Unit-3: Linear-time properties

B. Srivathsan

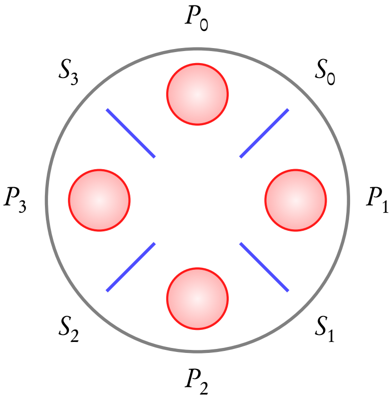
Chennai Mathematical Institute

NPTEL-course

July - November 2015

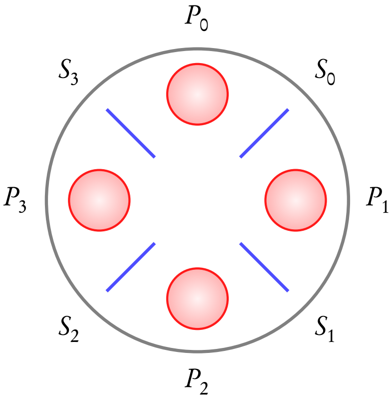
Module 1:

A problem in concurrency



$P_0 \ldots P_3$: *processes*

$S_0 \ldots S_3$: *resources*



$P_0 \ldots P_3$: *processes*

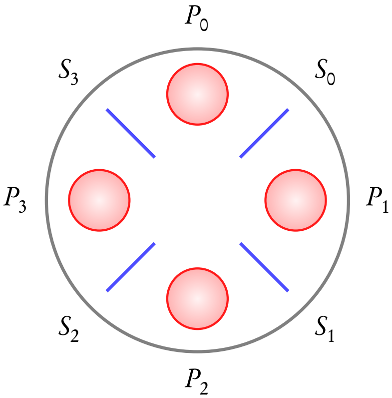
$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

only if

it has access to **resources**

$S_{(i-1)}$ and S_i



$P_0 \ldots P_3$: *processes*

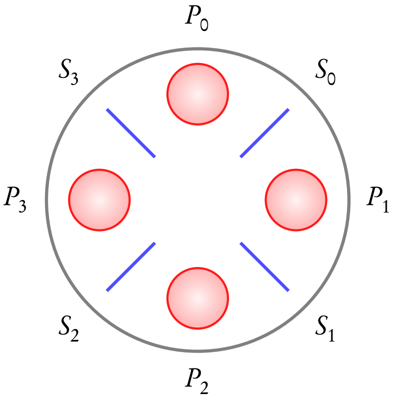
$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

only if

it has access to **resources**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$



$P_0 \ldots P_3$: *processes*

$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

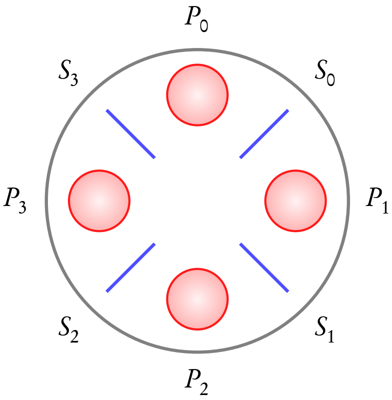
only if

it has access to **resources**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

How should the processes be **scheduled** so that **every process** can execute **infinitely often**?

Dining philosophers problem (Dijkstra)



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

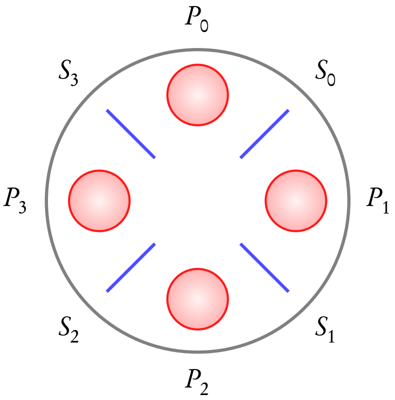
Philosopher P_i can **eat**

only if

he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

Dining philosophers problem (Dijkstra)



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

Philosopher P_i can **eat**

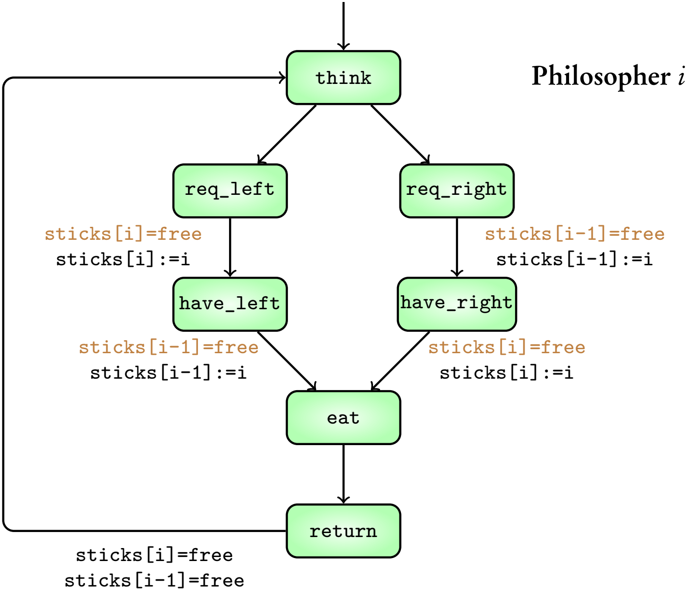
only if

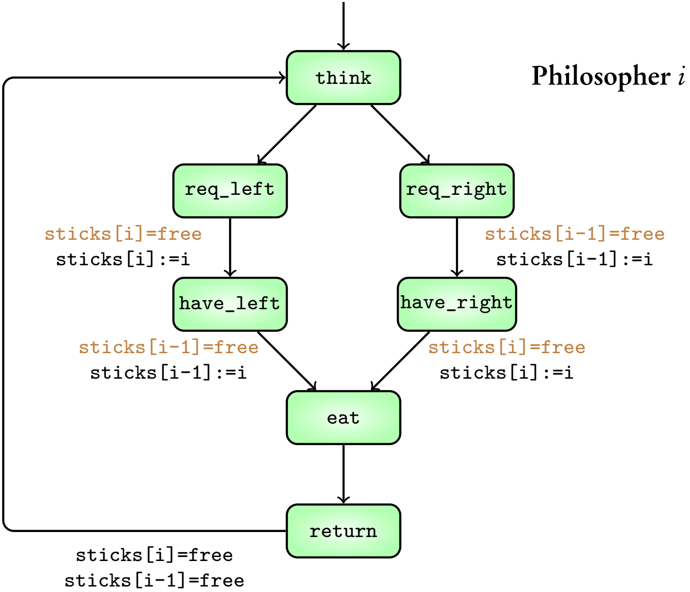
he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

What should the **protocol** be so that **every philosopher** can eat **infinitely often**?

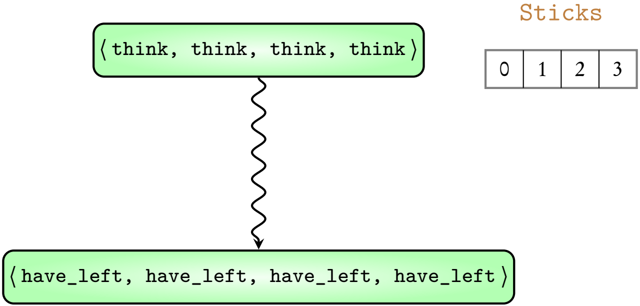
Coming next: A protocol for the dining philosophers


Philosopher i
think
req_left
req_right
sticks[i]=free
sticks[i]:=i
sticks[i-1]=free
sticks[i-1]:=i
have_left
have_right
sticks[i-1]=free
sticks[i-1]:=i
sticks[i]=free
sticks[i]:=i
eat
return
sticks[i]=free
sticks[i-1]=free


think
Philosopher i
req_left
req_right
sticks[i]=free
sticks[i]:=i
sticks[i-1]=free
sticks[i-1]:=i
have_left
have_right
sticks[i-1]=free
sticks[i-1]:=i
sticks[i]=free
sticks[i]:=i
eat
return
sticks[i]=free
sticks[i-1]=free

NuSMV demo

A deadlock



Sticks
0 1 2 3
⟨ think, think, think, think ⟩
⟨ have_left, have_left, have_left, have_left ⟩

In this unit...

What **properties** should be checked to **detect deadlocks**?

In this unit...

What **properties** should be checked to **detect deadlocks**?

- Module 2: Attach a mathematical meaning to properties

In this unit...

What **properties** should be checked to **detect deadlocks**?

- Module 2: Attach a mathematical meaning to properties
- Module 3, 4: Different examples of properties

In this unit...

What **properties** should be checked to **detect deadlocks**?

- Module 2: Attach a mathematical meaning to properties
- Module 3, 4: Different examples of properties
- Module 5: Answer to the question

Unit-3: Linear-time properties

B. Srivathsan

Chennai Mathematical Institute

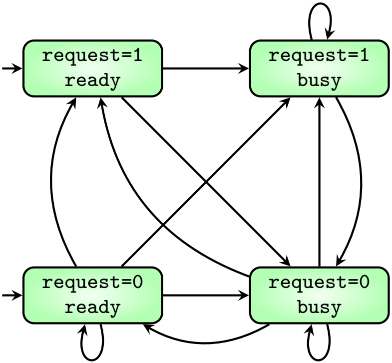
NPTEL-course

July - November 2015

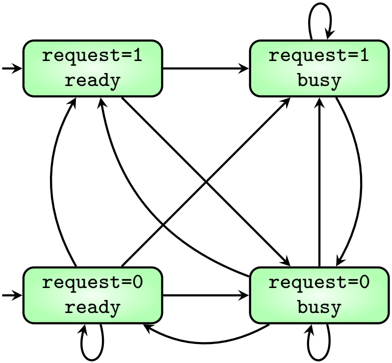
Module 2:

What is a “property”?

Goal: Attach a **mathematical meaning** to "property"

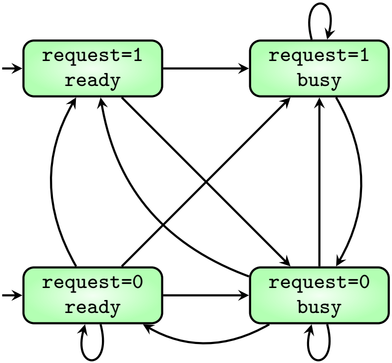



```
MODULE main
VAR
   request:  boolean;
   status:  {ready, busy}
ASSIGN
   init(status) := ready;
   next(status) := case
               request :  busy;
               TRUE : {ready,busy};
                    esac;
```




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                        request :  busy;
                        TRUE : {ready,busy};
                              esac;
```

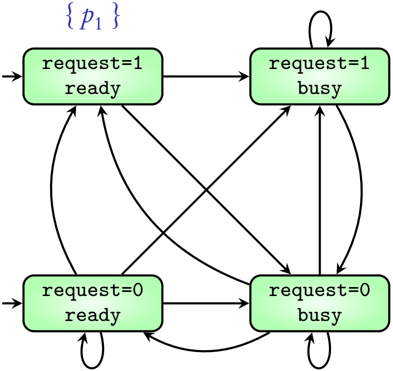
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request: boolean;
    status: {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                        request : busy;
                        TRUE : {ready,busy};
                            esac;
```

Atomic propositions

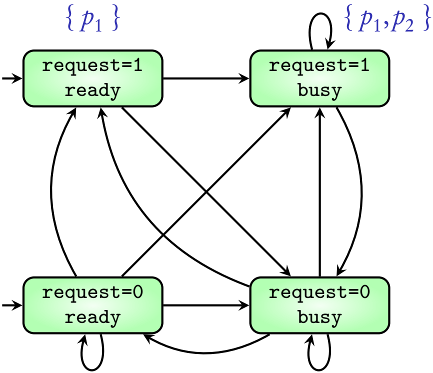
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                        request :  busy;
                        TRUE : {ready,busy};
                           esac;
```

Atomic propositions

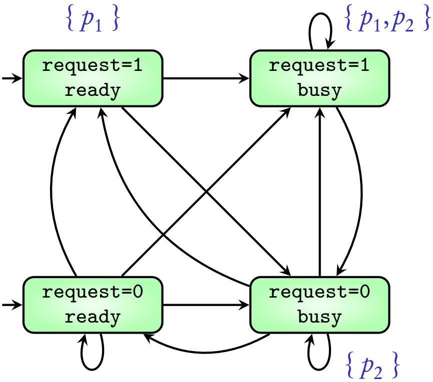
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                        esac;
```

Atomic propositions

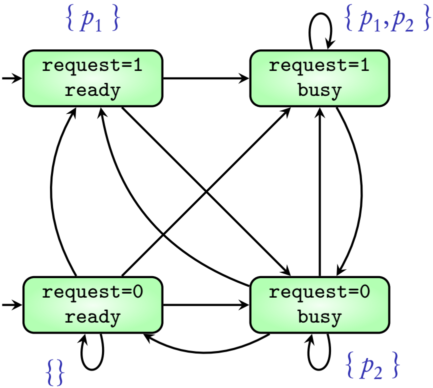
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                         esac;
```

Atomic propositions

p_1: (request=1) p_2: (status=busy)

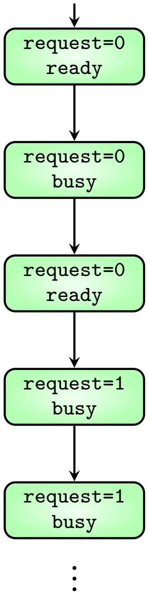



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                        request :  busy;
                        TRUE : {ready,busy};
                            esac;
```

Atomic propositions

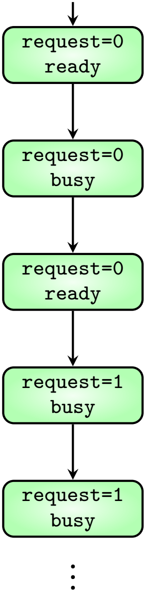
p_1: (request=1) p_2: (status=busy)

Execution



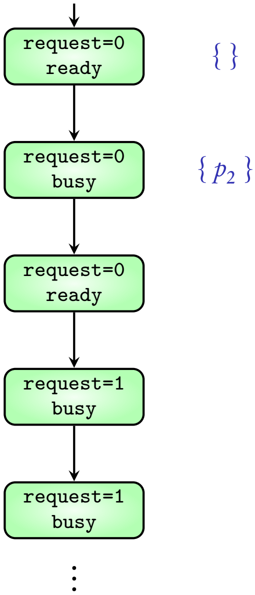
request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy

Execution


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy

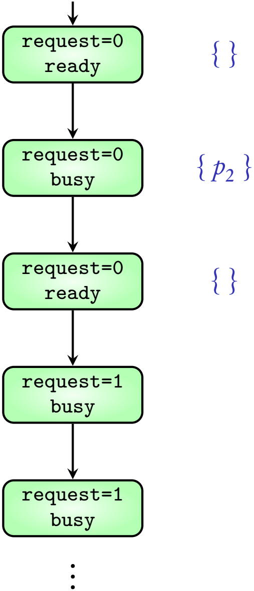
$\{\ \}$

Execution



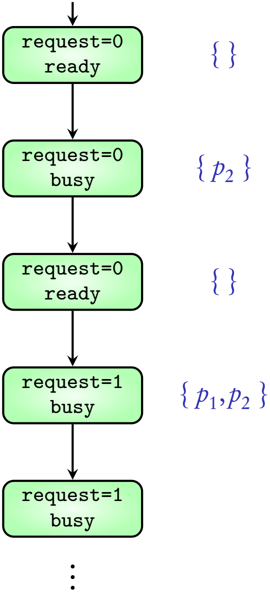
request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
request=1
busy
request=1
busy

Execution



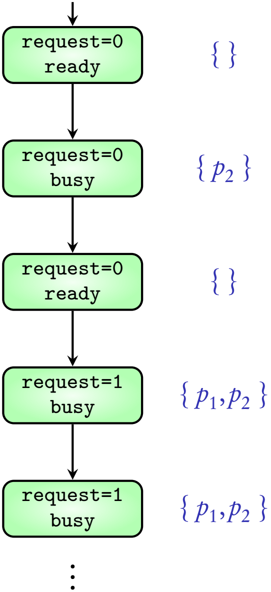
request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
request=1
busy

Execution

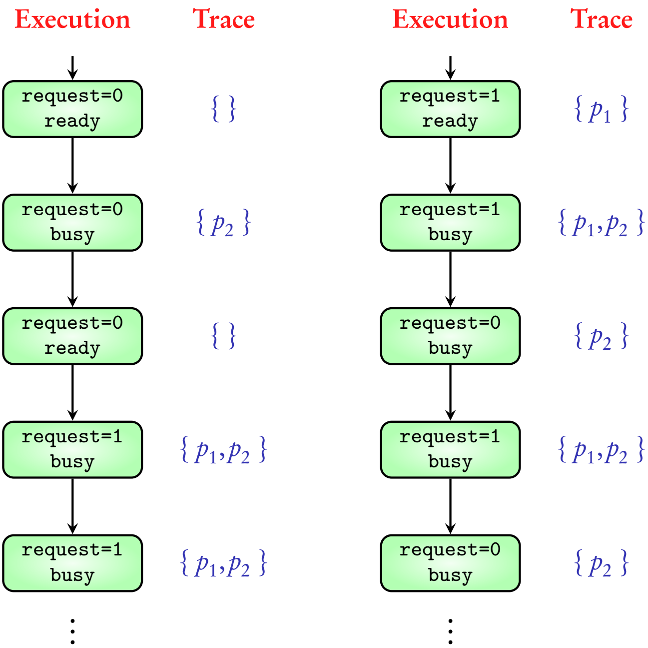


request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1, p_2 }
request=1
busy

Execution


request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1, p_2 }
request=1
busy
{ p_1, p_2 }

Execution	Trace
`request=0` `ready`	$\{ \}$
`request=0` `busy`	$\{ p_2 \}$
`request=0` `ready`	$\{ \}$
`request=1` `busy`	$\{ p_1, p_2 \}$
`request=1` `busy`	$\{ p_1, p_2 \}$
$\vdots$	


Execution
Trace
request=0 ready
{ }
request=0 busy
{ p_2 }
request=0 ready
{ }
request=1 busy
{ p_1,p_2 }
request=1 busy
{ p_1,p_2 }
Execution
Trace
request=1 ready
{ p_1 }
request=1 busy
{ p_1,p_2 }
request=0 busy
{ p_2 }
request=1 busy
{ p_1,p_2 }
request=0 busy
{ p_2 }

$$\mathbf{AP} = \{ p_1, p_2, \ldots, p_k \}$$

$$
\begin{aligned}
\mathbf{AP} = &\ \{\ p_1, p_2,\ \cdots, p_k\ \} \\
\mathit{PowerSet}(\mathbf{AP}) = &\ \{\ \{\ \}, \{p_1\},\ \ldots\ , \{p_k\}, \\
&\quad \{\ p_1, p_2\ \}, \{\ p_1, p_3\ \},\ \ldots\ , \{\ p_{k-1}, p_k\ \}, \\
&\qquad \cdots \\
&\quad \{\ p_1, p_2,\ \ldots, p_k\ \}\ \}
\end{aligned}
$$

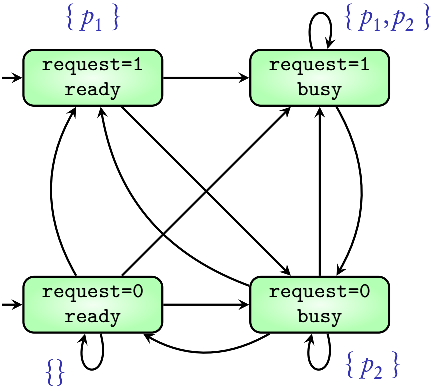
$$\begin{aligned}\mathbf{AP} &= \{\ p_1, p_2,\ \dots\ , p_k\ \} \\ \mathit{PowerSet}(\mathbf{AP}) &= \{\ \{\ \},\ \{p_1\},\ \dots\ ,\ \{p_k\}, \\ &\qquad \{\ p_1, p_2\ \},\ \{\ p_1, p_3\ \},\ \dots\ ,\ \{\ p_{k-1}, p_k\ \}, \\ &\qquad\qquad \dots \\ &\qquad \{\ p_1, p_2,\ \dots,\ p_k\ \}\ \}\end{aligned}$$

Trace(Execution) is an **infinite word** over *PowerSet*(**AP**)

$$\begin{aligned}\mathbf{AP} &= \{\ p_1, p_2, \ \dots\ , p_k\ \} \\ \mathit{PowerSet}(\mathbf{AP}) &= \{\ \{\ \},\ \{p_1\},\ \dots\ ,\ \{p_k\}, \\ &\qquad \{\ p_1, p_2\ \},\ \{\ p_1, p_3\ \},\ \dots\ ,\ \{\ p_{k-1}, p_k\ \}, \\ &\qquad \dots \\ &\qquad \{\ p_1, p_2,\ \dots,\ p_k\ \}\ \}\end{aligned}$$

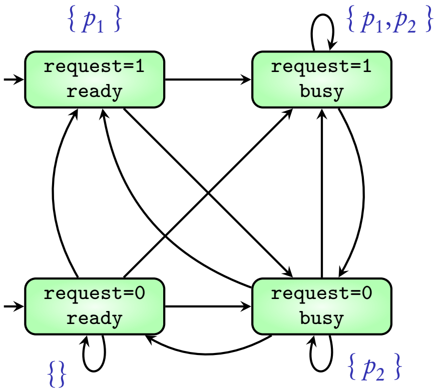
Trace(Execution) is an **infinite word** over *PowerSet*(**AP**)

Traces(TS) is the $\{$ Trace(σ) $\mid$ σ is an execution of the TS $\}$



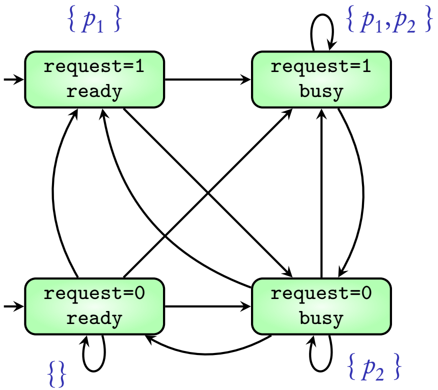
$\{ p_1 \}$
$\{ p_1, p_2 \}$
request=1
ready
request=1
busy
request=0
ready
request=0
busy
$\{\}$
$\{ p_2 \}$

Traces:



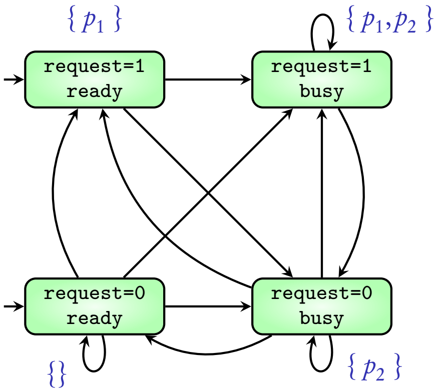
{ p_1 }
{ p_1,p_2 }
request=1
ready
request=1
busy
request=0
ready
request=0
busy
{}
{ p_2 }

Traces: { } { } { } { } { } { } { } { } …



Traces: $\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\ldots$

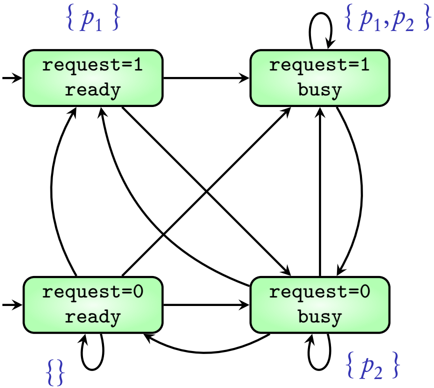
$\{\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\ldots$



Traces: $\{\}\,\{\}\,\{\}\,\{\}\,\{\}\,\{\}\,\{\}\,\{\}\dots$

$\{\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\dots$

$\{p_1\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\dots$

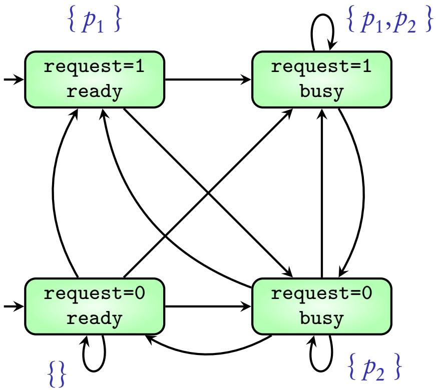


Traces: $\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\dots$

$\{\,\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\dots$

$\{p_1\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\dots$

$\{\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\dots$



Traces: $\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\,\{\,\}\ldots$

$\{\,\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\,\{p_2\}\ldots$

$\{p_1\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\,\{p_2\}\,\{p_1,p_2\}\ldots$

$\{\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\,\{p_1,p_2\}\ldots$

$\vdots$

Traces of a TS describe its **behaviour** with respect to the atomic propositions

Behaviour of TS

Atomic propositions

Set of its **traces**

Coming next: What is a **property**?

AP-$\textsc{inf}$ = set of **infinite words** over *PowerSet*(**AP**)

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{ A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \textbf{ each } A_i \text{ contains } p_1 \}$$

$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \ldots$

$\{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \ldots$

$\vdots$

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{\, A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \textbf{each } A_i \text{ contains } p_1 \,\}$$

$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\ldots$

$\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\ldots$

$\vdots$

Property 2: p_1 is true at least once and p_2 is always true

AP-INF $=$ set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{\, A_0A_1A_2 \cdots \in \text{AP-INF} \mid \textbf{ each } A_i \text{ contains } p_1 \,\}$$

$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\ldots$

$\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\ldots$

$\vdots$

Property 2: p_1 is true at least once and p_2 is always true

$$\{\, A_0A_1A_2 \cdots \in \text{AP-INF} \mid \textbf{ exists } A_i \text{ containing } p_1 \text{ and } \textbf{every } A_j \text{ contains } p_2 \,\}$$

$\{p_2\}\{p_1,p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_1,p_2\}\{p_2\}\ldots$

$\{p_1,p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\ldots$

$\vdots$

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

A property over AP is a **subset** of AP-INF

Behaviour of TS

Atomic propositions

Set of its **traces**

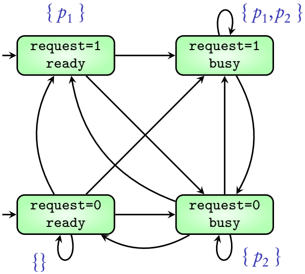
Property over AP

Subset of AP-INF

When does a transition system **satisfy** a property?

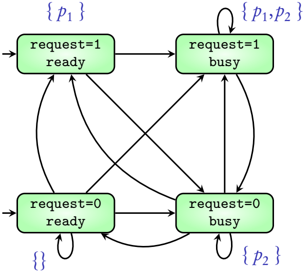
$$\mathrm{AP} = \{\ p_1,\ p_2\ \}$$

Transition System



$$\mathrm{AP} = \{\ p_1,\ p_2\ \ \}$$

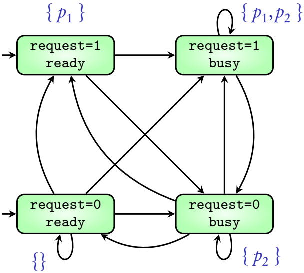
Transition System



Property

$$\mathrm{AP} = \{\ p_1,\ p_2\ \}$$

Transition System

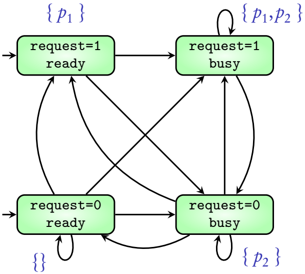


Property

$$\mathbf{G}\ p_1$$

$$\mathrm{AP} = \{\ p_1,\ p_2\ \}$$

Transition System



Property

$$\mathbf{G}\ p_1$$

Transition system TS satisfies property P if

$$\boxed{\textbf{Traces}(TS) \subseteq P}$$

A property over AP is a subset of AP-INF

A property over AP is a subset of AP-INF

$\longrightarrow$ hence also called **Linear-time property**

Behaviour of TS

Atomic propositions

Set of its **traces**

Property over AP

Subset of AP-INF

When does system satisfy property?

Unit-3: Linear-time properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

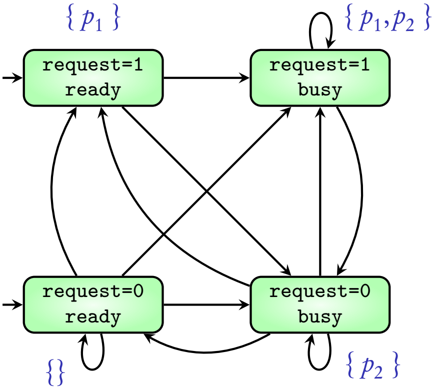
July - November 2015

Module 3:

Invariants

Atomic propositions AP $= \{ p_1, p_2 \}$

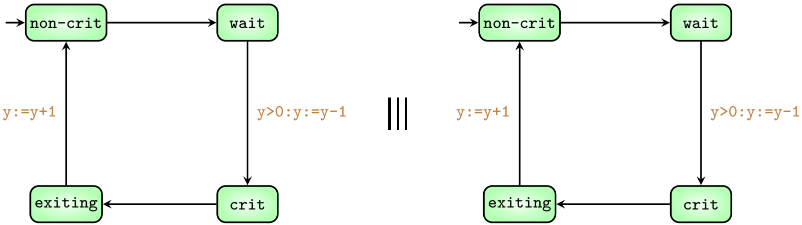
p_1: `request=1` p_2: `status=busy`



Atomic propositions $\mathrm{AP} = \{ p_1, p_2, p_3, p_4 \}$

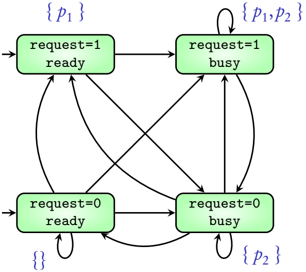
p_1: `pr1.location=crit` p_2: `pr1.location=wait`

p_3: `pr2.location=crit` p_4: `pr2.location=wait`



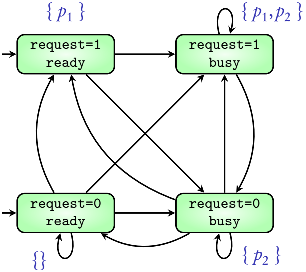
Atomic propositions $\mathrm{AP} = \{\, p_1, p_2 \,\}$

p_1: `request=1` p_2: `status=busy`



Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i contains $p_1 \}$

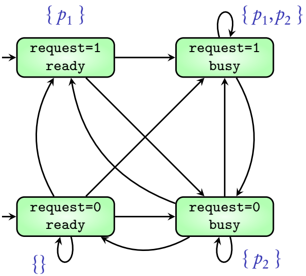
$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \ldots$

$\{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \ldots$

$\vdots$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF $=$ set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

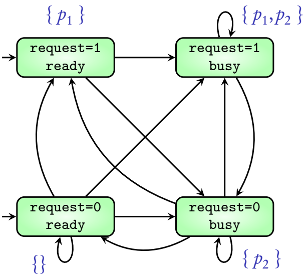
$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i contains $p_1 \}$

$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \dots$
$\{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \dots$
$\vdots$

Property 1 is written as $\mathbf{G}\, p_1$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i contains p_1 $\}$

$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \ldots$

$\{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_1 \} \{ p_1, p_2 \} \ldots$

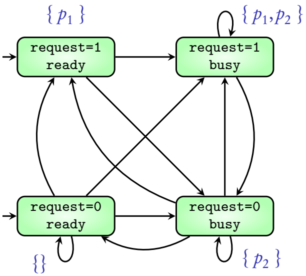
$\vdots$

Property 1 is written as **G** p_1

Above TS **does not satisfy** G p_1

Atomic propositions AP $= \{\, p_1, p_2 \,\}$

p_1: `request=1` p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

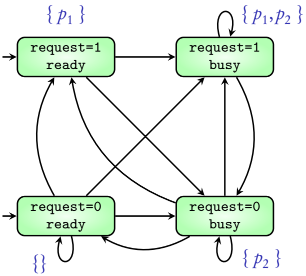
Property 2: $p_1 \wedge \neg p_2$ is always true

$\{\, A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid$ **each** A_i satisfies $p_1 \wedge \neg p_2 \,\}$

$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\ldots$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` $\quad$ p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 2: $p_1 \wedge \neg p_2$ is always true

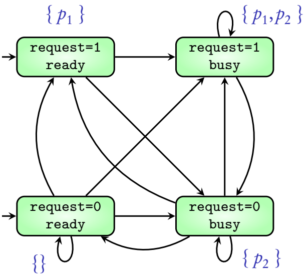
$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i satisfies $p_1 \wedge \neg p_2 \}$

$\{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \{ p_1 \} \dots$

Property 2 is written as $\mathbf{G}\ p_1 \wedge \neg p_2$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF $=$ set of **infinite words** over *PowerSet*(**AP**)

Property 2: $p_1 \wedge \neg p_2$ is always true

$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i satisfies $p_1 \wedge \neg p_2 \}$

$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\ldots$

Property 2 is written as $\mathbf{G}\ p_1 \wedge \neg p_2$

Above TS **does not satisfy** G $p_1 \wedge \neg p_2$

Invariants

AP-INF $=$ set of **infinite words** over *PowerSet*(**AP**)

Property: ϕ is **always true**
(where ϕ is a boolean expression over AP)

$\{\ A_0 A_1 A_2 \cdots \in$ AP-INF $\mid$ **each** A_i satisfies $\phi\ \}$

Invariants

AP-INF $=$ set of **infinite words** over *PowerSet*(**AP**)

Property: ϕ is **always true**
(where ϕ is a boolean expression over AP)

$\{ A_0 A_1 A_2 \cdots \in$ AP-INF $|$ **each** A_i satisfies $\phi \}$

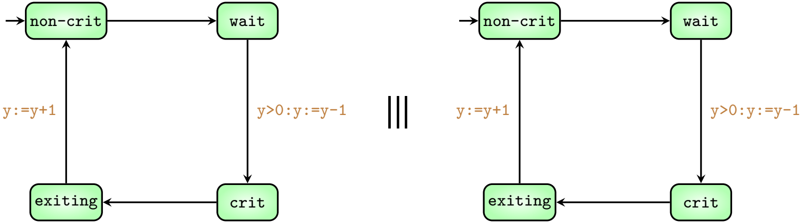
A property of the above form is called **invariant** property

It is written as G ϕ

Atomic propositions $\mathrm{AP} = \{ p_1, p_2, p_3, p_4 \}$

p_1: `pr1.location=crit` p_2: `pr1.location=wait`

p_3: `pr2.location=crit` p_4: `pr2.location=wait`



Above TS satisfies invariant property $\mathbf{G}\ \neg (p_1 \wedge p_3)$

Algorithm

Input: A TS and property $\mathsf{G}\,\phi$

Output: Does TS satisfy invariant $\mathsf{G}\,\phi$?

Algorithm

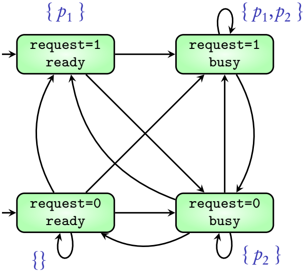
Input: A TS and property $\mathbf{G}\ \phi$

Output: Does TS satisfy invariant $\mathbf{G}\ \phi$?

A TS satisfies an invariant ϕ

if and only if

every **reachable state** of the TS satisfies ϕ



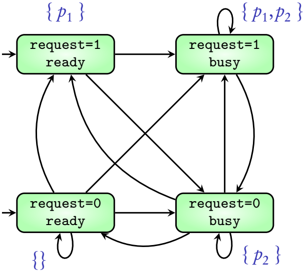
Property to check: $\mathrm{G}\ p_1$

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \land (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

1

R U b

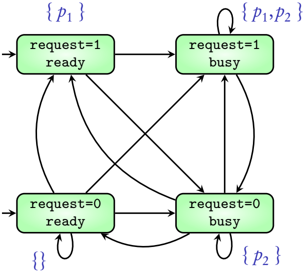


Property to check: $\mathbf{G}\, p_1$

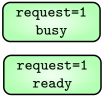
set R, **stack** U, **bool** b
for all initial states s
    if $s \notin R$ **then**
        visit(s)
    endif
return b

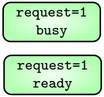
procedure visit (states s)
    $push(s,U)$; $R := R \cup \{ s \}$
    while (U is not empty)
        $s' := top(U)$
        if $Post(s') \subseteq R$ **then**
            $pop(U)$
            $b := b \wedge (s' \models \phi)$
        else
            let $s'' \in Post(s') \setminus R$
            $push(s'', U)$
            $R := R \cup \{s''\}$
        endif
    endwhile

request=1 ready	request=1 ready	1
R	U	b



Property to check: $\mathbf{G}\, p_1$



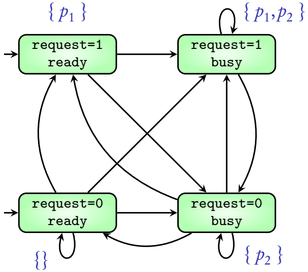


1

R $\qquad$ U $\qquad$ b

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

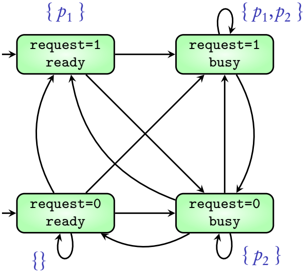


Property to check: $\mathbf{G}\ p_1$

R	U	b
request=0 busy	request=0 busy	
request=1 busy	request=1 busy	
request=1 ready	request=1 ready	1

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

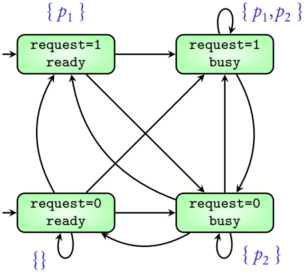


Property to check: $\mathbf{G}\, p_1$

R	U	b
request=0 busy	request=0 busy	
request=0 busy	request=0 busy	
request=1 busy	request=1 busy	
request=1 ready	request=1 ready	1

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

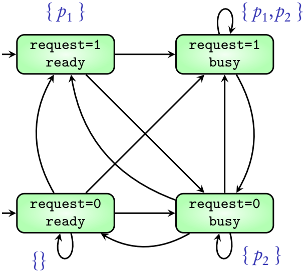


Property to check: $\mathrm{G}\, p_1$

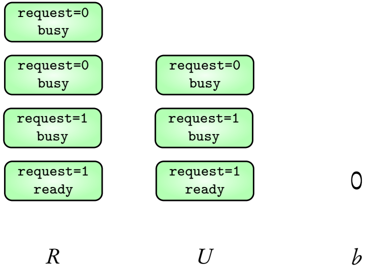
R	U	b
request=0 busy	request=0 busy	
request=0 busy	request=0 busy	
request=1 busy	request=1 busy	
request=1 ready	request=1 ready	0

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

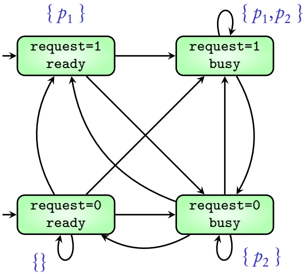


Property to check: $\mathbf{G}\, p_1$

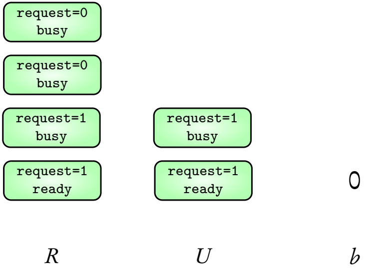


set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

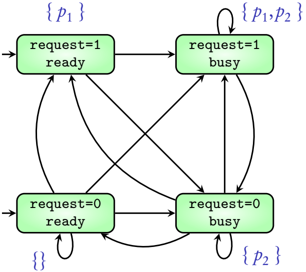


Property to check: $\mathbf{G}\, p_1$

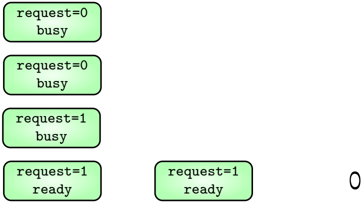


set R, **stack** U, **bool** b
for all initial states s
    if $s \notin R$ **then**
        visit(s)
    endif
return b

procedure visit (states s)
    $push(s, U)$; $\quad R := R \cup \{ s \}$
    while (U is not empty)
        $s' := top(U)$
        if $Post(s') \subseteq R$ **then**
            $pop(U)$
            $b := b \wedge (s' \models \phi)$
        else
            let $s'' \in Post(s') \setminus R$
            $push(s'', U)$
            $R := R \cup \{s''\}$
        endif
    endwhile



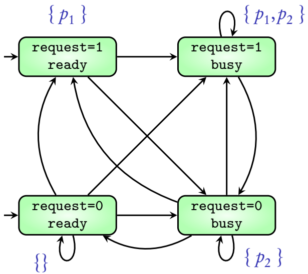
Property to check: $\mathbf{G}\, p_1$



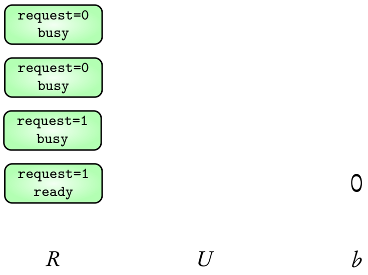
R U b

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile

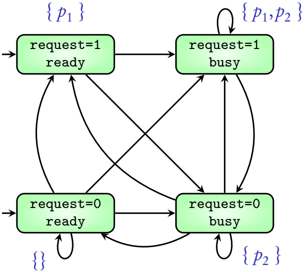


Property to check: $\mathrm{G}\ p_1$



set R, **stack** U, **bool** b
for all initial states s
    if $s \notin R$ **then**
        visit(s)
    endif
return b

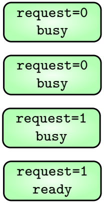
procedure visit (states s)
    $push(s, U)$; $\quad R := R \cup \{ s \}$
    while (U is not empty)
        $s' := top(U)$
        if $Post(s') \subseteq R$ **then**
            $pop(U)$
            $b := b \wedge (s' \models \phi)$
        else
            let $s'' \in Post(s') \setminus R$
            $push(s'', U)$
            $R := R \cup \{s''\}$
        endif
    endwhile



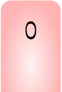
Property to check: $\mathrm{G}\, p_1$

set R, **stack** U, **bool** b
for all initial states s
  if $s \notin R$ **then**
    visit(s)
  endif
return b

procedure visit (states s)
  $push(s, U)$; $R := R \cup \{ s \}$
  while (U is not empty)
    $s' := top(U)$
    if $Post(s') \subseteq R$ **then**
      $pop(U)$
      $b := b \wedge (s' \models \phi)$
    else
      let $s'' \in Post(s') \setminus R$
      $push(s'', U)$
      $R := R \cup \{s''\}$
    endif
  endwhile



Property **not satisfied**



R U b

Invariants

G ϕ

Algorithm to check invariants

Unit-3: Linear-time properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

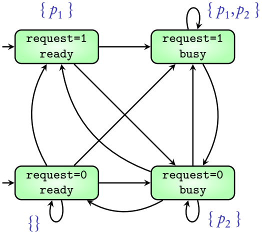
July - November 2015

Module 4:

Safety properties

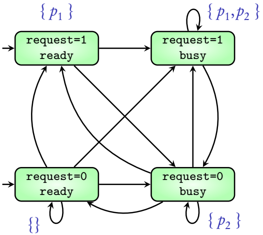
Atomic propositions AP = $\{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property: Always: if p_1 is true, then in the **next step** p_2 is true

$$\{ A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \text{ if } A_i \text{ contains } p_1, \text{ then } A_{i+1} \text{ contains } p_2 \}$$

$\{ p_1 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \dots$

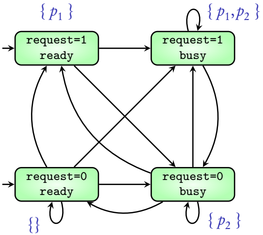
$\{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \dots$

$\{ \} \{ \} \{ \} \{ \} \{ \} \{ \} \dots$

$\vdots$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` $\quad$ p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property: Always: if p_1 is true, then in the **next step** p_2 is true

$\{ A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \text{ if } A_i \text{ contains } p_1, \text{ then } A_{i+1} \text{ contains } p_2 \}$

$\{ p_1 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \ldots$

$\{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \ldots$

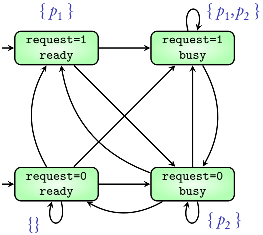
$\{ \} \{ \} \{ \} \{ \} \{ \} \{ \} \ldots$

$\vdots$

Property is written as $\mathbf{G} \ (p_1 \longrightarrow X p_2 \ )$

Atomic propositions AP $= \{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property: Always: if p_1 is true, then in the **next step** p_2 is true

$$\{ A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \text{ if } A_i \text{ contains } p_1, \text{ then } A_{i+1} \text{ contains } p_2 \}$$

$\{ p_1 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \{ p_2 \} \{ p_1 \} \{ p_1, p_2 \} \ldots$

$\{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \{ p_2 \} \ldots$

$\{ \} \{ \} \{ \} \{ \} \{ \} \{ \} \ldots$

$\vdots$

Property is written as $\mathbf{G}\ (p_1 \longrightarrow X p_2)$

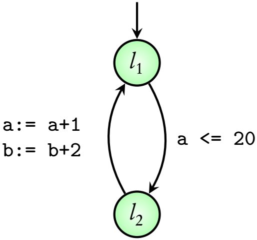
Above TS **satisfies** this property

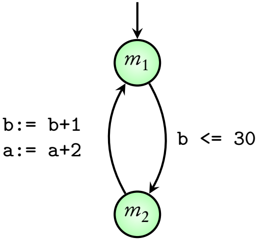
$\mathbf{X}$ operator

- $G\,(p_1 \longrightarrow XXp_2)$:
  - Always: if p_1 is true then in the next to next step p_2 is true
- $F\,(p_1 \wedge X\neg p_1)$:
  - Somewhere: p_1 is true and in the next step it becomes false
- $G\,(\,Xp_2 \longrightarrow p_1)$:
  - Always: if p_2 is true then in the previous step p_1 is true

```
while a <= 20
      a := a+1
      b := b+2
```

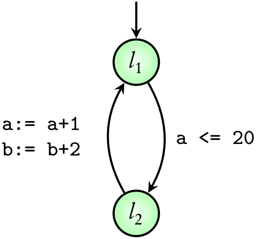
```
while b <= 30
      b:=b+1
      a:=a+2
```

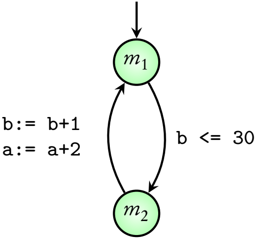




```
while a <= 20
      a := a+1
      b := b+2
```

```
while b <= 30
      b:=b+1
      a:=a+2
```

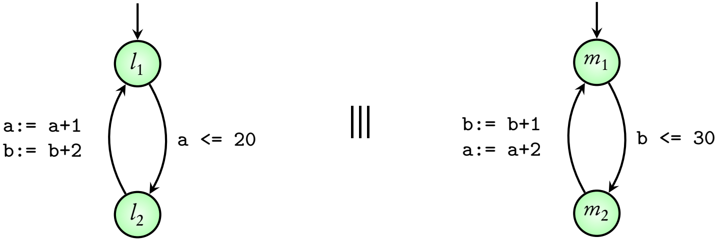




Check: Whenever $a \geq 10$, in the next to next step $b \geq 12$

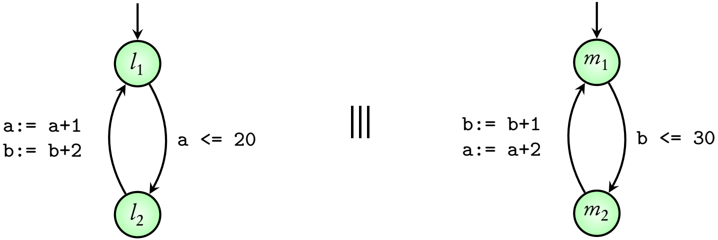
Atomic propositions AP $= \{ p_1, p_2 \}$

p_1 : `a >= 10` p_2 : `b >= 12`



Atomic propositions AP $= \{ p_1, p_2 \}$

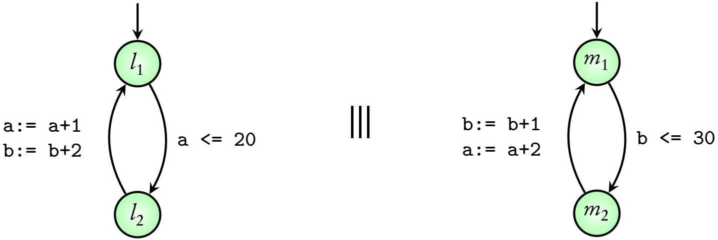
p_1 : `a >= 10` p_2 : `b >= 12`



Check: G $(p_1 \longrightarrow XXp_2)$

Atomic propositions $\mathrm{AP} = \{ p_1, p_2 \}$

p_1 : `a >= 10` p_2 : `b >= 12`



Check: $\mathrm{G}\,(p_1 \longrightarrow XXp_2\ )$

NuSMV demo

Coming next: idea of safety properties

Property 1: if p_1 is true, then p_2 should be true in the next step

$\{p_1\}\{\neg p_2\}$ "something bad"

Property 1: if p_1 is true, then p_2 should be true in the next step

$$\{ p_1 \}\{\neg p_2\}$$

"something bad"

Property 2: if p_1 is true, then p_2 should be true in the next to next step

$$\{ p_1 \}\{\dots\}\{\neg p_2\}$$

"something bad"

Property 1: if p_1 is true, then p_2 should be true in the next step

$\rightsquigarrow \{p_1\}\{\neg p_2\}$ "something bad"

Property contains all words where **something bad** is absent

Property 2: if p_1 is true, then p_2 should be true in the next to next step

$\rightsquigarrow \{p_1\}\{\dots\}\{\neg p_2\}$ "something bad"

Safety properties

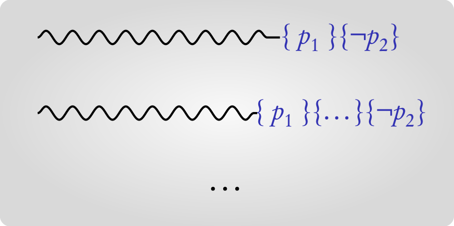
AP-INF = set of **infinite words** over *PowerSet*(**AP**)

P: a property over AP

Safety properties

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

P: a property over AP



Bad-Prefixes

P is a safety property if there **exists** a set Bad-Prefixes such that

Safety properties

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

P: a property over AP

$\{p_1\}\{\neg p_2\}$

$\{p_1\}\{\ldots\}\{\neg p_2\}$

$\ldots$

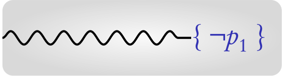
Bad-Prefixes

P is a safety property if there **exists** a set Bad-Prefixes such that

P is the set of **all words** that **do not start** with a Bad-Prefix

Invariants are **special cases** of safety properties

Property: Always p_1 is true



“Bad-Prefixes”

Safety properties

Avoiding bad prefixes

X operator

Unit-3: Linear-time properties

B. Srivathsan

Chennai Mathematical Institute

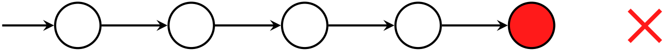
NPTEL-course

July - November 2015

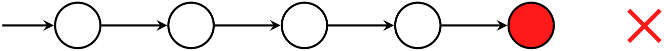
Module 5:

Liveness properties

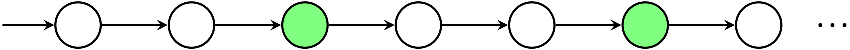
Safety: Something bad **never happens**



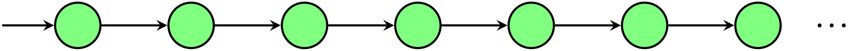
Safety: Something bad **never happens**



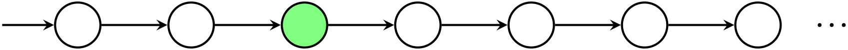
Liveness: Something **good** happens **infinitely often**



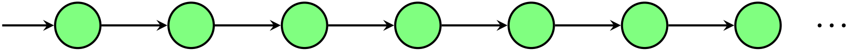
G p: Always p



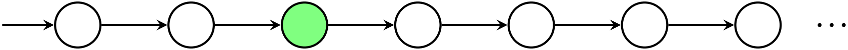
F p: Sometime p



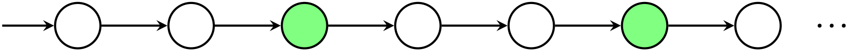
$G\ p$: Always p



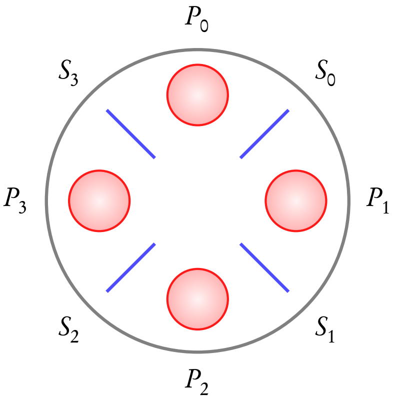
$F\ p$: Sometime p



$G\ F\ p$: Infinitely often p



Recall...



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

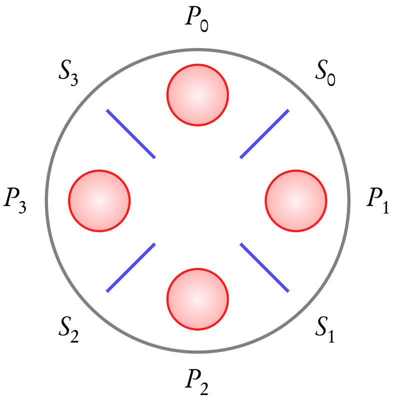
Philosopher P_i can **eat**

only if

he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

Recall...



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

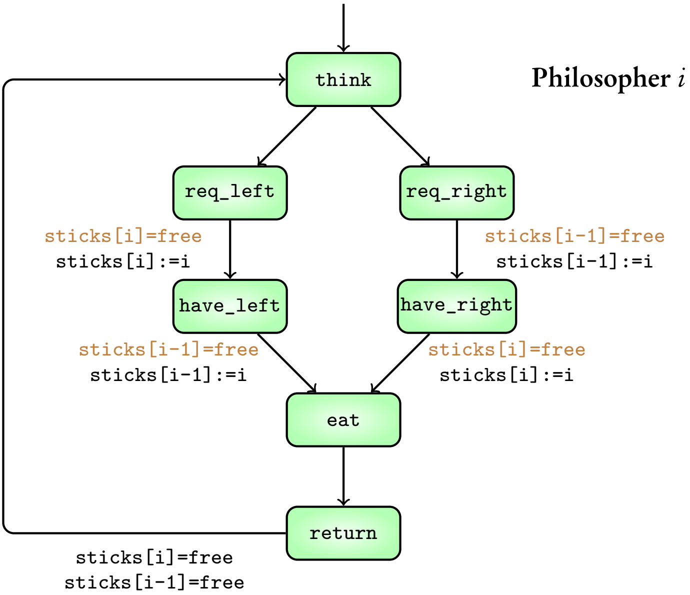
Philosopher P_i can **eat**

only if

he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

What should the **protocol** be so that **every philosopher** can eat **infinitely often**?



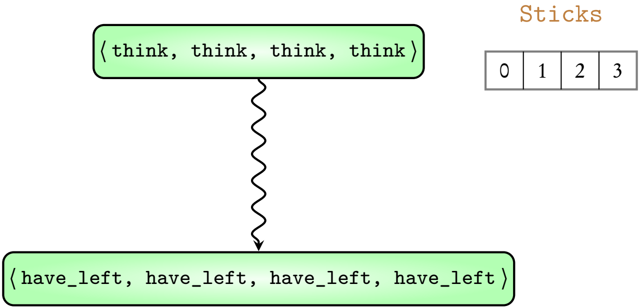
NuSMV code for the protocol

Sticks

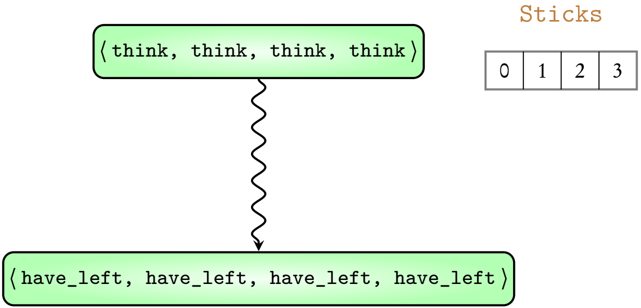
0	1	2	3

$\langle$ think, think, think, think $\rangle$

$\langle$ have_left, have_left, have_left, have_left $\rangle$



What properties should be checked in order to **reveal the deadlock**?



What properties should be checked in order to **reveal the deadlock**?

G F (phil0.location=eat) & G F (phil1.location=eat) &
G F (phil2.location=eat) & G F (phil3.location=eat)

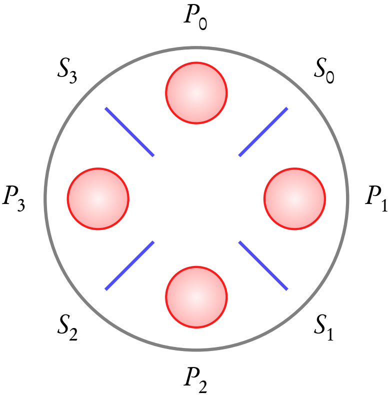
- If **counterexample** is due to only `main` process being scheduled
  - **Not a fair** scheduler

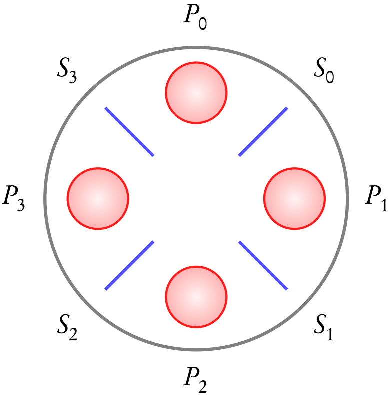
- If **counterexample** is due to only `main` process being scheduled
    - **Not a fair** scheduler
    - Add a `FAIRNESS running` in the philosopher module

- If **counterexample** is due to only `main` process being scheduled
    - **Not a fair** scheduler
    - Add a `FAIRNESS running` in the philosopher module

NuSMV demo

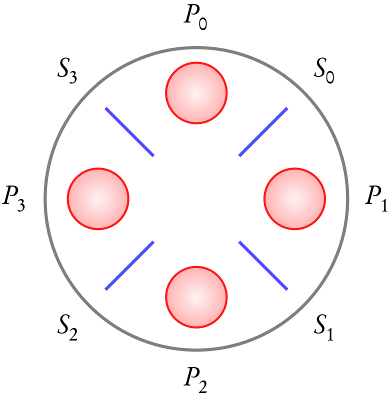
Coming next: Another solution for the dining philosophers problem


P_0
S_3
S_0
P_3
P_1
S_2
S_1
P_2

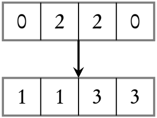


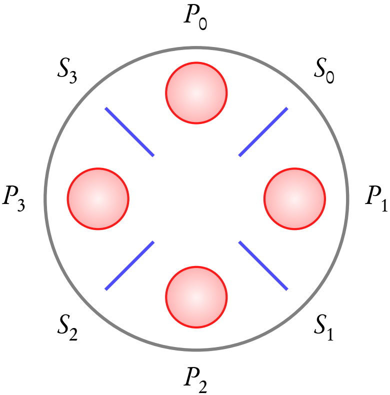
Sticks

0	2	2	0



Sticks



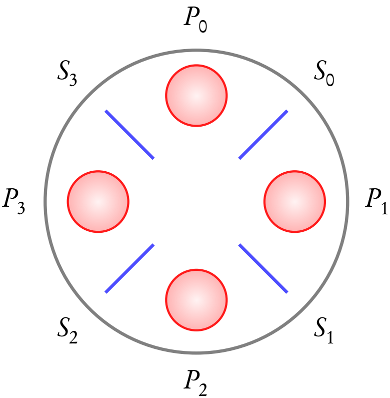


Sticks

0	2	2	0

1	1	3	3

0	2	2	0



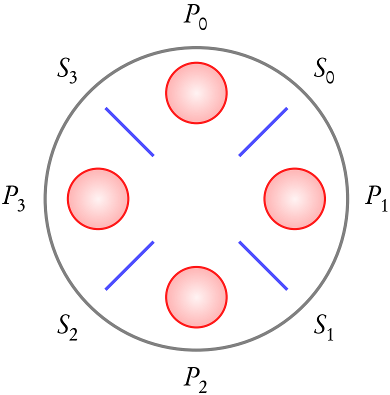
Sticks

0	2	2	0

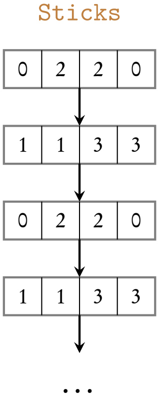
1	1	3	3

0	2	2	0

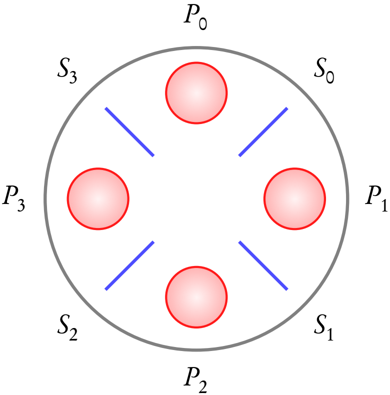
1	1	3	3

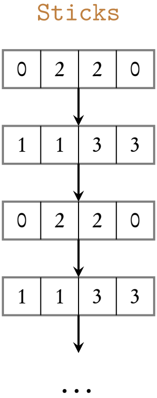


P_0
S_3
S_0
P_3
P_1
S_2
S_1
P_2



Sticks
0 2 2 0
1 1 3 3
0 2 2 0
1 1 3 3
...





This solution is deadlock-free

Liveness properties

Good happens **infinitely often**

```
FAIRNESS running
```

Unit-3: Linear time properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Summary

- Behaviour of a TS described as a set of its **traces**
- A **property** is a **set of infinite words** over *PowerSet*(AP) (Linear-time property)
- TS satisfies property if its traces are contained in the property
- Invariants, Safety, Liveness, Fairness

Important concepts: Atomic propositions, X operator, detecting deadlocks