

# Modeling Systems

## Chapter 3

- Exercise handout today

# Modeling Systems

- 3.1 Transition Systems and Kripke Structures
- 3.2 Nondeterminism and Inputs
- 3.3 First-Order Logic and Symbolic Representations
- 3.4 Boolean Encoding
- 3.5 Modeling Digital Circuits
- 3.6 Modeling Programs
- 3.7 Fairness

# Systems and Correctness

- We consider a broad range of systems
  - Hardware (digital circuitry)
  - Software
- We want to check that the system is **correct**
  - Meets high-level requirements
  - Captured in the form of **system properties**

# Why Model?

## Specification

States what you want to prove

## System

Abstract away unnecessary details

- How does the OS scheduler work?
- How is the CPU pipeline implemented?
- What are the voltage levels in the CPU?

## But careful!

- Carelessly implemented CPUs introduce side channels
- Alpha particles may cause bits to flip
- Your formally verified system will fail when hit with a hammer
- ...

# What is a Model?

- A **model** is a description of the **behavior** of the system
- Behavior is
  - a set of observations
  - as the system evolves its state over time
- We check **algorithmically** that the model satisfies the properties
- To this end the model...
  - must have **sufficient detail** to prove the property
  - but should not be **too complex**

# Transition Systems

- A transition system is a formal model
- Formal models enable formal proof

# Kripke Structures



# Inputs

- Inputs are fully under the control of the environment
- We can use nondeterminism to model inputs

# Inputs: Light Switch Example

- Input: “button pressed” or “button released”, controlled by a hand, which is part of the environment
- Output: “light on” or “light off”



- Button is “retractive”, it bounces back
- When the light is off, pushing the button turns the light on
- When the light is on, pushing the button turns the light off

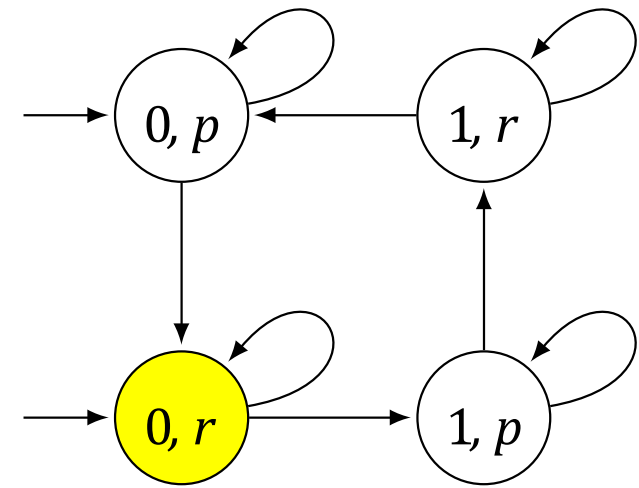
# Inputs : Light Switch Example



light switch  
"released" =  $r$



light bulb  
"off" = 0



model of the controller

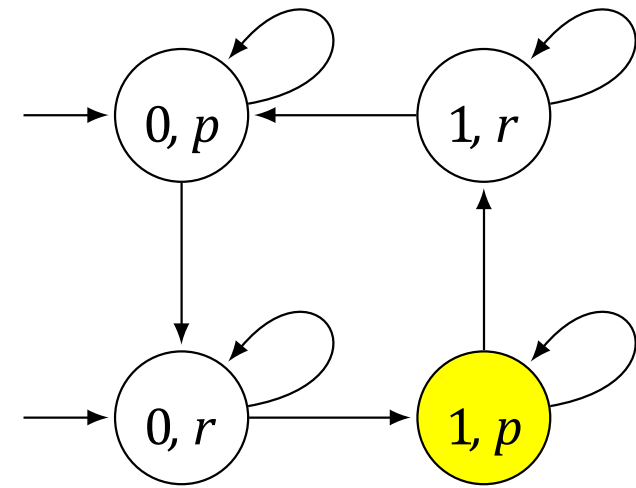
# Inputs : Light Switch Example



light switch  
"pressed" =  $p$



light bulb  
"on" = 1



model of the controller

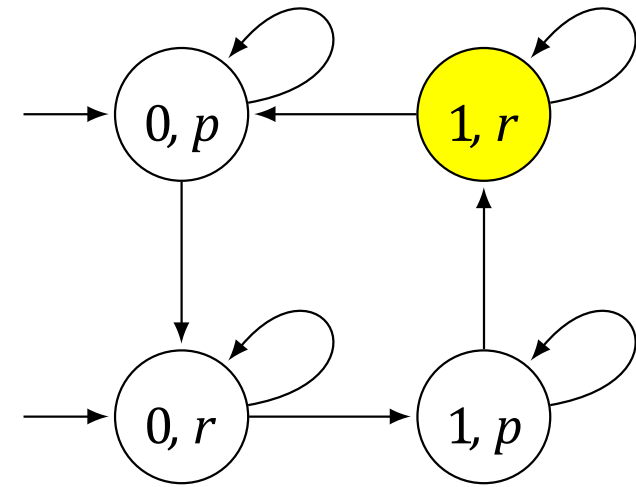
# Inputs : Light Switch Example



light switch  
"released" =  $r$



light bulb  
"on" =  $1$



model of the controller

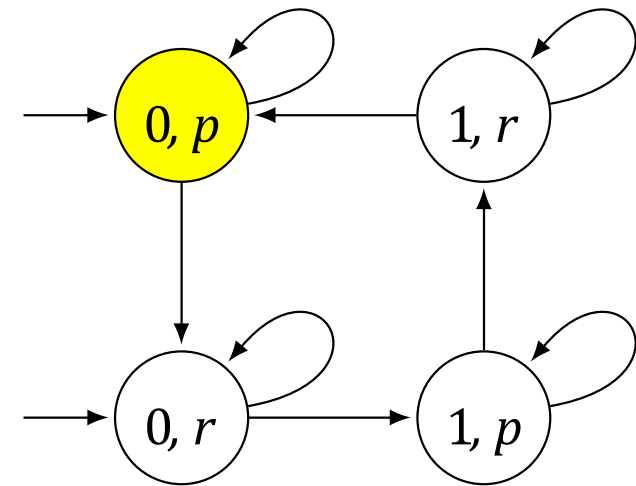
# Inputs : Light Switch Example



light switch  
"pressed" =  $p$



light bulb  
"off" = 0



model of the controller

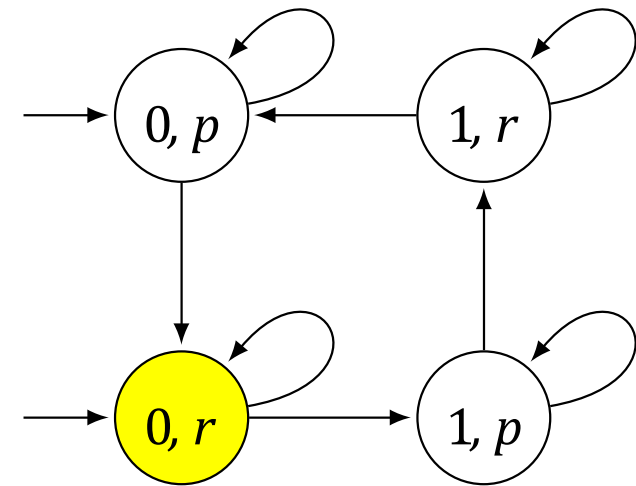
# Inputs : Light Switch Example



light switch  
"released" =  $r$



light bulb  
"off" = 0



model of the controller

# Kripke Structure $M = (S, S_0, R, AP, L)$

- $S$  – (finite) set of **states**
- $S_0 \subseteq S$  – set of **initial states**
- $R \subseteq S \times S$  – left-total **transition relation**
  - For every  $s \in S$  there exists  $s' \in S$  such that  $(s, s') \in R$
  - Left-total implies that every path is infinite
- $AP$  – finite set of **atomic propositions**
- $L : S \rightarrow 2^{AP}$  – **labeling function** that associates every state with the atomic propositions true in that state



# First-Order Logic and Symbolic Representations

# Symbolic Representation

$V = \{v_1, \dots, v_n\}$  system variables

$D_v$  domain of  $v$

$s: V \rightarrow \prod_{v \in V} D_v$  valuation, state

## Example

# Symbolic Representation

$V = \{v_1, \dots, v_n\}$  system variables

$D_v$  domain of  $v$

$s: V \rightarrow \prod_{v \in V} D_v$  valuation, state

## Example

$V = \{v_1, v_2, v_3\}, D_{v_i} = \mathbf{N}$

State space:  $\mathbf{N}^3$

examples of state:  $\{(v_1, 2), (v_2, 3), (v_3, 8)\}$  (short: (2,3,8))

# Characteristic Functions

In general, a formula is a set of states.

# Characteristic Functions

In general, a formula is a set of states.

$$v_1 = 2 \wedge v_2 = 3 \wedge v_3 = 8$$

$$(2,3,8)$$

$$v_1 = 2 \wedge v_2 = 3$$

$$\{(2,3,n_3) \mid n_3 \in \mathbf{N}\}$$

$$v_2 = 3 \wedge v_3 = v_1 + v_2$$

$$\{(n_1, 3, n_1 + 3) \mid n_1 \in \mathbf{N}\}$$

*true*

$$\mathbf{N}^3$$

# Sets and Formulas

Formula

$\mathcal{A}, \mathcal{B}$

Set

$A, B$

$A \cup B$

$A \cap B$

$S = D_{v_1} \times \dots \times D_{v_n}$

$S \setminus A$

## Example

$$v_1 = 2 \wedge v_2 = 3$$

$$v_2 = 3 \wedge v_3 = v_1 + v_2$$

$$\{(2, 3, n_3) \mid n_3 \in \mathbf{N}\}$$

$$\{(n_1, 3, n_1 + 3) \mid n_1 \in \mathbf{N}\}$$

•

# Sets and Formulas

Formula	Set
$\mathcal{A}, \mathcal{B}$	$A, B$
$\mathcal{A} \vee \mathcal{B}$	$A \cup B$
$\mathcal{A} \wedge \mathcal{B}$	$A \cap B$
$true$	$S = D_{v_1} \times \dots \times D_{v_n}$
$\neg \mathcal{A} \wedge \mathcal{B}$	$S \setminus A$

## Example

$$v_1 = 2 \wedge v_2 = 3$$

$$v_2 = 3 \wedge v_3 = v_1 + v_2$$

$$v_1 = 2 \wedge v_2 = 3 \wedge v_2 = 3 \wedge v_3 = v_1 + v_2$$

$$v_1 = 2 \wedge v_2 = 3 \vee v_2 = 3 \wedge v_3 = v_1 + v_2$$

$$\{(2, 3, n_3) \mid n_3 \in \mathbf{N}\}$$

$$\{(n_1, 3, n_1 + 3) \mid n_1 \in \mathbf{N}\}$$

$$(2, 3, 5)$$

$$\{(2, 3, n_3) \mid n_3 \in \mathbf{N}\} \cup \{(n_1, 3, n_1 + 3) \mid n_1 \in \mathbf{N}\}$$

# Transition Systems

## Example

System with variables  $x, y$  that range over  $\{0,1\}$ .

Initially,  $(x, y) = (1,1)$  and then

$x := (x + y) \text{ mod } 2.$



# Transition Systems

## Example

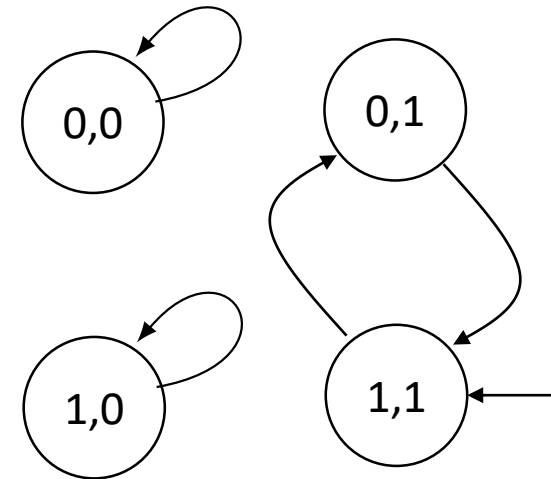
System with variables  $x, y$  that range over  $\{0,1\}$ .

Initially,  $(x, y) = (1,1)$  and then

$x := (x + y) \bmod 2$ .

Initial states:  $\mathcal{S}_0(x, y) = x = 1 \wedge y = 1$

Transitions:  $\mathcal{R}(x, y, x', y') = (x' = (x + y) \bmod 2) \wedge (y' = y)$



Kripke structure

# Modeling Digital Circuits

- Inputs are fully under the control of the environment
- We can use nondeterminism to model inputs

## Example

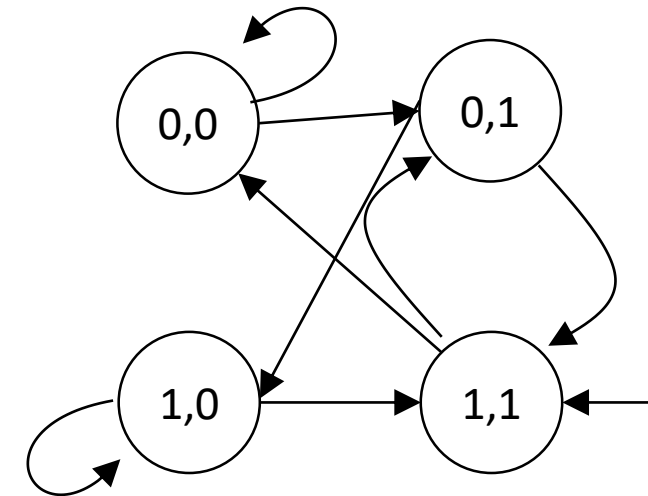
System with variables  $x, y$  that range over  $\{0,1\}$ .

Initially,  $(x, y) = (1,1)$  and then

- $x := (x + y) \bmod 2$ .
- $y$  is an input

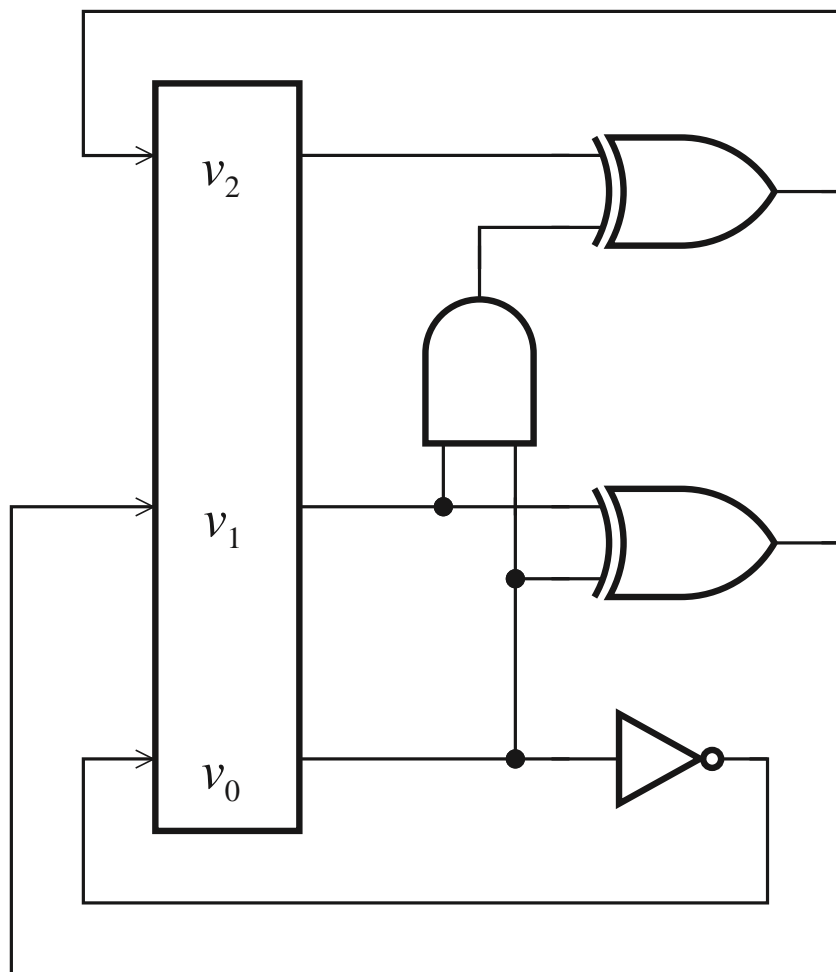
Initial states:  $\mathcal{S}_0(x, y) = x = 1 \wedge y = 1$

Transitions:  $\mathcal{R}(x, y, x', y') = (x' = (x + y) \bmod 2)$



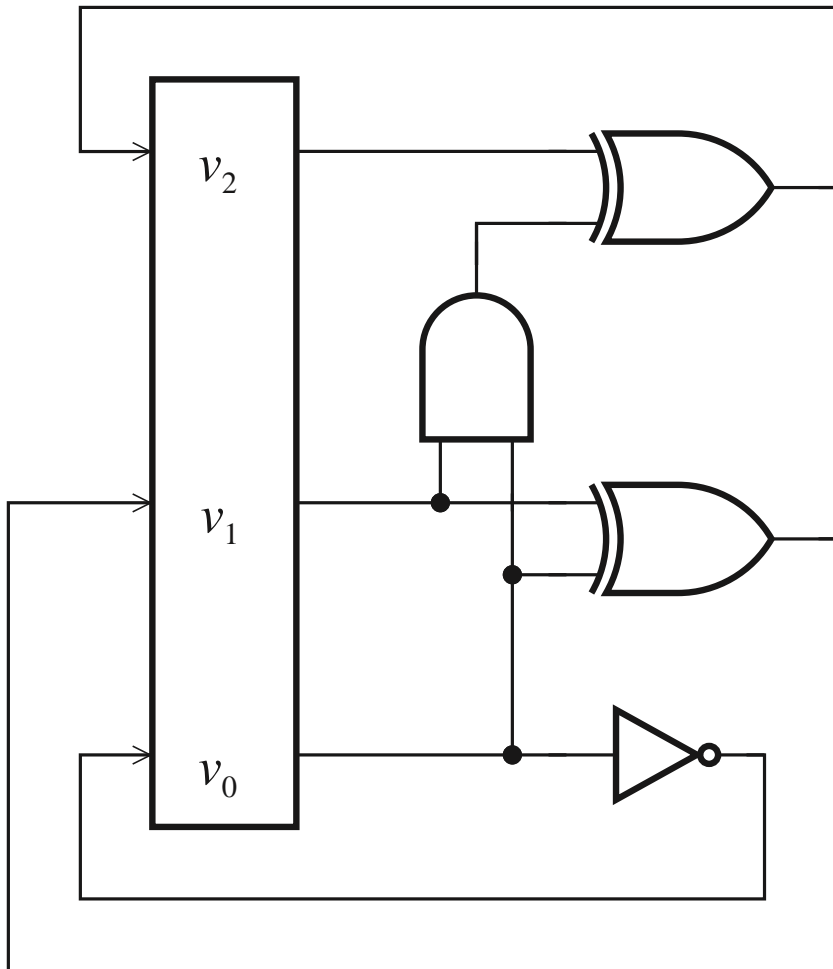
Kripke structure

## 3-bit Counter



$$\begin{aligned}\mathcal{R}_0(V, V') &= (v'_0 \leftrightarrow \neg v_0) \\ \mathcal{R}_1(V, V') &= (v'_1 \leftrightarrow v_0 \oplus v_1) \\ \mathcal{R}_2(V, V') &= (v'_2 \leftrightarrow v_2 \oplus (v_0 \wedge v_1)) \\ \mathcal{R}(V, V') &= \mathcal{R}_0 \wedge \mathcal{R}_1 \wedge \mathcal{R}_2\end{aligned}$$

# 3-bit Counter

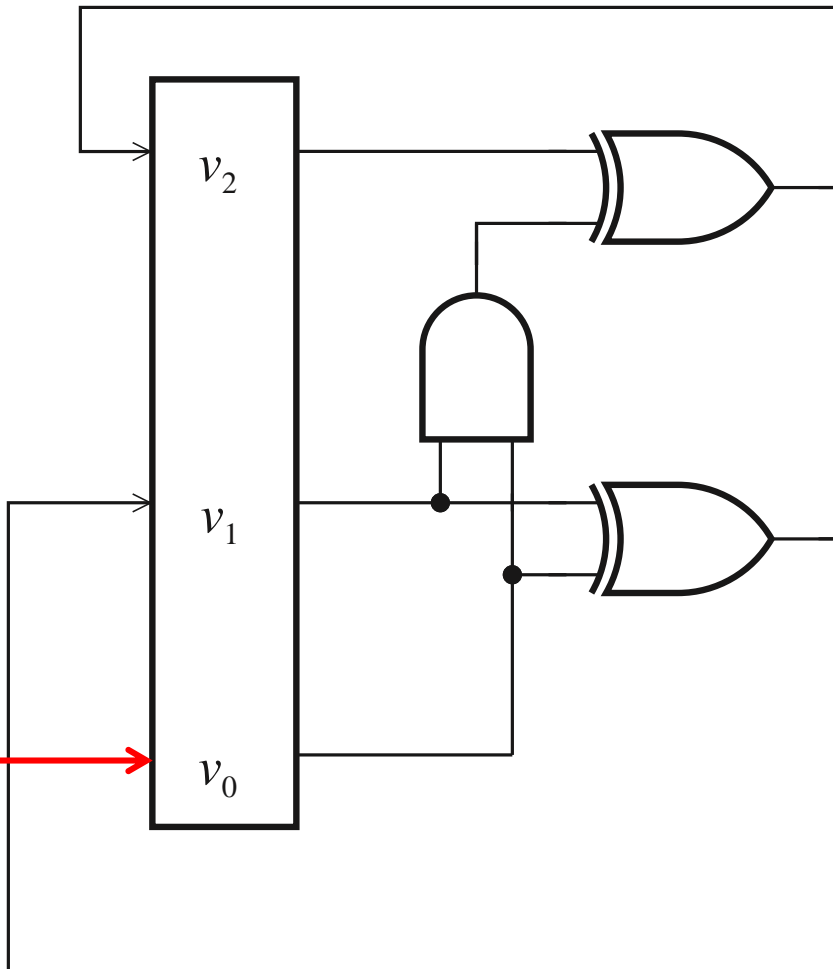


## Inputs

Inputs can be anything - model as nondeterministic

$$\mathcal{R}_0(V, V') = .$$

$$\mathcal{R}_1(V, V') = (v'_1 \leftrightarrow v_0 \oplus v_1) \quad \mathcal{R}_2(V, V') = (v'_2 \leftrightarrow v_2 \oplus (v_0 \wedge v_1))$$



# Inputs

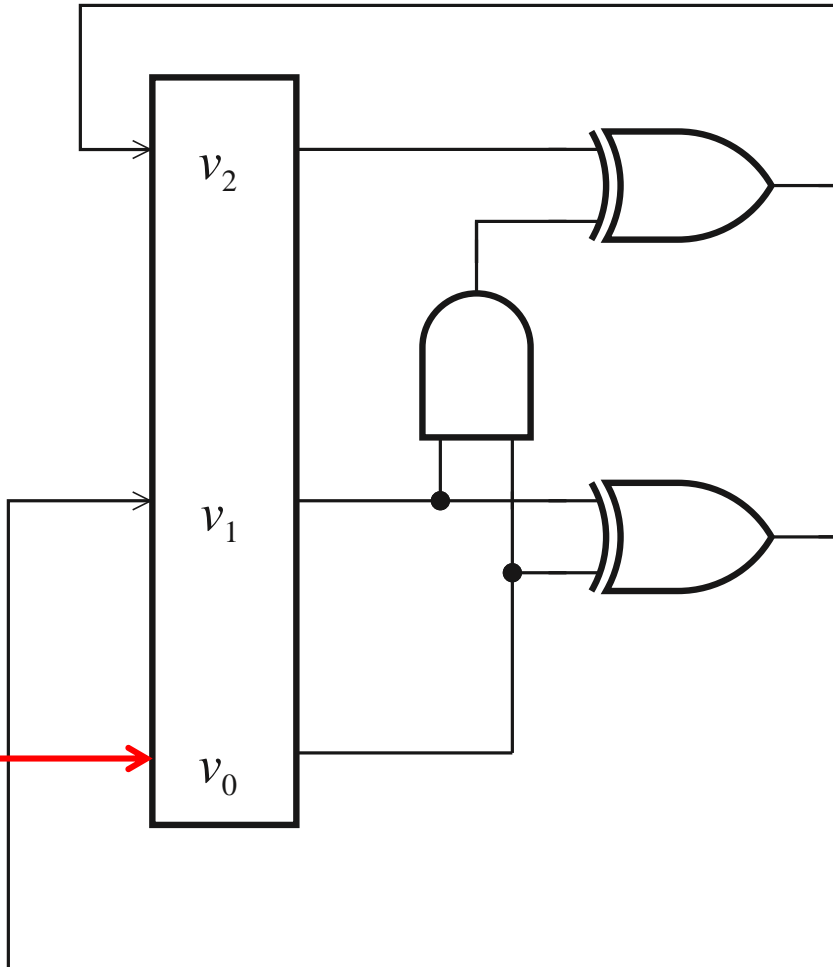
Inputs can be anything - model as nondeterministic

$$\mathcal{R}_0(V, V') = \text{true} \text{ no constraints on } v_1$$

$$\mathcal{R}_1(V, V') = (v'_1 \leftrightarrow v_0 \oplus v_1)$$

$$\mathcal{R}_2(V, V') = (v'_2 \leftrightarrow v_2 \oplus (v_0 \wedge v_1))$$

What does the Kripke structure look like?



# Symbolic Representations

**Hope:** Sets (transition relation, all reachable states) will have small formulas

## We know

- + size of transition relation  $\cong$  size of circuit, software
- To represent a subset of  $\{1, \dots, 2^k\}$  we need  $2^k$  bits in general

We will try to find algorithms that tend to produce small formulas

# Asynchronous Systems

**skipped**



# Software

# Modeling Software

## Programs

Consist of

- consecution (;)
- `if`
- `while`
- `x := e`
- `skip`
- labels `L:`

**Assume** every line has a label.

## Example

P::

```
l: cobegin P0 || P1 coend;
```

P0::

```
l0: while true do
```

```
    NC0: wait(turn = 0);
```

```
    CR0: turn := 1
```

```
end while
```

P1::

```
l1: while true do
```

```
    NC1: wait(turn = 1);
```

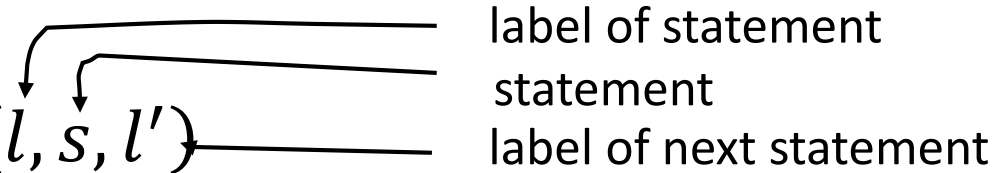
```
    CR1: turn := 0
```

```
end while
```

# Translation

Define  $same(Y) = \bigwedge_{y \in Y} y = Y'$

Define  $\mathcal{C}(l, s, l')$



label of statement  
statement  
label of next statement

$$\mathcal{C}(l, v := e, l') =$$

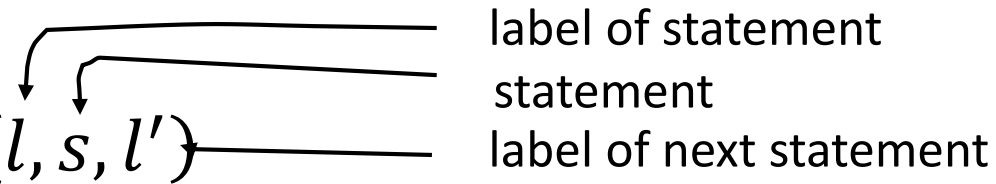
$$\mathcal{C}(l, skip, l') =$$

$$\mathcal{C}(l, (P; l' : P'), l'') =$$

# Translation

Define  $same(Y) = \bigwedge_{y \in Y} y = Y'$

Define  $\mathcal{C}(l, s, l')$



label of statement  
statement  
label of next statement

$$\mathcal{C}(l, v := e, l') = pc = l \wedge pc' = l' \wedge v' = e \wedge same(V \setminus \{v\}),$$

$$\mathcal{C}(l, skip, l') = pc = l \wedge pc' = l' \wedge same(V),$$

$$\mathcal{C}(l, (P; l': P'), l'') = \mathcal{C}(l, P, l') \vee \mathcal{C}(l', P', l''),$$

# Translation

$\mathcal{C}(l, \text{if } b \text{ then } l_1: P1 \text{ else } l_2: P2 \text{ end if}, l') =$

$\mathcal{C}(l, \text{while } b \text{ do } l_1: P1 \text{ end while}, l') =$

# Translation

$$\begin{aligned} \mathcal{C}(l, \mathbf{if\ } b \mathbf{\ then\ } l_1: P1 \mathbf{\ else\ } l_2: P2 \mathbf{\ end\ if}, l') = & \\ [pc = l \wedge b \wedge pc' = l_1 \wedge \mathit{same}(V)] \vee & \\ [pc = l \wedge \neg b \wedge pc' = l_2 \wedge \mathit{same}(V)] \vee & \\ \mathcal{C}(l_1, P1, l') \vee & \\ \mathcal{C}(l_2, P2, l') & \end{aligned}$$

$$\begin{aligned} \mathcal{C}(l, \mathbf{while\ } b \mathbf{\ do\ } l_1: P1; l_2: \mathbf{end\ while}, l') = & \\ [pc = l \wedge b \wedge pc' = l_1 \wedge \mathit{same}(V)] \vee & \\ [pc = l \wedge \neg b \wedge pc' = l' \wedge \mathit{same}(V)] \vee & \\ [pc = l_2 \wedge pc' = l \wedge \mathit{same}(V)] \vee & \\ \mathcal{C}(l_1, P1, l_2) & \end{aligned}$$

# Concurrency

**P :: cobegin**

l1: P1 l1' ||

l2: P2 l2'

**coend**

Three program counters:

1.  $pc$  for the program that invokes cobegin
2.  $pc_1$  for Thread 1
3.  $pc_2$  for Thread 2

$pc = susp$  means that the program is not running.

$$\begin{aligned} \mathcal{C}(l, \mathbf{P}, l') &= (pc = l \wedge pc' = susp \wedge pc'_1 = l_1 \wedge pc'_2 = l_2 \wedge same(V)) \vee \\ & (pc = susp \wedge pc_1 = l'_1 \wedge pc_2 = l'_2 \wedge pc' = l' \wedge pc'_1 = susp \wedge pc'_2 = susp \wedge same(V)) \vee \\ & \bigvee_{i=1}^n (C(l_i, P_i, l'_i) \wedge same(V \setminus V_i) \wedge same(PC \setminus \{pc_i\})) \end{aligned}$$

## Example

P0::

I0: **while** true **do**

    NC0: **wait**(turn = 0);

    CR0: turn := 1

**end while**

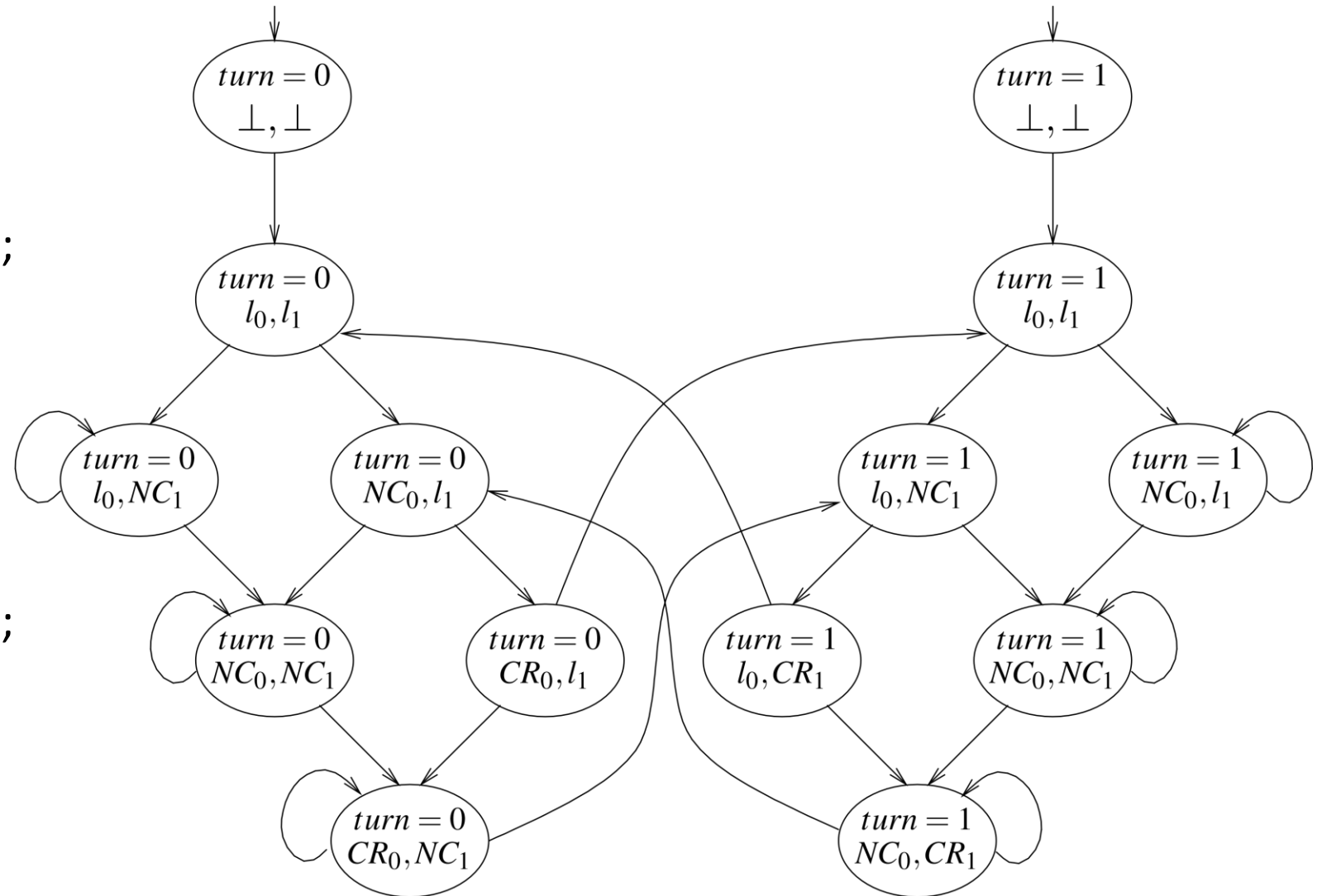
P1::

I1: **while** true **do**

    NC1: **wait**(turn = 1);

    CR1: turn := 0

**end while**





# Termination

- Programs can end
- Kripke structures are not allowed to have dead ends, reminder:
  - $R \subseteq S \times S$  – left-total **transition relation**
    - For every  $s \in S$  there exists  $s' \in S$  such that  $(s, s') \in R$
    - Left-total implies that every path is infinite
- We solve this contradiction by assuming that programs end in a self loop that does nothing

# Fairness

# Fairness

- skipped