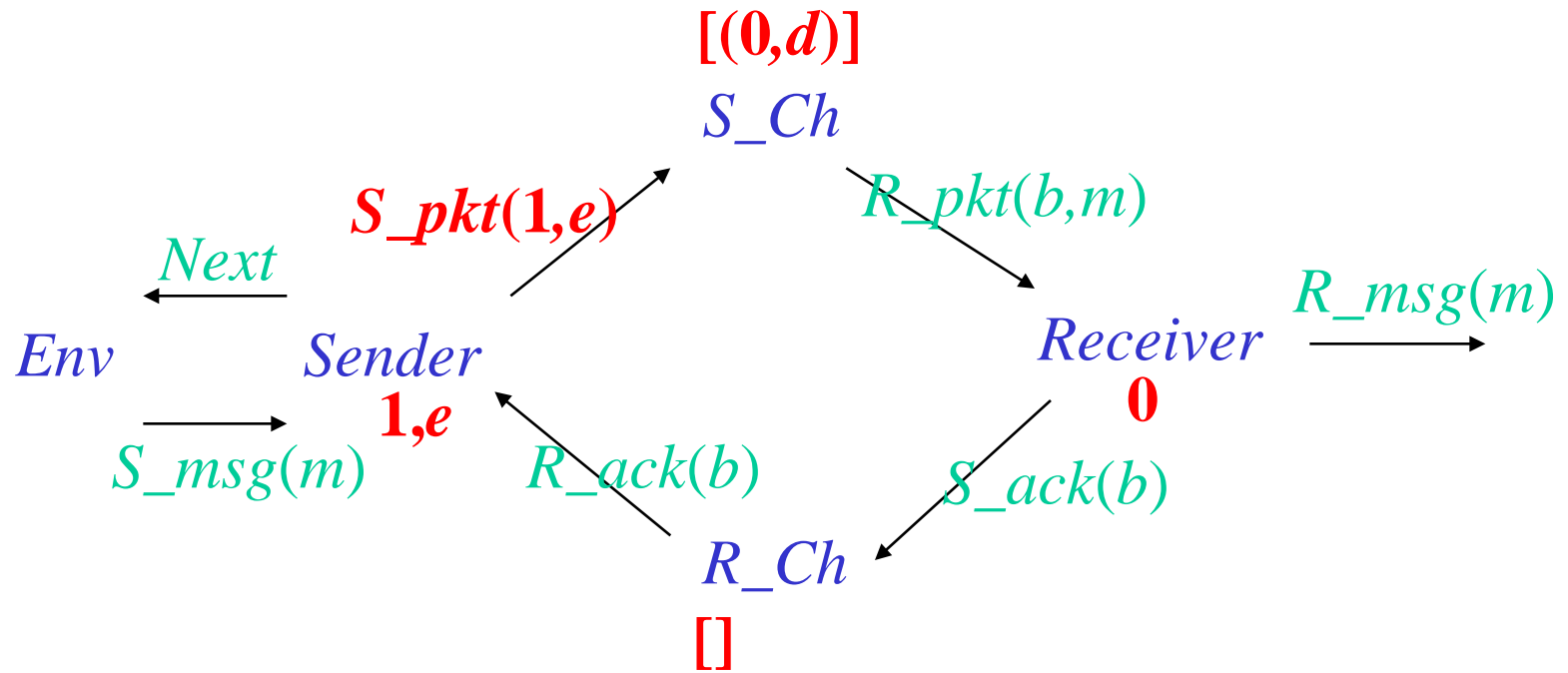


# Alternating Bit Protocol

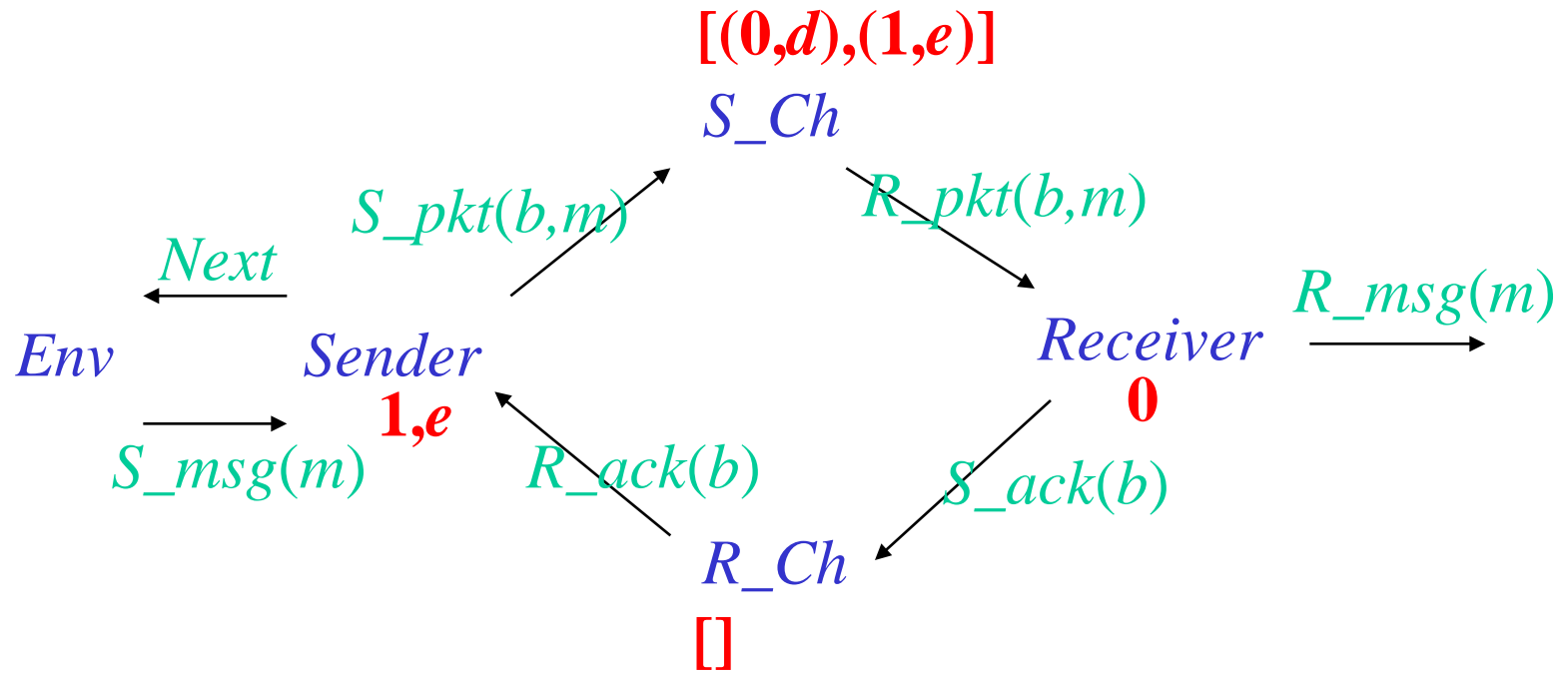




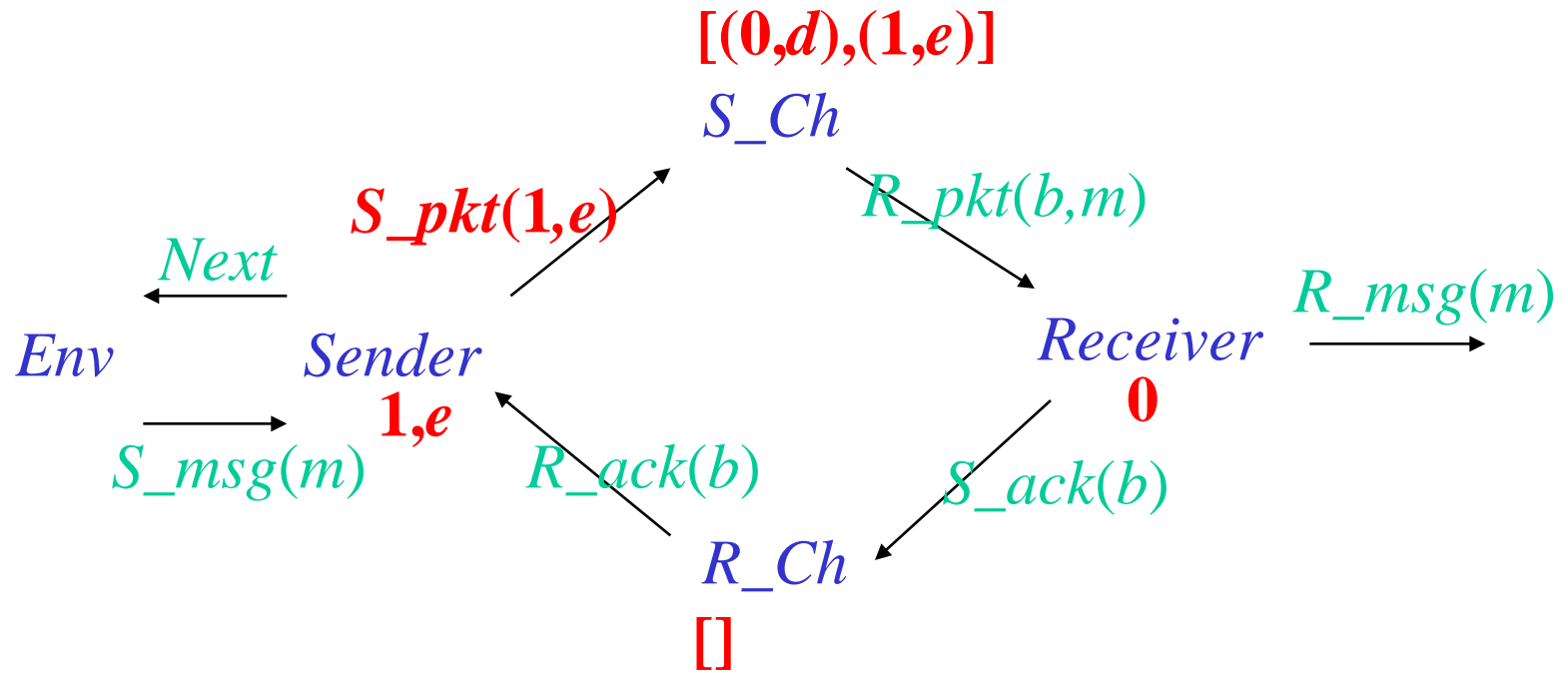
# Alternating Bit Protocol



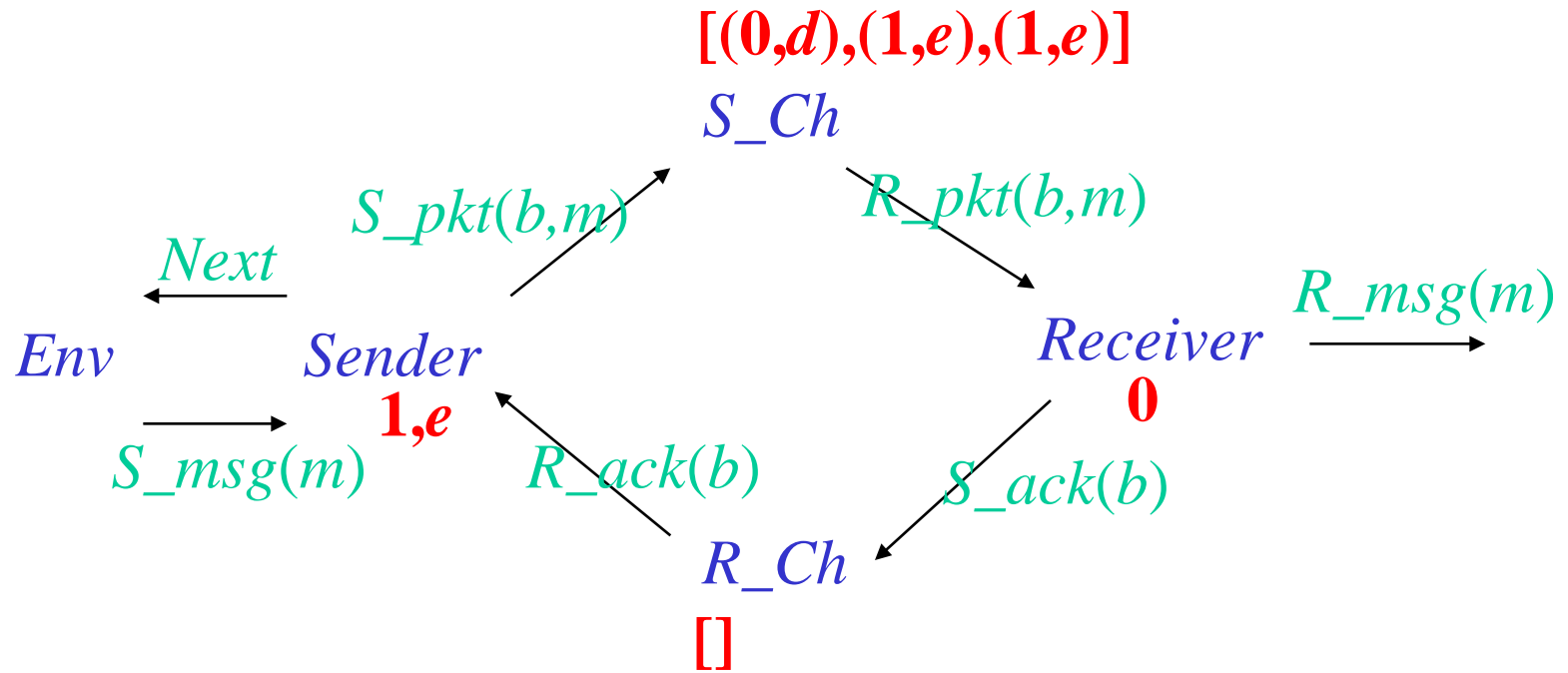
# Alternating Bit Protocol



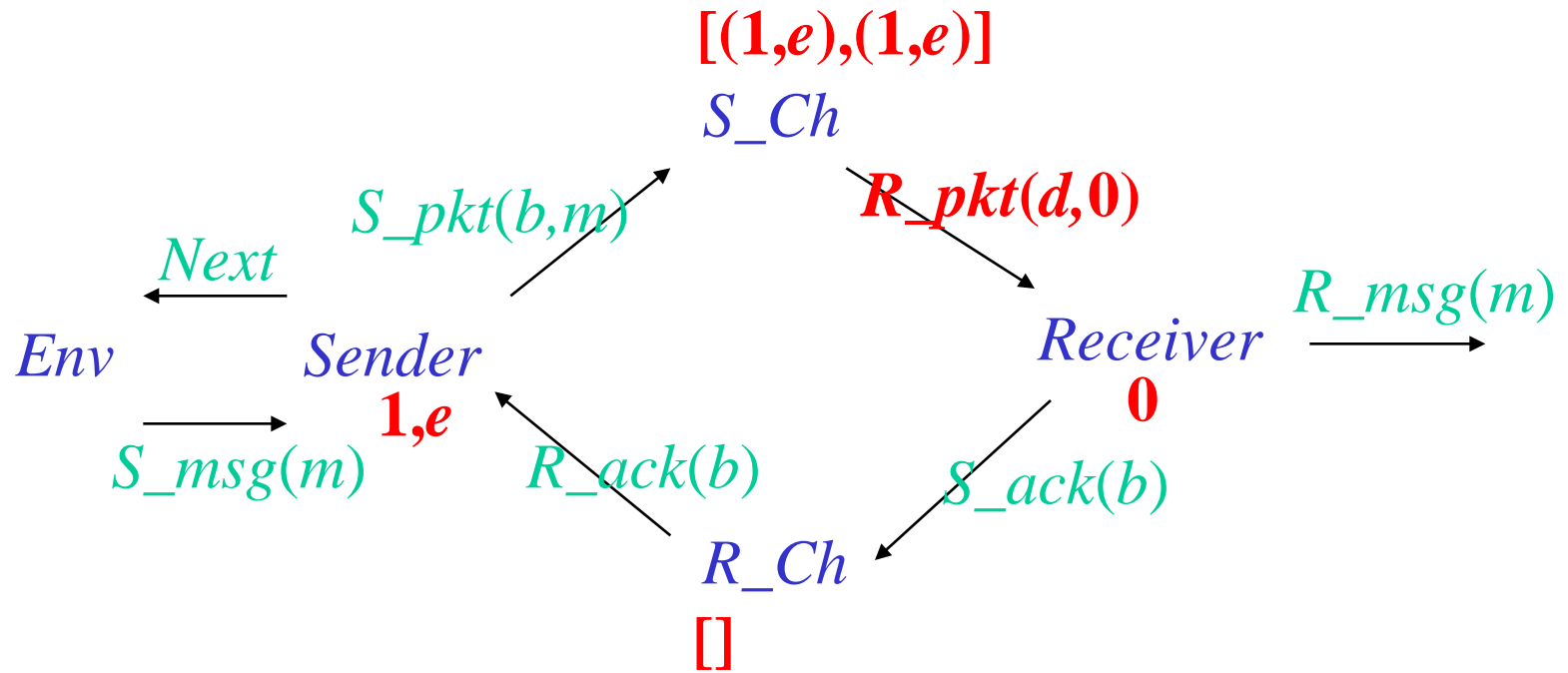
# Alternating Bit Protocol



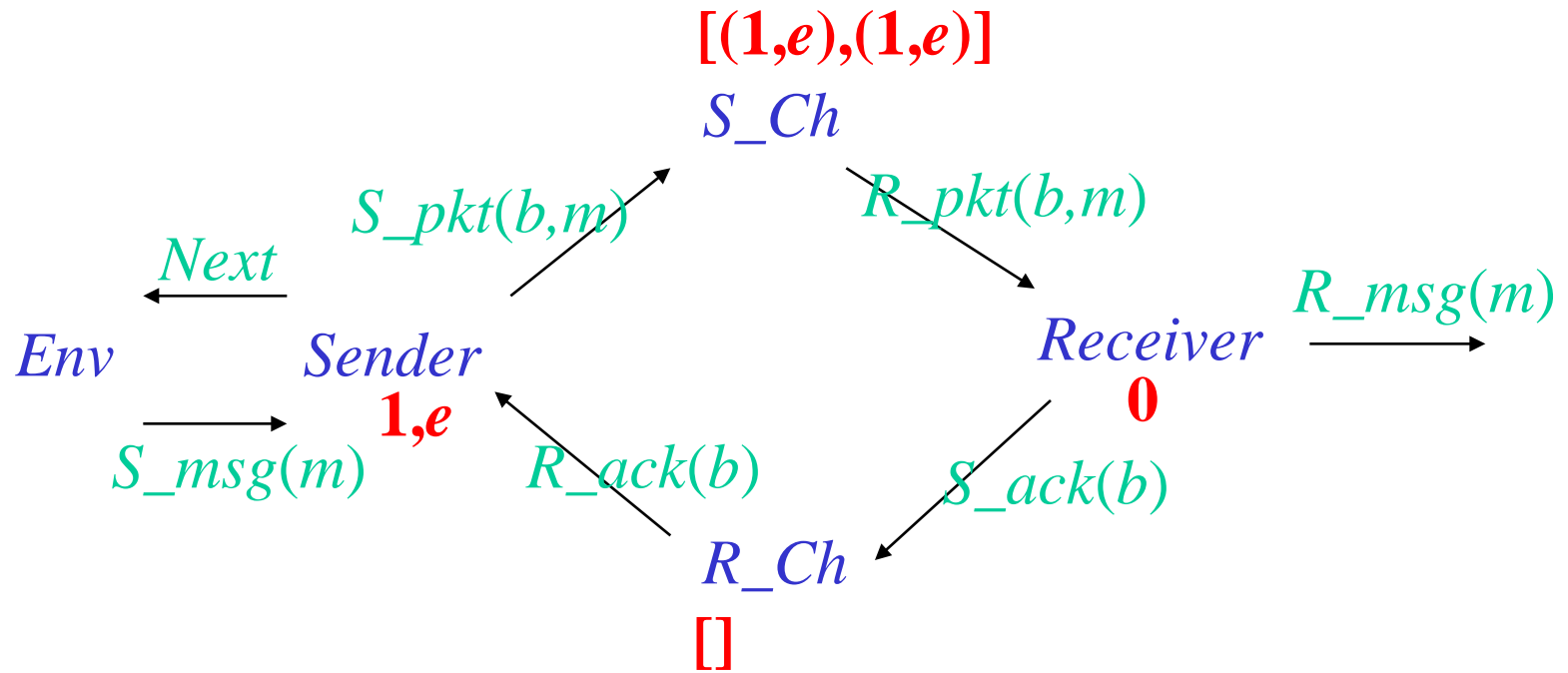
# Alternating Bit Protocol



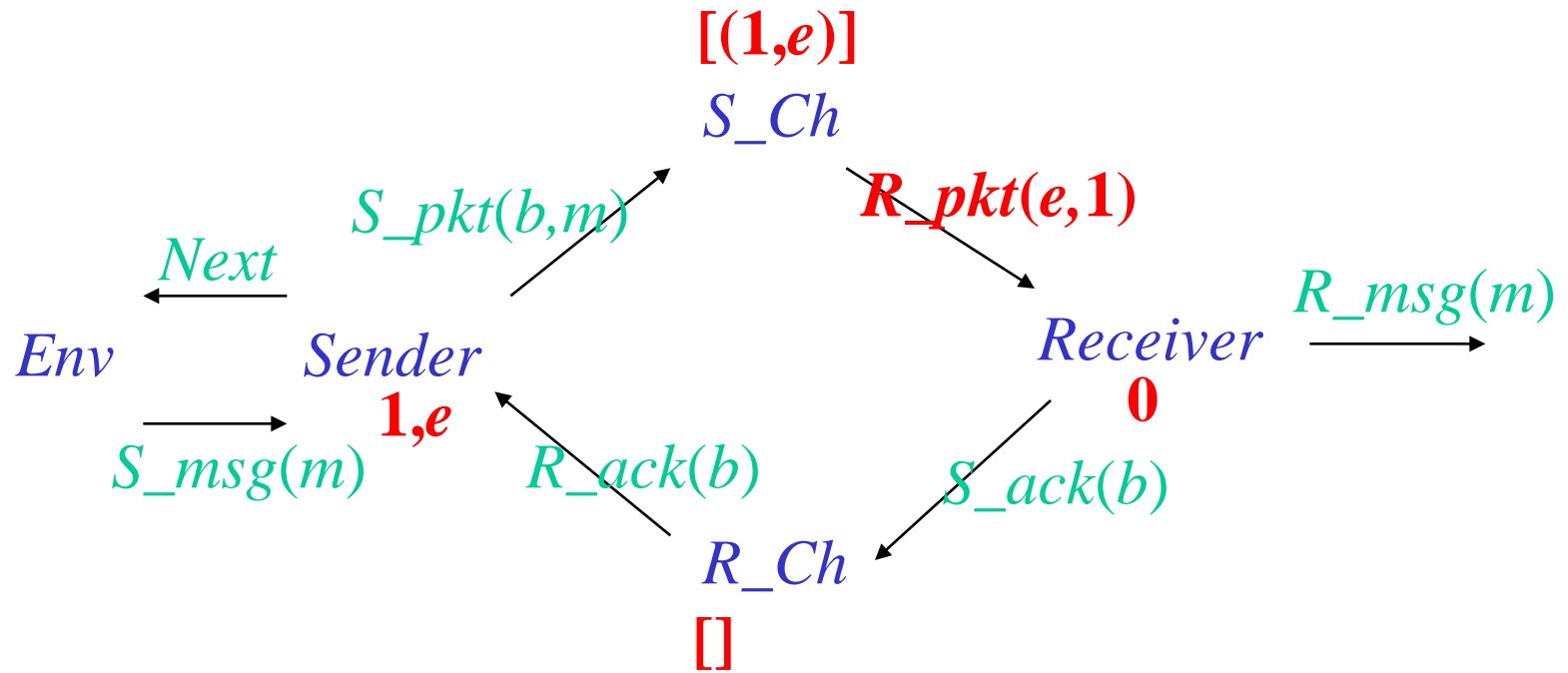
# Alternating Bit Protocol



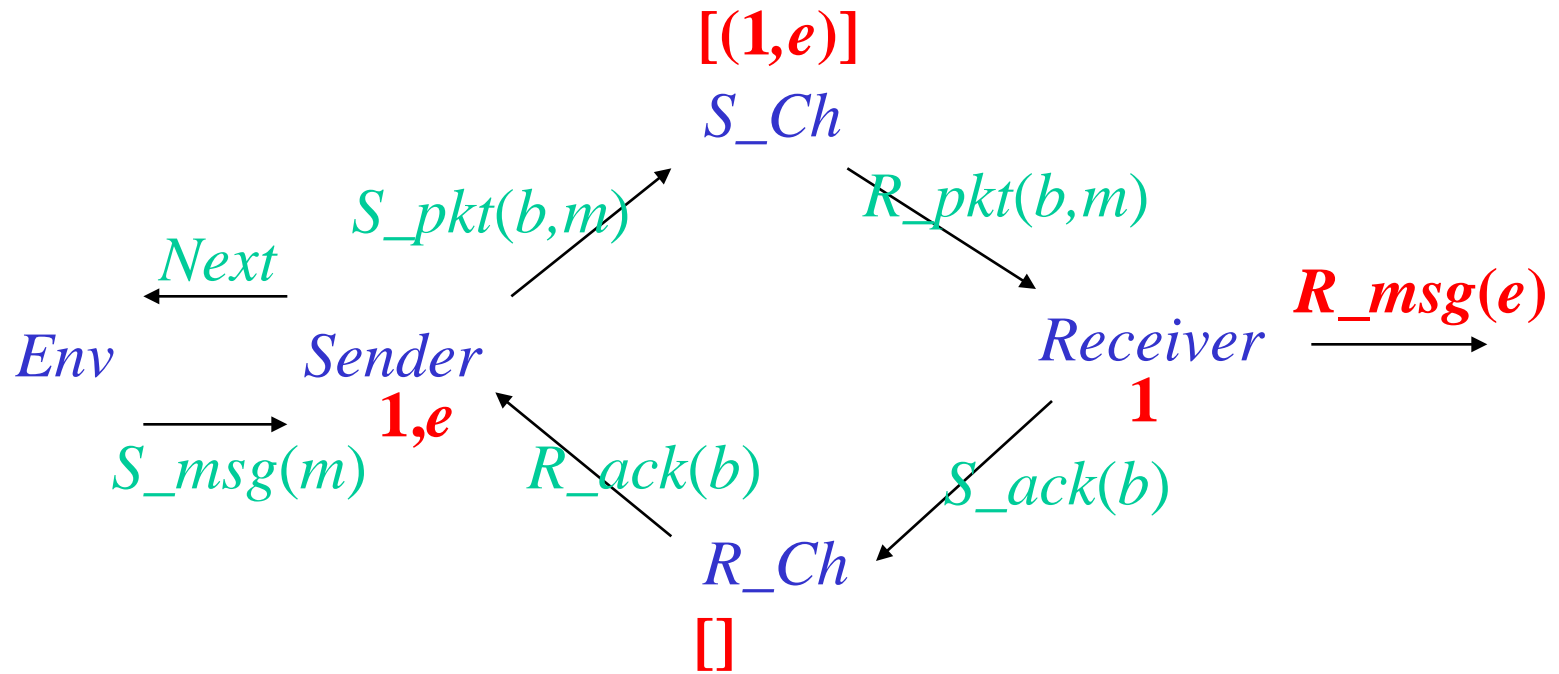
# Alternating Bit Protocol



# Alternating Bit Protocol

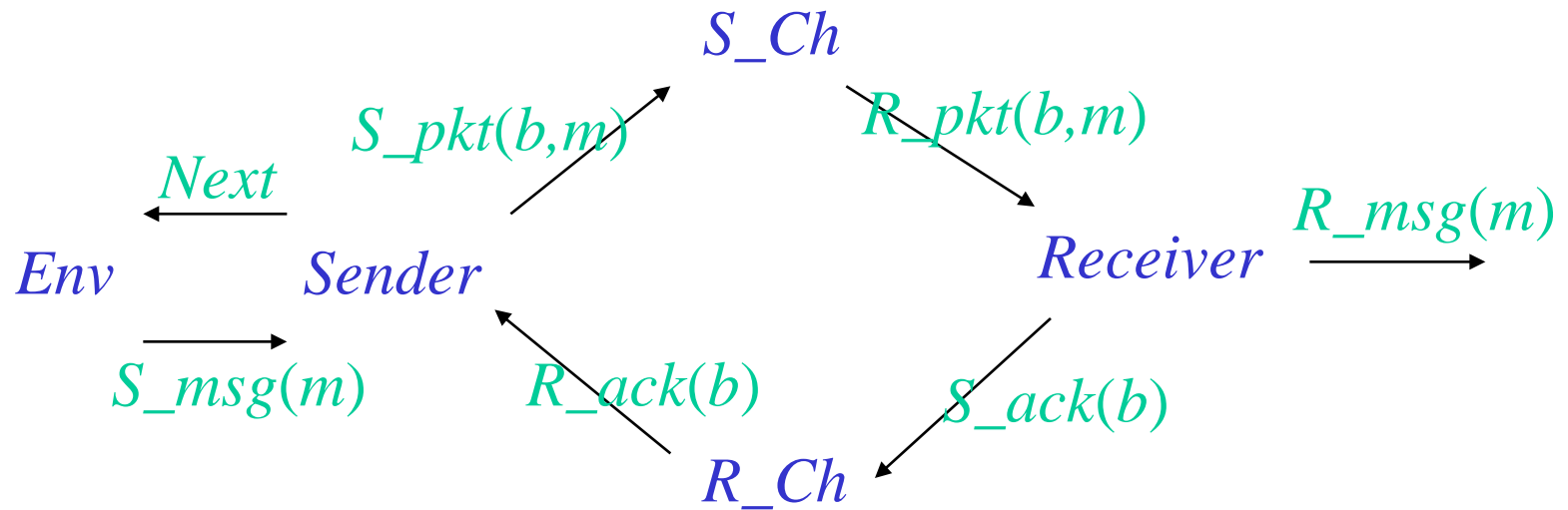


# Alternating Bit Protocol





# Alternating Bit Protocol



*Sender* state: *message* : *None* or *Some(m)*

*header* : 0 or 1

trans.: • if *message*=*None* then

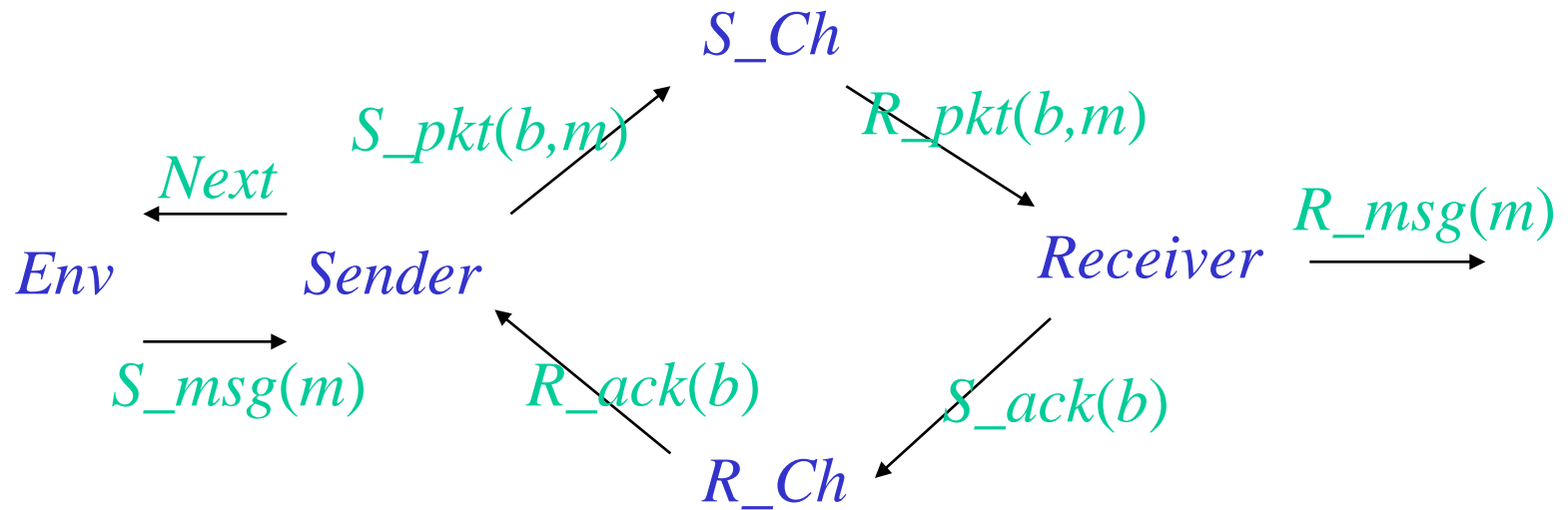
send  $Next$  ; get  $S\_msg(m)$  ; *message* := *Some(m)*

• if *message*=*Some(m)* then send  $S\_pkt(header,m)$

• when  $R\_ack(b)$  is received

if  $b=header$  then flip *header* ; *message* := *None*

# Alternating Bit Protocol



## Receiver

state: *header* : 0 or 1

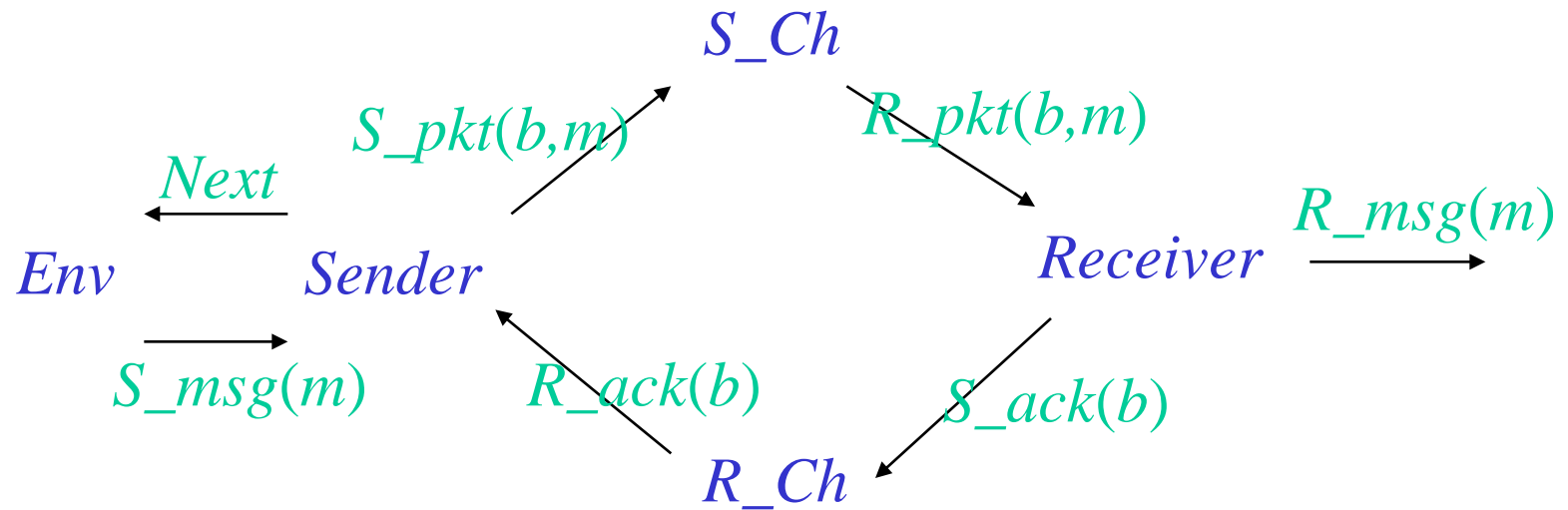
trans.: • when  $R\_pkt(b, m)$  is received

if  $b \neq header$  then

send  $R\_msg(m)$  ; flip *header*

• send  $S\_ack(header)$

# Alternating Bit Protocol



**S\_Ch** state: queue of messages

- trans.:
- when  $S\_pkt(b,m)$  is received  
enqueue  $m$  or ignore  $m$  ( $m$  is lost)
  - if the queue is not empty  
let  $m$  be the first message ; send  $R\_pkt(b,m)$   
dequeue  $m$  or do not dequeue  $m$  ( $m$  is duplicated)

# Concrete States

- Queues may grow unlimitedly
  - An element of a queue may be duplicated indefinitely
- The concrete state space is obviously infinite

# Abstraction of Channels

- Reducing queues by removing duplicated elements in a queue
  - For example,  $[d,d,d,d,e,e,e,f]$  is reduced to  $[d,e,f]$
- Accordingly, transitions of channels are abstracted

## *Abstract $S\_Ch$*

state: reduced queue of messages

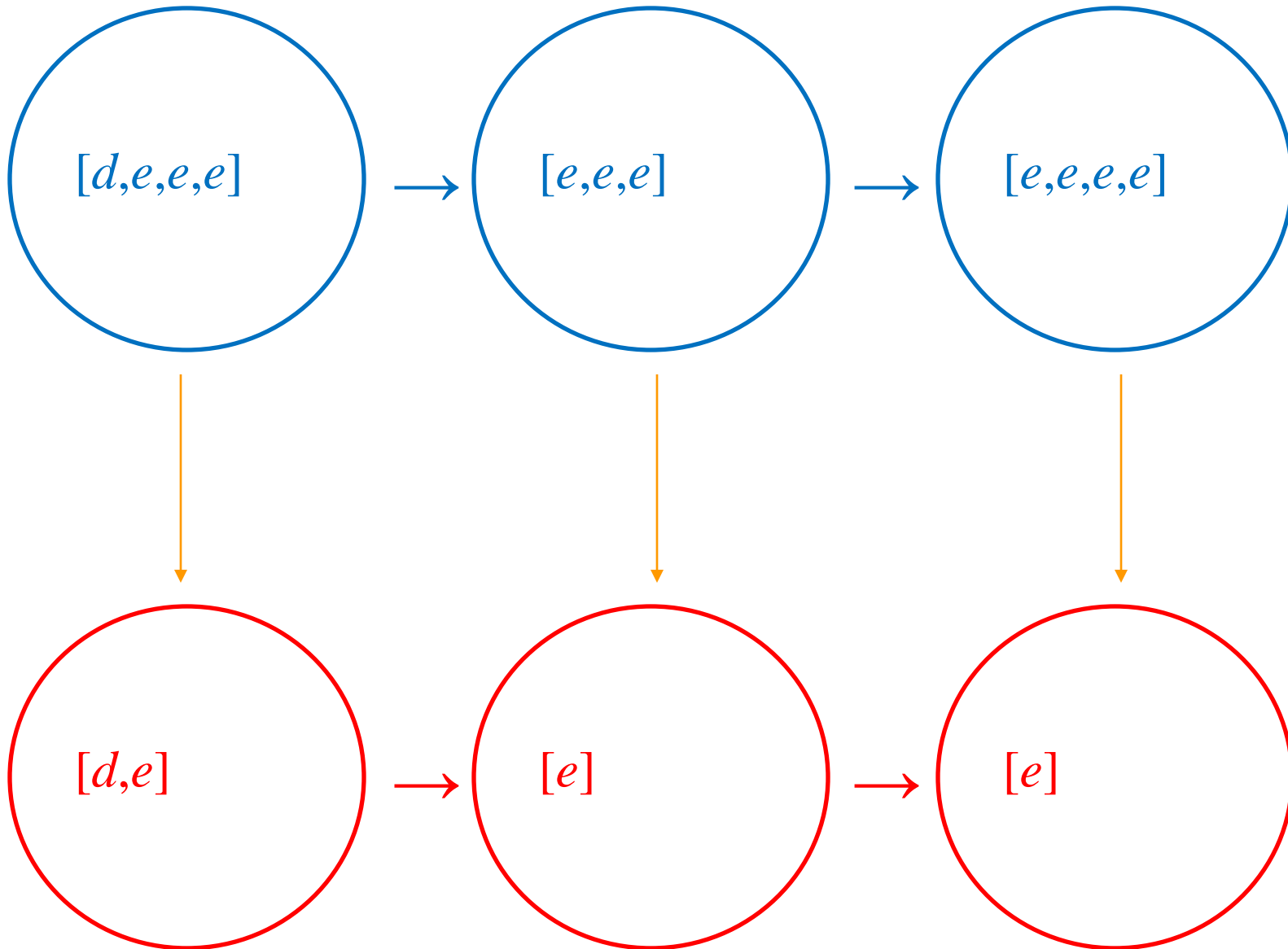
trans.: • when  $S\_pkt(b,m)$  is received

enqueue  $m$  ; reduce or ignore  $m$  ( $m$  is lost)

• if the queue is not empty

let  $m$  be the first message ; send  $R\_pkt(b,m)$

dequeue  $m$  or do not dequeue  $m$  ( $m$  is duplicated)



# Abstract States

- Queues in channels are reduced
- Even though the abstract state space is still infinite in this example,
- Only a finite number of abstract states are reachable from an initial state
  - The length of a reduced queue in a channel is at most 2

# Simulation

- If  $t$  is an abstraction of  $s$  and  $s \rightarrow s'$  then there exists an abstraction  $t'$  of  $s'$  such that  $t \rightarrow t'$  (if  $s \rightarrow s'$  then  $\alpha(s) \rightarrow \alpha(s')$ )

Concrete path

$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$

Abstract path

$t_0$



# Simulation

- If  $t$  is an abstraction of  $s$  and  $s \rightarrow s'$  then there exists an abstraction  $t'$  of  $s'$  such that  $t \rightarrow t'$  (if  $s \rightarrow s'$  then  $\alpha(s) \rightarrow \alpha(s')$ )

Concrete path

$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$

Abstract path

$t_0 \rightarrow t_1$

# Simulation

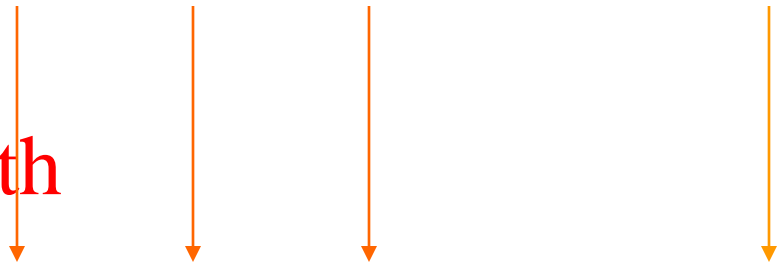
- If  $t$  is an abstraction of  $s$  and  $s \rightarrow s'$  then there exists an abstraction  $t'$  of  $s'$  such that  $t \rightarrow t'$  (if  $s \rightarrow s'$  then  $\alpha(s) \rightarrow \alpha(s')$ )

Concrete path

$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_n$

Abstract path

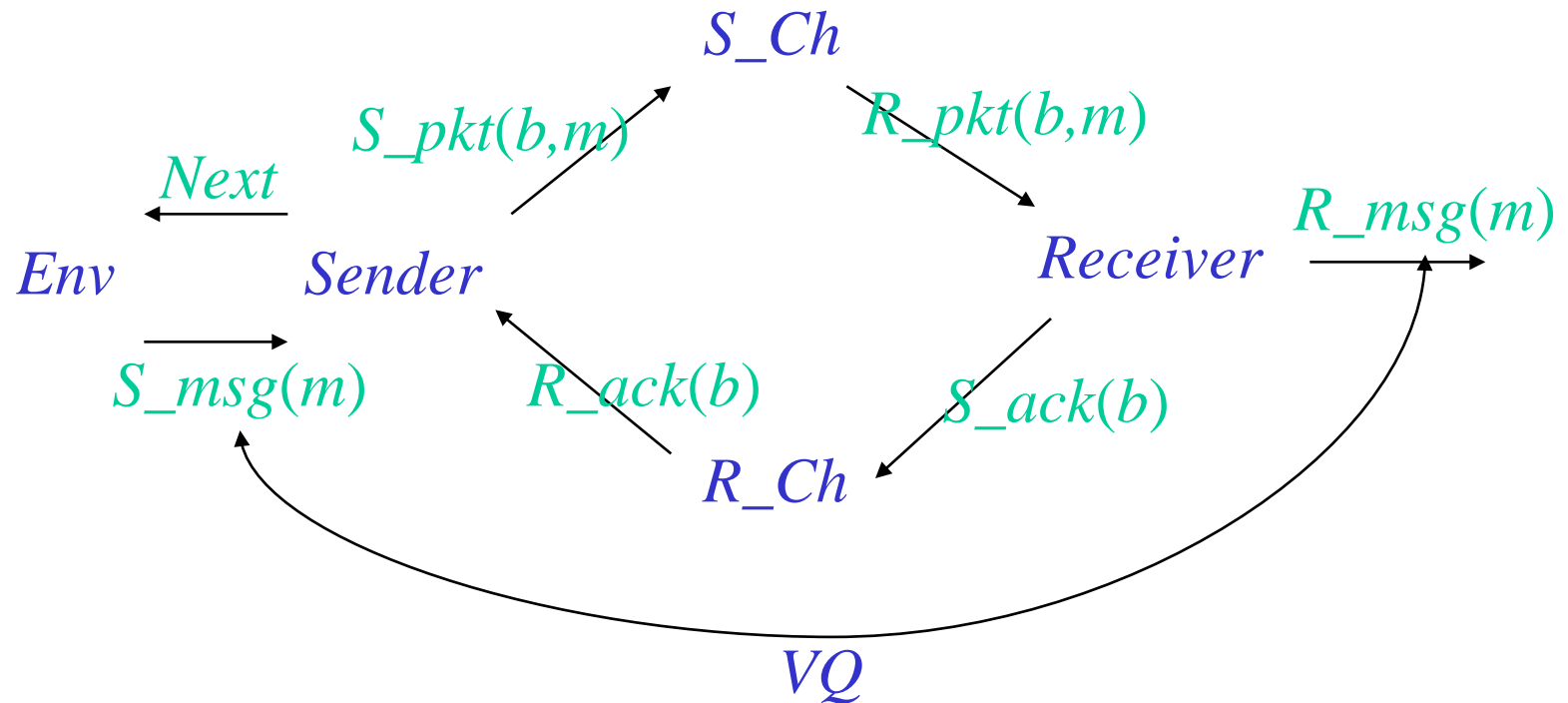
$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n$



# Abstract Model Checking

- If a property holds in any abstract path, then the corresponding property holds in any concrete path
- Example
  - After a transition by  $S\_msg(m)$ , if there exists a transition by  $R\_msg(m')$ , then  $m=m'$

# Alternating Bit Protocol



After a transition by  $S\_msg(m)$ , if there exists a transition by  $R\_msg(m')$ , then  $m=m'$   
(This property is expressed by a virtual queue ( $VQ$ ) that stores  $S\_msg(m)$  and compares it with  $R\_msg(m')$ )

# Microsoft Static Driver Verifier (SDV)

- Hostile model of the driver's execution environment
  - Harness code simulates the operating system initializing and invoking the device driver
  - Stub code provides the semantics for the kernel APIs
- SLAM Toolkit
  - CEGAR
- API usage rules (properties)
  - About 60

# SLAM Toolkit

- Safety verification of system software
  - Target: Device drivers for Windows with well defined interface
- Three phases
  - C2BP: tool for translating (abstracting) C programs to Boolean programs, using predicates in specifications (API usage rules)
  - BEBOP: model checker for Boolean programs
  - NEWTON: tool that analyzes error paths produced by the model checker, and discovers predicates for refining Boolean programs

# CEGAR

- Counter
- Example-
- Guided
- Abstraction
- Refinement

```
state {  
    enum { Unlocked=0, Locked=1 }  
    state = Unlocked;  
}
```

```
KeAcquireSpinLock.return {  
    if (state == Locked)  
        abort;  
    else  
        state = Locked;  
}
```

```
KeReleaseSpinLock.return {  
    if (state == Unlocked)  
        abort;  
    else  
        state = Unlocked;  
}
```

# SLIC spec. Specification Language for Interface Checking



```
enum { Unlocked=0, Locked=1 }
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ;
}

void KeAcquireSpinLock_return {
    if (state == Locked)
        slic_abort();
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```

C program  
obtained by  
compiling  
SLIC spec.

```

void example() {
    do {
        KeAcquireSpinLock();

        nPacketsOld = nPackets;
        req = devExt->WLHV;
        if(req && req->status) {
            devExt->WLHV = req->Next;
            KeReleaseSpinLock();

            irp = req->irp;
            if(req->status > 0){
                irp->IoS.Status = SUCCESS;
                irp->IoS.Info = req->Status;
            } else {
                irp->IoS.Status = FAIL;
                irp->IoS.Info = req->Status;
            }
            SmartDevFreeBlock(req);
            IoCompleteRequest(irp);
            nPackets++;
        }
    } while(nPackets!=nPacketsOld);
    KeReleaseSpinLock();
}

```

Code for a  
device  
driver

```

void example() {
    do {
        KeAcquireSpinLock();
A:KeAcquireSpinLock_return();
        nPacketsOld = nPackets;
        req = devExt->WLHV;
        if(req && req->status) {
            devExt->WLHV = req->Next;
            KeReleaseSpinLock();
B: KeReleaseSpinLock_return();
            irp = req->irp;
            if(req->status > 0){
                irp->IoS.Status = SUCCESS;
                irp->IoS.Info = req->Status;
            } else {
                irp->IoS.Status = FAIL;
                irp->IoS.Info = req->Status;
            }
            SmartDevFreeBlock(req);
            IoCompleteRequest(irp);
            nPackets++;
        }
    } while(nPackets!=nPacketsOld);
    KeReleaseSpinLock();
C:KeReleaseSpinLock_return();
}

```

Device driver  
code with  
inserted checks  
for specification

```

decl {state==Locked}, {state==Unlocked};

void slic_abort() begin slic_abort; skip; end
      Boolean var.           Boolean var.
void KeAcquireSpinLock_return
begin
  if ({state==Locked})
    slic_abort();
  else
    {state==Locked}, {state==Unlocked} := T,F;
begin

void KeReleaseSpinLock_return
begin
  if ({state==Unlocked})
    slic_abort();
  else
    {state==Locked}, {state==Unlocked} := F,T;
end

```

Boolean program obtained from SLIC spec.

```
decl bL, bU;

void slic_abort() begin SLIC_ERROR: skip; end

void KeAcquireSpinLock_return
begin
  if (bL)
    slic_abort();
  else
    bL,bU := T,F;
begin

void KeReleaseSpinLock_return
begin
  if (bU)
    slic_abort();
  else
    bL,bU := F,T;
end
```

Boolean program obtained from SLIC spec.

```

void example() begin
  do {
    skip();
    A:KeAcquireSpinLock_return();
    skip;
    skip;
    if(*) then
      skip;
      skip();
    B: KeReleaseSpinLock_return();
    skip;
    if(*) then
      skip;
      skip;
    else
      skip;
      skip;
    fi
    skip;
    skip;
    skip;
  }
  while(*);
  skip();
  C:KeReleaseSpinLock_return();
end

```

Boolean program  
obtained from device  
driver code

\* : undetermined

# Model Checking

- Find a path that reaches “**SLIC\_ERROR**”
- In this case, an error path  
**A, A, SLIC\_ERROR**  
is found
- Verify if the error path is valid with respect to the original C program
  - Verification condition generation (VCGen)
- In this case, it is not valid because the predicate  
**nPackets==nPacketsOld**  
is both true and false in the path

# Re-abstraction

- Use the predicate **nPackets==nPacketsOld**
- The statement **nPacketsOld = nPackets** makes the predicate true
- The statement **nPackets++** makes the predicate false if it is now true (detected by the theorem prover)

```
bool choose(pos, neg)
begin
    if (pos) then return T; elsif (neg) then return F;
    elsif (*) then return T; else return F; fi
end
```



```

void example() begin
  do {
    skip();
    A:KeAcquireSpinLock_return();
    b := T;
    skip();
    if(*) then
      skip();
      skip();
    B: KeReleaseSpinLock_return();
    skip();
    if(*) then
      skip();
      skip();
    else
      skip();
      skip();
    fi
    skip();
    skip();
    b := choose(F,b);
  } while(!b);
  skip();
  C:KeReleaseSpinLock_return();
end

```

Boolean program  
obtained by re-  
abstraction

**b:**

**nPackets==nPacketsOld**

# Model Checking Again

- In this case, there is no error path that reaches “**SLIC\_ERROR**”
- Loop invariant :  
 $(\text{state} = \text{Locked} \wedge \text{nPackets} = \text{nPacketsOld})$   
 $\vee (\text{state} = \text{Unlocked} \wedge \text{nPackets} \neq \text{nPacketsOld})$

# References

- T. Ball, and S. K. Rajamani. Automatically Validating Temporal Safety Properties of Interfaces, *SPIN*, LNCS2057, 2001.
- T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers, *EuroSys*, 2006.
- T. Ball, E. Bounimova, R. Kumar, V. Levin. SLAM2: Static Driver Verification with Under 4% False Alarms, *FMCAD*, 2010.

# SLAM

- SLAM2
- The Static Driver Verifier Research Platform
- <http://research.microsoft.com/en-us/projects/slam/>
- Related project
  - <http://mtc.epfl.ch/software-tools/blast/index-epfl.php>
    - Lazy abstraction