

# Abstraction – Part 2

## Lecture #6 of Principles of Model Checking

*Joost-Pieter Katoen*

Software Modeling and Verification Group

affiliated to University of Twente, Formal Methods and Tools

University of Twente, September 29, 2010

## Content of this lecture

- **Stutter trace equivalence**
  - definition, properties, LTL (no next) equivalence
- **Stutter bisimulation**
  - definition, properties, no LTL (no next) equivalence
- **Divergence sensitivity**
  - divergence-sensitive bisimulation, CTL\* (no next) equivalence
- **Divergence-sensitive bisimulation minimisation**
  - basic idea of algorithm, complexity

---

## Content of this lecture

- ⇒ Stutter trace equivalence
  - definition, properties, LTL (no next) equivalence
- Stutter bisimulation
  - definition, properties, no LTL (no next) equivalence
- Divergence sensitivity
  - divergence-sensitive bisimulation, CTL\* (no next) equivalence
- Divergence-sensitive bisimulation minimisation
  - basic idea of algorithm, complexity

# Motivation

- Bisimulation, simulation and trace equivalence are *strong*
  - each transition  $s \rightarrow s'$  must be matched by a **transition** of a related state
  - for comparing models at different abstraction levels, this is too fine
  - consider e.g., modeling an abstract action by a sequence of concrete actions
- Idea: allow for **sequences** of “invisible” actions
  - each transition  $s \rightarrow s'$  must be matched by a **path fragment** of a related state
  - matching means: ending in a state related to  $s'$ , and all previous states invisible
- Abstraction of such internal computations yields coarser quotients
  - but: what kind of properties are preserved?
  - but: can such quotients still be obtained efficiently?
  - but: how to treat infinite internal computations?

## Motivating example

Let  $TS_{conc}$  model the concrete program fragment

```
i := y; z := 1;  
while i > 1 do  
    z := z * i; i := i - 1;  
od  
x := z;
```

that computes the factorial of  $y$  iteratively.

Let  $TS_{abs}$  be the transition system of the (abstract) program  $x := y!$

Clearly,  $TS_{abs}$  and  $TS_{conc}$  are in some sense equivalent

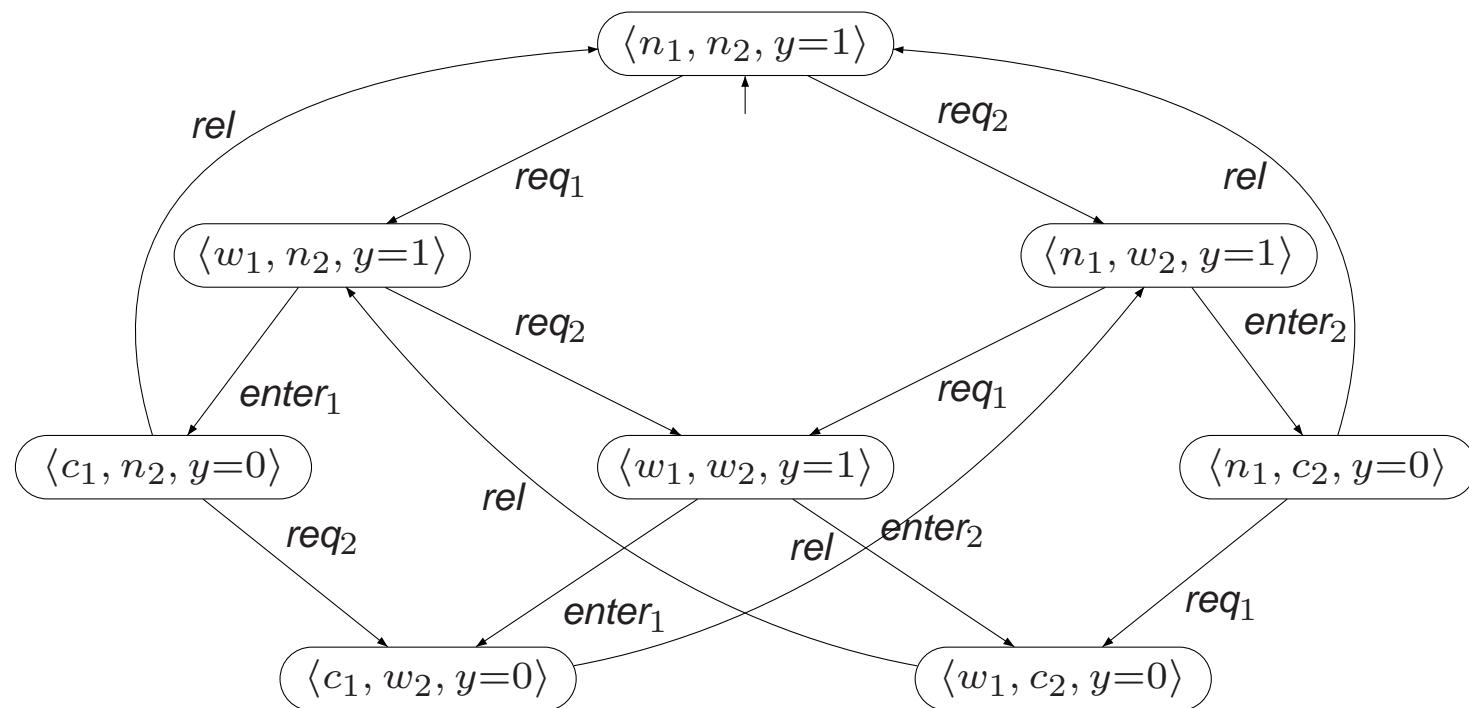
## Stuttering equivalence

- $s \rightarrow s'$  in transition system  $TS$  is a **stutter step** if  $L(s) = L(s')$
- Paths  $\pi_1$  and  $\pi_2$  are **stutter equivalent**, denoted  $\pi_1 \triangleq \pi_2$ :
  - if there exists an infinite sequence  $A_0 A_1 A_2 \dots$  with  $A_i \subseteq AP$  and
  - natural numbers  $n_0, n_1, n_2, \dots, m_0, m_1, m_2, \dots > 0$  such that:

$$\begin{aligned} \text{trace}(\pi_1) &= \underbrace{A_0 \dots A_0}_{n_0\text{-times}} \underbrace{A_1 \dots A_1}_{n_1\text{-times}} \underbrace{A_2 \dots A_2}_{n_2\text{-times}} \dots \\ \text{trace}(\pi_2) &= \underbrace{A_0 \dots A_0}_{m_0\text{-times}} \underbrace{A_1 \dots A_1}_{m_1\text{-times}} \underbrace{A_2 \dots A_2}_{m_2\text{-times}} \dots \end{aligned}$$

$\Rightarrow \pi_1 \triangleq \pi_2$  if both their traces are of the form  $A_0^+ A_1^+ A_2^+ \dots$  for  $A_i \subseteq AP$

# Semaphore-based mutual exclusion



## Stutter equivalent traces

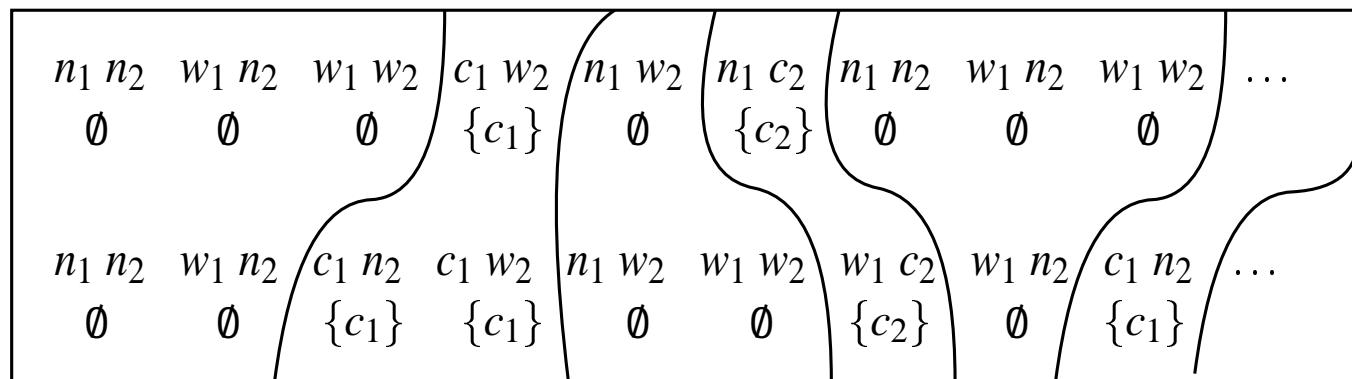
These infinite paths are stutter equivalent

$$\begin{aligned}\pi_1 &= \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle w_1, w_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \langle n_1, w_2 \rangle \rightarrow \\ &\quad \langle n_1, c_2 \rangle \rightarrow \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle w_1, w_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \dots \\ \pi_2 &= \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle c_1, n_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \langle n_1, w_2 \rangle \rightarrow \\ &\quad \langle w_1, w_2 \rangle \rightarrow \langle w_1, c_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle c_1, n_2 \rangle \rightarrow \dots\end{aligned}$$

Hence,  $\pi_1 \triangleq \pi_2$ , since for  $AP = \{ crit_1, crit_2 \}$ :

$$\begin{aligned}trace(\pi_1) &= \oslash^3 \{ crit_1 \} \oslash \{ crit_2 \} \oslash^3 \{ crit_1 \} \dots \text{ and} \\ trace(\pi_2) &= \oslash^2 (\{ crit_1 \})^2 \oslash^2 \{ crit_2 \} \oslash \{ crit_1 \} \dots\end{aligned}$$

# Pictorially



## Stutter trace equivalence

Transition systems  $TS_i$  over  $AP$ ,  $i=1, 2$ , are *stutter-trace equivalent*:

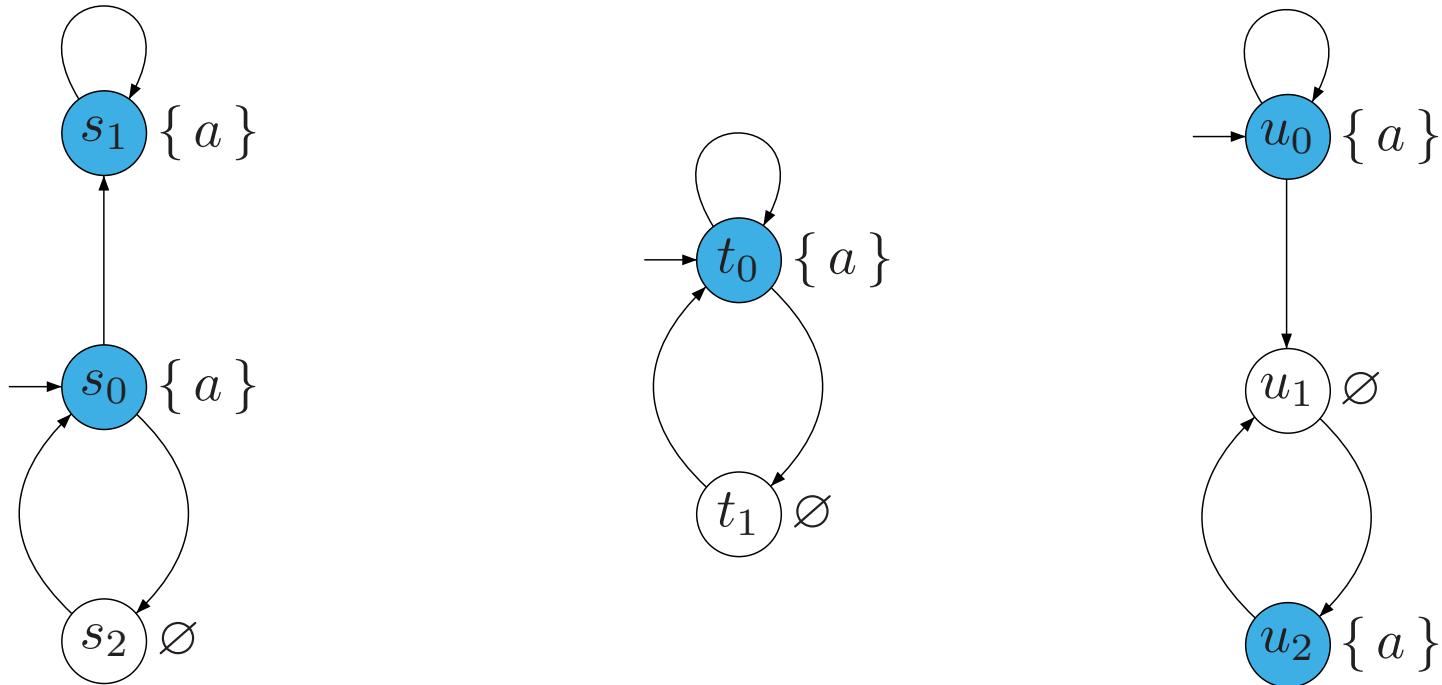
$$TS_1 \triangleq TS_2 \quad \text{if and only if} \quad TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1$$

where  $\trianglelefteq$ , pronounced *stutter trace inclusion*, is defined by:

$$TS_1 \trianglelefteq TS_2 \quad \text{iff} \quad \forall \sigma_1 \in \text{Traces}(TS_1) \left( \exists \sigma_2 \in \text{Traces}(TS_2). \ \sigma_1 \triangleq \sigma_2 \right)$$

$\text{Traces}(TS_1) = \text{Traces}(TS_2)$  implies  $TS_1 \triangleq TS_2$ , but not always the converse

## Example



$TS_1 \triangleq TS_2$ ,  $TS_1 \not\leq TS_3$  and  $TS_2 \not\leq TS_3$ , but  $TS_3 \trianglelefteq TS_2$  and  $TS_3 \trianglelefteq TS_1$

## The $\bigcirc$ operator

Stuttering equivalence does **not** preserve the validity of next-formulas:

$\sigma_1 = ABBB\dots$  and  $\sigma_2 = AAABBBB\dots$  for  $A, B \subseteq AP$  and  $A \neq B$

Then for  $b \in B \setminus A$ :

$$\sigma_1 \triangleq \sigma_2 \quad \text{but} \quad \sigma_1 \models \bigcirc b \quad \text{and} \quad \sigma_2 \not\models \bigcirc b.$$

$\Rightarrow$  a logical characterization of  $\triangleq$  can only be obtained by omitting  $\bigcirc$

in fact, it turns out that this is the only modal operator that is not preserved by  $\triangleq$ !

## Stutter trace and $\text{LTL}_{\setminus \bigcirc}$ equivalence

For traces  $\sigma_1$  and  $\sigma_2$  over  $2^{AP}$  it holds:

$\sigma_1 \triangleq \sigma_2 \Rightarrow (\sigma_1 \models \varphi \text{ if and only if } \sigma_2 \models \varphi)$   
for any  $\text{LTL}_{\setminus \bigcirc}$  formula  $\varphi$  over  $AP$

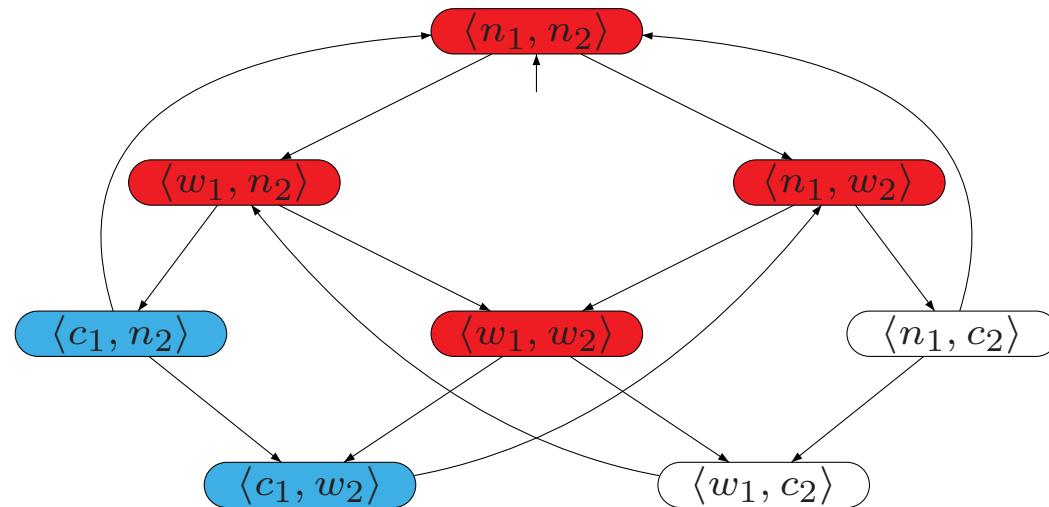
$\text{LTL}_{\setminus \bigcirc}$  denotes the class of LTL formulas without the next operator  $\bigcirc$

## Stutter trace and $LTL_{\setminus \bigcirc}$ equivalence

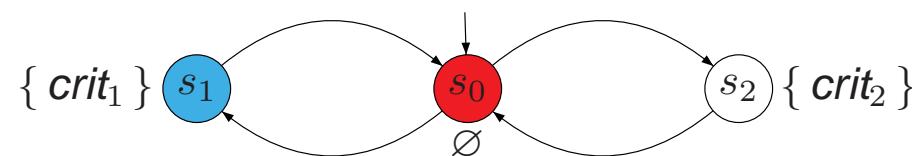
For transition systems  $TS_1$ ,  $TS_2$  without terminal states:

- (a)  $TS_1 \triangleq TS_2$  if and only if  $(TS_1 \equiv_{LTL \setminus \bigcirc} TS_2)$
- (b) if  $TS_1 \trianglelefteq TS_2$  then for any  $LTL_{\setminus \bigcirc}$  formula  $\varphi$ :  $TS_2 \models \varphi$  implies  $TS_1 \models \varphi$

## Semaphore-based mutual exclusion



This transition system is stutter trace-equivalent:



# Content of this lecture

- Stutter trace equivalence
  - definition, properties, LTL (no next) equivalence
- ⇒ Stutter bisimulation
  - definition, properties, no LTL (no next) equivalence
- Divergence sensitivity
  - divergence-sensitive bisimulation, CTL\* (no next) equivalence
- Divergence-sensitive bisimulation minimisation
  - basic idea of algorithm, complexity

## Stutter bisimulation

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  be a transition system and  $\mathcal{R} \subseteq S \times S$   
 $\mathcal{R}$  is a *stutter-bisimulation* for  $TS$  if for all  $(s_1, s_2) \in \mathcal{R}$ :

1.  $L(s_1) = L(s_2)$
2. if  $s'_1 \in Post(s_1)$  with  $(s'_1, s_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_2 u_1 \dots u_n s'_2$  with  $n \geq 0$  and  $(s_1, u_i) \in \mathcal{R}$  and  $(s'_1, s'_2) \in \mathcal{R}$
3. if  $s'_2 \in Post(s_2)$  with  $(s_1, s'_2) \notin \mathcal{R}$ , then there exists a finite path fragment  $s_1 v_1 \dots v_n s'_1$  with  $n \geq 0$  and  $(s_2, v_i) \in \mathcal{R}$  and  $(s'_1, s'_2) \in \mathcal{R}$

$s_1, s_2$  are *stutter-bisimulation equivalent*, denoted  $s_1 \approx_{TS} s_2$ ,  
if there exists a stutter bisimulation  $\mathcal{R}$  for  $TS$  with  $(s_1, s_2) \in \mathcal{R}$

# Stutter bisimulation

$$s_1 \approx s_2$$

$$\downarrow$$

$$s'_1$$

(with  $s'_1 \not\approx s_2$ )

can be completed to

$$s_1 \approx s_2$$

$$\downarrow$$

$$s_1 \approx u_1$$

$$\downarrow$$

$$s_1 \approx u_2$$

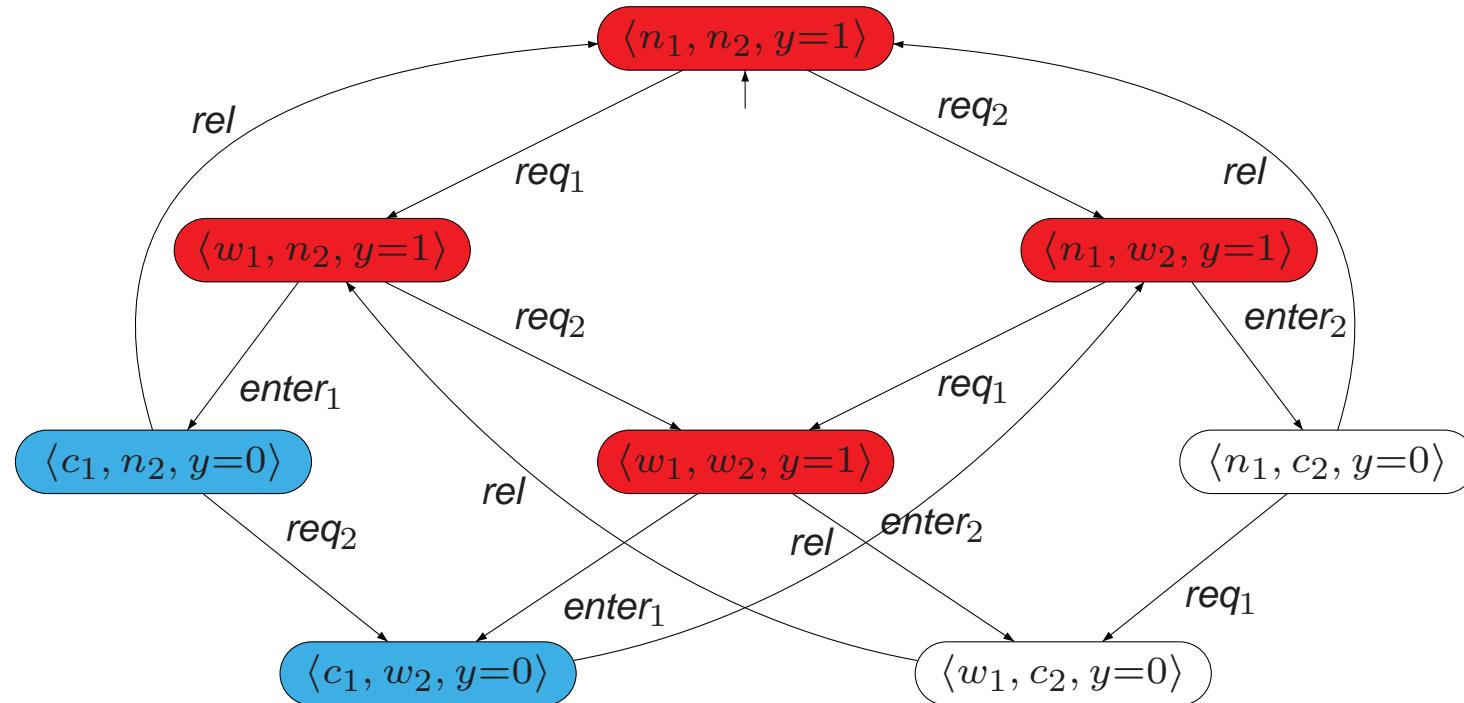
$$\vdots$$

$$s_1 \approx u_n$$

$$\downarrow$$

$$s'_1 \approx s'_2$$

# Semaphore-based mutual exclusion



stutter-bisimilar states for  $AP = \{ crit_1, crit_2 \}$

## Stutter-bisimilar transition systems

Let  $TS_i = (S_i, \mathbf{Act}_i, \rightarrow_i, I_i, \mathbf{AP}, L_i)$ ,  $i = 1, 2$ , be transition systems

$TS_1$  and  $TS_2$  are stutter bisimilar, denoted  $TS_1 \approx TS_2$ , if there exists a stutter bisimulation  $\mathcal{R}$  on  $TS_1 \oplus TS_2$  such that:

$$\forall s_1 \in I_1. (\exists s_2 \in I_2. (s_1, s_2) \in \mathcal{R}) \text{ and } \forall s_2 \in I_2. (\exists s_1 \in I_1. (s_1, s_2) \in \mathcal{R})$$

## Stutter bisimulation quotient

Let  $TS = (S, Act, \rightarrow, I, AP, L)$  and stutter bisimulation  $\mathcal{R} \subseteq S \times S$  be an *equivalence*

The *quotient* of  $TS$  under  $\mathcal{R}$  is defined by:

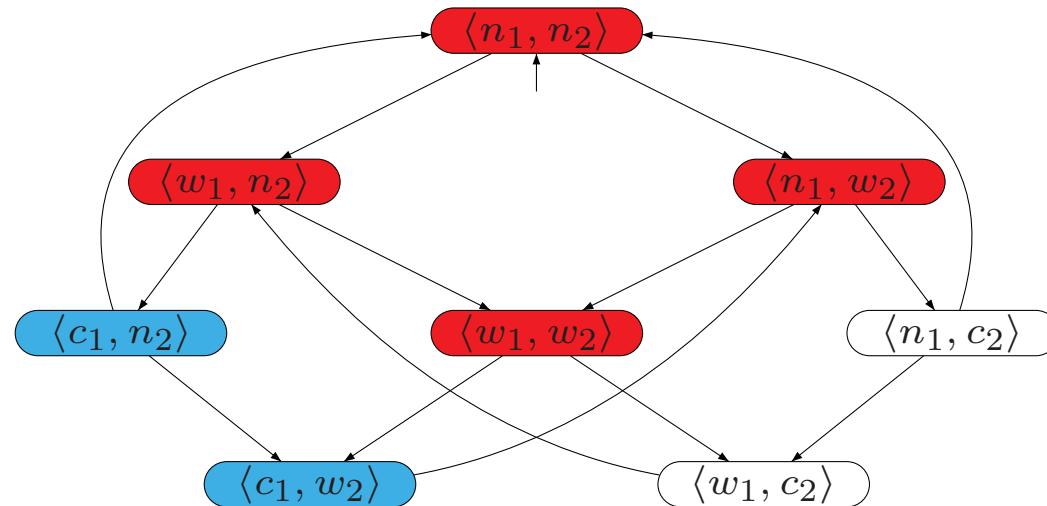
$$TS/\mathcal{R} = (S', \{\tau\}, \rightarrow', I', AP, L')$$

where

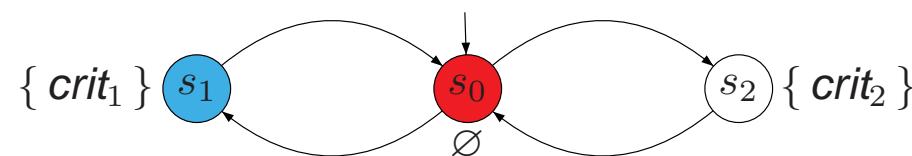
- $S' = S/\mathcal{R} = \{[s]_{\mathcal{R}} \mid s \in S\}$  with  $[s]_{\mathcal{R}} = \{s' \in S \mid (s, s') \in \mathcal{R}\}$
- $I' = \{[s]_{\mathcal{R}} \mid s \in I\}$
- $L'([s]_{\mathcal{R}}) = L(s)$
- $\rightarrow'$  is defined by: 
$$\frac{s \xrightarrow{\alpha} s' \text{ and } (s, s') \notin \mathcal{R}}{[s]_{\mathcal{R}} \xrightarrow{\tau'} [s']_{\mathcal{R}}}$$

note that (a) no self-loops occur in  $TS/\approx_{TS}$  and (b)  $TS \approx TS/\approx_{TS}$

## Semaphore-based mutual exclusion



The stutter-bisimulation quotient:

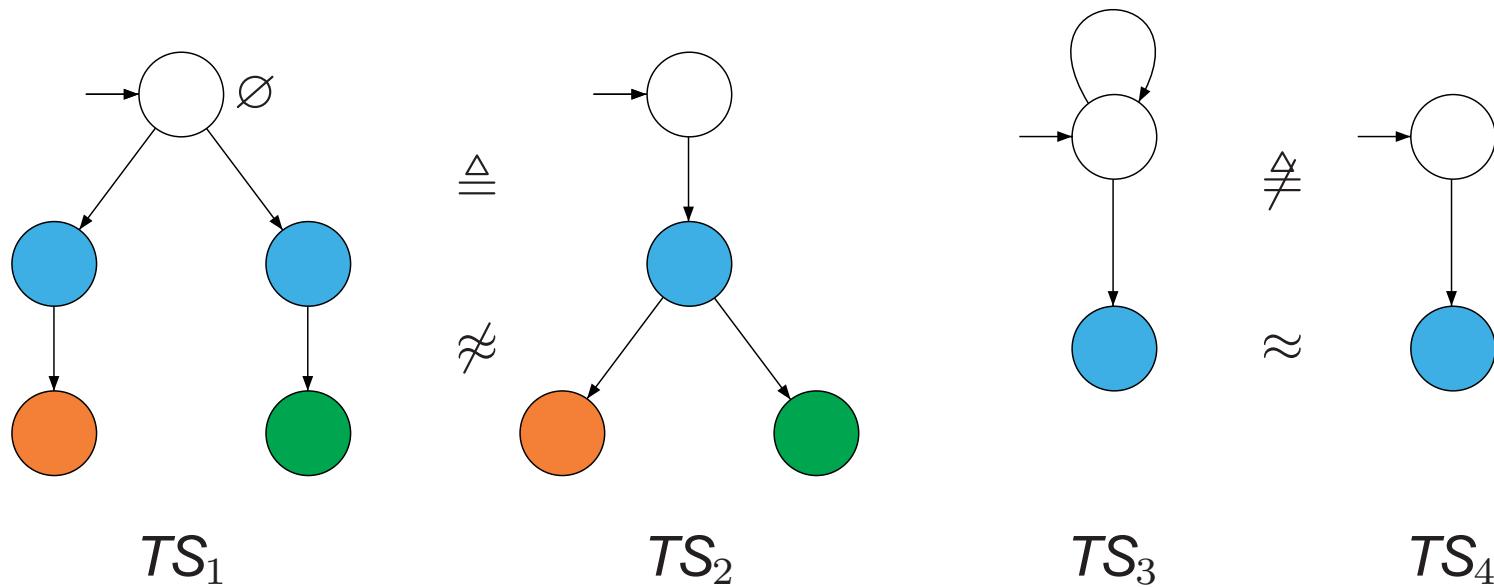


## Stutter trace and stutter bisimulation

For transition systems  $TS_1$  and  $TS_2$  over  $AP$ :

- Known fact:  $TS_1 \sim TS_2$  implies  $Traces(TS_1) = Traces(TS_2)$
- But *not*:  $TS_1 \approx TS_2$  implies  $TS_1 \triangleq TS_2$ !
- So:
  - bisimilar transition systems are trace equivalent
  - **but** stutter-bisimilar transition systems are not always stutter trace-equivalent!
- Why? Paths that only stutter!
  - stutter bisimulation does not impose any constraint on such paths
  - **but**  $\triangleq$  requires the existence of a stuttering equivalent trace

## Stutter trace and stutter bisimulation are incomparable



## Stutter bisimulation does not preserve LTL<sub>\lozenge</sub>



$TS_{left} \approx TS_{right}$  but  $TS_{left} \not\models \lozenge a$  and  $TS_{right} \models \lozenge a$

reason: presence of infinite stutter paths in  $TS_{left}$

# Content of this lecture

- Stutter trace equivalence
  - definition, properties, LTL (no next) equivalence
- Stutter bisimulation
  - definition, properties, no LTL (no next) equivalence

⇒ Divergence sensitivity

- divergence-sensitive bisimulation, CTL\* (no next) equivalence

- Divergence-sensitive bisimulation minimisation
  - basic idea of algorithm, complexity

## Divergence sensitivity

- *Stutter paths* are paths that only consist of stutter steps
  - no restrictions are imposed on such paths by a stutter bisimulation
- Stutter paths *diverge*: they never leave an equivalence class
- Remedy: only relate *divergent* states or *non-divergent* states
  - divergent state = a state that has a stutter path
  - ⇒ relate states only if they either both have stutter paths or none of them
- This yields *divergence-sensitive stutter bisimulation* ( $\approx^{\text{div}}$ )
  - ⇒  $\approx^{\text{div}}$  is strictly finer than  $\triangleq$  (and  $\approx$ )

# Outlook

formal relation	trace equivalence	bisimulation	simulation
complexity	PSPACE-complete	PTIME	PTIME
logical fragment	LTL	CTL*	$\forall$ CTL*
preservation	strong	strong match	weak match

formal relation	stutter trace equivalence	divergence-sensitive stutter bisimulation
complexity	PSPACE-complete	PTIME
logical fragment	$LTL_{\setminus \bigcirc}$	$CTL^*_{\setminus \bigcirc}$
preservation	strong	strong match

## Divergence sensitivity

Let  $TS$  be a transition system and  $\mathcal{R}$  an equivalence relation on  $S$

- $s$  is  **$\mathcal{R}$ -divergent** if there exists an infinite path fragment  $s s_1 s_2 \dots \in \text{Paths}(s)$  such that  $(s, s_j) \in \mathcal{R}$  for all  $j > 0$ 
  - $s$  is  $\mathcal{R}$ -divergent if there is an infinite path starting in  $s$  that only visits  $[s]_{\mathcal{R}}$
- $\mathcal{R}$  is **divergence sensitive** if for any  $(s_1, s_2) \in \mathcal{R}$ :
  - $s_1$  is  $\mathcal{R}$ -divergent implies  $s_2$  is  $\mathcal{R}$ -divergent
  - $\mathcal{R}$  is divergence-sensitive if in any  $[s]_{\mathcal{R}}$  either all or none states are  $\mathcal{R}$ -divergent

## Divergent-sensitive stutter bisimulation

$s_1, s_2$  are *divergent-sensitive stutter-bisimilar*, denoted  $s_1 \approx_{TS}^{div} s_2$ , if:

$\exists$  divergent-sensitive stutter bisimulation  $\mathcal{R}$  on  $TS$  such that  $(s_1, s_2) \in \mathcal{R}$

$\approx_{TS}^{div}$  is an equivalence, the coarsest divergence-sensitive stutter bisimulation for  $TS$

and the union of all divergence-sensitive stutter bisimulations for  $TS$

## Quotient transition system under $\approx^{\text{div}}$

$TS/\approx^{\text{div}} = (S', \{\tau\}, \rightarrow', I', AP, L')$ , the *quotient* of  $TS$  under  $\approx^{\text{div}}$

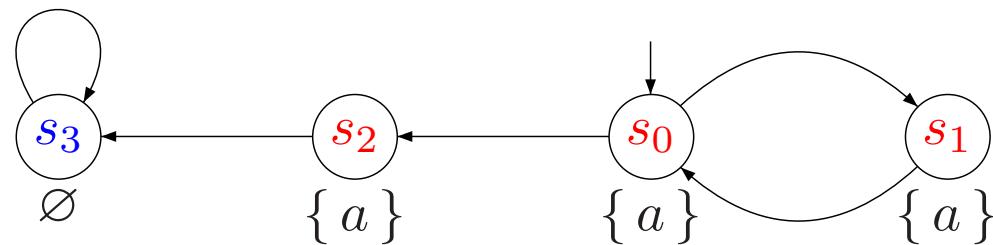
where

- $S'$ ,  $I'$  and  $L'$  are defined as usual (for eq. classes  $[s]_{\text{div}}$  under  $\approx^{\text{div}}$ )
- $\rightarrow'$  is defined by:

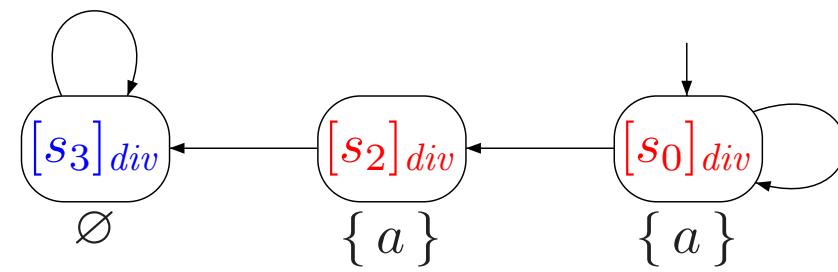
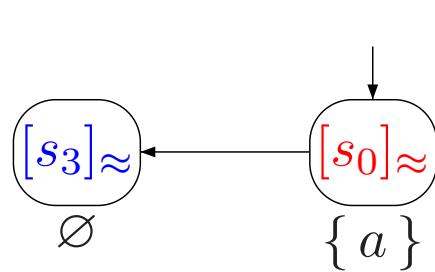
$$\frac{s \xrightarrow{\alpha} s' \wedge s \not\approx^{\text{div}} s'}{[s]_{\text{div}} \xrightarrow[\text{div}]{\tau} [s']_{\text{div}}} \quad \text{and} \quad \frac{s \text{ is } \approx^{\text{div}}\text{-divergent}}{[s]_{\text{div}} \xrightarrow[\text{div}]{\tau} [s]_{\text{div}}}$$

note that  $TS \approx^{\text{div}} TS/\approx^{\text{div}}$

## Example



transition system  $TS$



# Summary

*stutter trace inclusion:*

$$TS_1 \trianglelefteq TS_2 \quad \text{iff} \quad \forall \sigma_1 \in \text{Traces}(TS_1) \ \exists \sigma_2 \in \text{Traces}(TS_2). \ \sigma_1 \triangleq \sigma_2$$

*stutter trace equivalence:*

$$TS_1 \triangleq TS_2 \quad \text{iff} \quad TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1$$

*stutter bisimulation equivalence:*

$$TS_1 \approx TS_2 \quad \text{iff} \quad \text{there exists a stutter bisimulation for } (TS_1, TS_2)$$

*stutter bisimulation equivalence with divergence:*

$$TS_1 \approx^{\text{div}} TS_2 \quad \text{iff} \quad \text{there exists a divergence-sensitive}\\ \text{stutter bisimulation for } (TS_1, TS_2)$$

## $CTL_{\setminus \bigcirc}^*$ and $CTL_{\setminus \bigcirc}$ equivalence vs $\approx^{div}$

For finite transition system  $TS$  without terminal states, and  $s_1, s_2$  in  $TS$ :

$$s_1 \approx_{TS}^{div} s_2 \quad \text{iff} \quad s_1 \equiv_{CTL_{\setminus \bigcirc}^*} s_2 \quad \text{iff} \quad s_1 \equiv_{CTL_{\setminus \bigcirc}} s_2$$

## CTL\*<sub>\bigcirc</sub> and CTL<sub>\bigcirc</sub> equivalence vs $\approx^{\text{div}}$

For finite transition system  $TS$  without terminal states, and  $s_1, s_2$  in  $TS$ :

$$s_1 \approx_{TS}^{\text{div}} s_2 \quad \text{iff} \quad s_1 \equiv_{CTL^*_{\bigcirc}} s_2 \quad \text{iff} \quad s_1 \equiv_{CTL_{\bigcirc}} s_2 \quad \text{iff} \quad s_1 \equiv_{CTL_{\bigcirc, \cup}} s_2$$

# A producer-consumer example

## Producer

```

 $in := 0;$ 
while true {
    produce  $d_1, \dots, d_n$ ;
    for  $i = 1$  to  $n$  {
        wait until ( $buffer[in] = \perp$ ) {
             $buffer[in] := d_i$ ;
             $in := (in + 1) \bmod m$ ; }
    }
}

```

## Consumer

```

 $out := 0;$ 
while true {
    for  $j = 1$  to  $n$  {
        wait until ( $buffer[out] \neq \perp$ ) {
             $e_j := buffer[out]$ ;
             $buffer[out] := \perp$ ;
             $out := (out + 1) \bmod m$ ; }
    }
    consume  $e_1, \dots, e_n$ 
}

```

# An abstraction

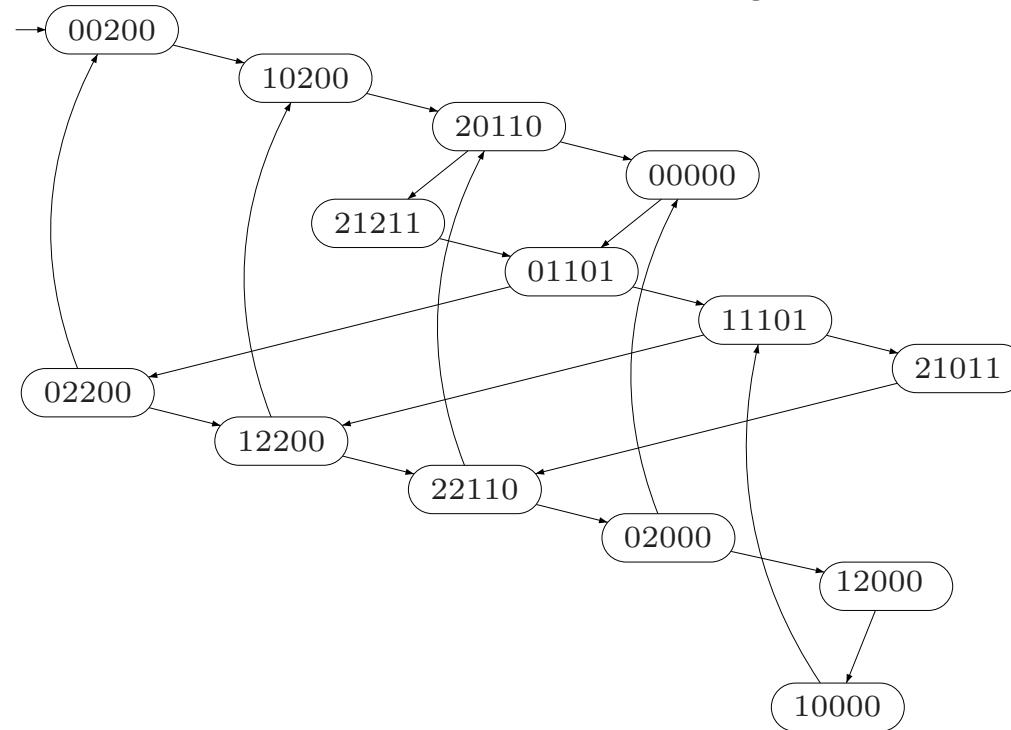
## Producer

```
while true {  
    produce;  
    for i = 1 to n {  
        wait until (free > 0) {  
            free := free - 1;  
        }  
    }  
}
```

## Consumer

```
while true {  
    for j = 1 to n {  
        wait until (free < m) {  
            free := free + 1;  
        }  
    }  
    consume  
}
```

## Abstract transition system

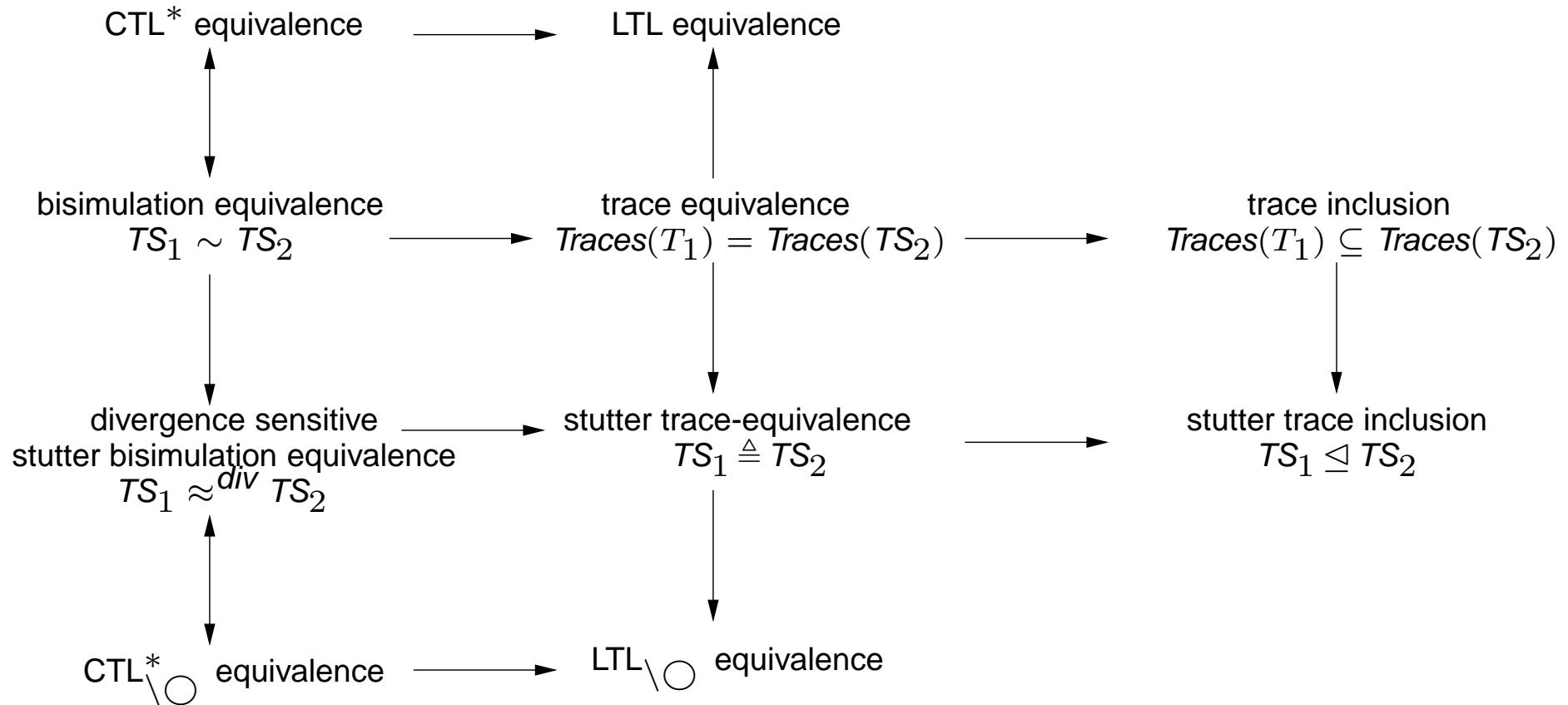


$\ell_0$  : *produce*

$\ell_1$  :  $\langle \text{if } (\text{free} > 0) \text{ then } i := 1; \text{free}-- \text{ fi} \rangle$

$\ell_2$  :  $\langle \text{if } (\text{free} > 0) \text{ then } i := 0; \text{free}-- \text{ fi} \rangle ; \text{ goto } \ell_0$

# Equivalences and logical equivalence



---

## Content of this lecture

- **Stutter trace equivalence**
  - definition, properties, LTL (no next) equivalence
- **Stutter bisimulation**
  - definition, properties, no LTL (no next) equivalence
- **Divergence sensitivity**
  - divergence-sensitive bisimulation, CTL\* (no next) equivalence

⇒ **Divergence-sensitive bisimulation minimisation**

- basic idea of algorithm, complexity

## Quotienting: Motivation

- Quotienting wrt.  $\approx^{\text{div}}$  allows to *abstract from stutter steps*
  - in particular  $TS \approx^{\text{div}} TS/\approx^{\text{div}}$
  - typically we have  $|TS| \gg |TS/\approx^{\text{div}}|$
- $TS_1 \approx^{\text{div}} TS_2$  if and only if  $(TS_1 \models \Phi \text{ iff } TS_2 \models \Phi)$ 
  - for any  $\text{CTL}_{\setminus \bigcirc}^*$  (or  $\text{CTL}_{\setminus \bigcirc}$ ) formula  $\Phi$

$\Rightarrow$  To check  $TS \models \Phi$ , it suffices to check whether  $TS/\approx^{\text{div}} \models \Phi$

- quotienting with respect to  $\approx^{\text{div}}$  is a useful preprocessing step of model checking

# Quotienting: A two-phase approach

[Groote and Vaandrager, 1990]

## 1. A quotienting algorithm to determine $TS/\approx$ :

- remove *stutter cycles* from  $TS$
- a refine operator to *efficiently split* (blocks of) partitions
- exploit partition-refinement (as for bisimulation  $\sim$ )

## 2. A quotienting algorithm to determine $TS/\approx^{\text{div}}$ :

- *transform*  $TS$  into a (divergence-sensitive) transition system  $\overline{TS}$
- $\overline{TS}$  is divergent-sensitive, i.e.,  $\approx_{\overline{TS}}$  and  $\approx_{\overline{TS}}^{\text{div}}$  coincide
- determine  $\overline{TS}/\approx$  using the quotienting algorithm for  $\approx$
- “distill”  $TS/\approx^{\text{div}}$  from  $\overline{TS}/\approx$

## Partition-refinement

from now on, we assume that  $TS$  is finite

- Iteratively compute a partition of  $S$
- Initially:  $\Pi_0$  equals  $\Pi_{AP} = \{ (s, t) \in S \times S \mid L(s) = L(t) \}$  as before
- Repeat until no change:  $\Pi_{i+1} := \text{Refine}_{\approx}(\Pi_i)$ 
  - loop invariant:  $\Pi_i$  is coarser than  $S/\approx$  and finer than  $\{ S \}$
- Return  $\Pi_i$ 
  - termination:  $\mathcal{R}_{\Pi_0} \supsetneq \mathcal{R}_{\Pi_1} \supsetneq \mathcal{R}_{\Pi_2} \supsetneq \dots \supsetneq \mathcal{R}_{\Pi_i} = \approx_{TS}$
  - time complexity: maximally  $|S|$  iterations needed

## Theorem

$S/\approx$  is the *coarsest* partition  $\Pi$  of  $S$  such that:

- (i)  $\Pi$  is finer than the initial partition  $\Pi_{AP}$ , and
- (ii)  $B \cap \text{Pre}_\Pi^*(C) = \emptyset$  or  $B \subseteq \text{Pre}_\Pi^*(C)$  for all  $B, C \in \Pi$

for partition  $\Pi$  of  $S$  and blocks  $B, C$  in  $\Pi$  we have:

$s \in \text{Pre}_\Pi^*(C)$  whenever  $\underbrace{s_1 s_2 \dots s_{n-1}}_{\in B} \underbrace{s_n}_{\in C} \in \text{Paths}(s)$

state  $s$  can reach  $C$  via a path that is completely in  $B$  ( $= [s]_\Pi$ )

## The refinement operator

- Let:  $\text{Refine}_{\approx}(\Pi, C) = \bigcup_{B \in \Pi} \text{Refine}_{\approx}(B, C)$  for  $C$  a block in  $\Pi$

– where  $\text{Refine}_{\approx}(B, C) = \{B \cap \text{Pre}_{\Pi}^*(C), B \setminus \text{Pre}_{\Pi}^*(C)\} \setminus \{\emptyset\}$

- Basic properties:

– for  $\Pi$  finer than  $\Pi_{AP}$  and coarser than  $S/\approx$ :

$\text{Refine}_{\approx}(\Pi, C)$  is finer than  $\Pi$  and  $\text{Refine}_{\approx}(\Pi, C)$  is coarser than  $S/\approx$

–  $\Pi$  is strictly coarser than  $S/\approx$  if and only if there exists a *splitter* for  $\Pi$

what is an appropriate splitter for  $\approx$ ?

## Splitter for $\approx$

Let  $\Pi$  be a partition of  $S$  and let  $C, B \in \Pi$ .

1.  $C$  is a  **$\Pi$ -splitter** for  $B$  if and only if:

$$B \neq C \quad \text{and} \quad B \cap \text{Pre}_\Pi^*(C) \neq \emptyset \quad \text{and} \quad B \setminus \text{Pre}_\Pi^*(C) \neq \emptyset$$

2.  $\Pi$  is  **$C$ -stable** if there is no  $B \in \Pi$  such that  $C$  is a  $\Pi$ -splitter for  $B$
3.  $\Pi$  is **stable** if  $\Pi$  is  $C$ -stable for all blocks  $C \in \Pi$

# Partition-refinement

*Input:* finite transition system  $TS$  with state space  $S$

*Output:* stutter-bisimulation quotient space  $S/\approx$

---

```

 $\Pi := \Pi_{AP}$ ;                                     (* as before *)
while ( $\exists B, C \in \Pi$ .  $C$  is a  $\Pi$ -splitter for  $B$ ) do
  choose such  $B, C \in \Pi$ ;
   $\Pi := (\Pi \setminus \{B\}) \cup \{ \underbrace{B \cap \text{Pre}_\Pi^*(C)}_{B_1}, \underbrace{B \setminus \text{Pre}_\Pi^*(C)}_{B_2} \} \setminus \{ \emptyset \}$ ;    (* refine  $\Pi$  *)
od
return  $\Pi$ 

```

## Stutter cycles

- $s_0 s_1 \dots \underbrace{s_n}_{= s_0}$  is a *stutter cycle* if  $s_i s_{i+1}$  is a stutter step
- For stutter cycle  $s_0 s_1 s_2 \dots s_n$  in transition system  $TS$ :

$$s_0 \approx_{TS}^{div} s_1 \approx_{TS}^{div} \dots \approx_{TS}^{div} s_n$$

- Corollary: for finite  $TS$  and state  $s$  in  $TS$ :

$s$  is  $\approx^{div}$  –divergent if and only if  
a stutter cycle is reachable from  $s$  via a path in  $[s]_{div}$

## Removal of stutter cycles: How?

1. Determine the SCCs in  $G(TS)$  that only contain stutter steps
  - use depth-first search to find these strongly connected components (SCCs)
2. Collapse any stutter SCC into a single state
  - $C \rightarrow' C'$  with  $C \neq C'$  whenever  $s \rightarrow s'$  in  $TS$  with  $s \in C$  and  $s' \in C'$

⇒ Resulting  $TS'$  has no stutter cycles

- $s_1 \approx_{TS} s_2$  if and only if  $\underbrace{C_1}_{s_1 \in C_1} \approx_{TS'} \underbrace{C_2}_{s_2 \in C_2}$

from now on, assume transition systems have **no** stutter cycles

## A “local” splitter characterization

- $C$  is a  $\Pi$ -splitter for  $B$  if and only if:

$$B \neq C \quad \text{and} \quad B \cap \text{Pre}_\Pi^*(C) \neq \emptyset \quad \text{and} \quad B \setminus \text{Pre}_\Pi^*(C) \neq \emptyset$$

- How to avoid the computation of  $\text{Pre}_\Pi^*(C)$  for  $C \in \Pi$ ?
- No stutter cycles  $\Rightarrow$  block  $B \in \Pi$  has at least one *exit state*
  - exit state = a state with only direct successors outside  $B$ :

$$\text{Bottom}(B) = \{s \in B \mid \text{Post}(s) \cap B = \emptyset\}$$

- For finite TS without stutter cycles,  $C$  is a  $\Pi$ -splitter for  $B$  iff:

$$B \neq C \quad \text{and} \quad B \cap \text{Pre}(C) \neq \emptyset \quad \text{and} \quad \text{Bottom}(B) \setminus \text{Pre}(C) \neq \emptyset$$

# Time complexity

The partition-refinement algorithm to compute  $TS/\approx$   
has a worst-case time complexity in  $\mathcal{O}\left(|S| \cdot (|AP| + M)\right)$

# Approach

1. A quotienting algorithm to determine  $TS/\approx$ :

- remove *stutter cycles* from  $TS$
- a refine operator to *efficiently split* (blocks of) partitions
- exploit partition-refinement (as for bisimulation  $\sim$ )

⇒ A quotienting algorithm to determine  $TS/\approx^{\text{div}}$ :

- *transform*  $TS$  into a (divergence-sensitive) transition system  $\overline{TS}$
- $\overline{TS}$  is divergent-sensitive, i.e.,  $\approx_{\overline{TS}}$  and  $\approx_{\overline{TS}}^{\text{div}}$  coincide
- determine  $\overline{TS}/\approx$  using the quotienting algorithm for  $\approx$
- “distill”  $TS/\approx^{\text{div}}$  from  $\overline{TS}/\approx$

## Divergence expansion

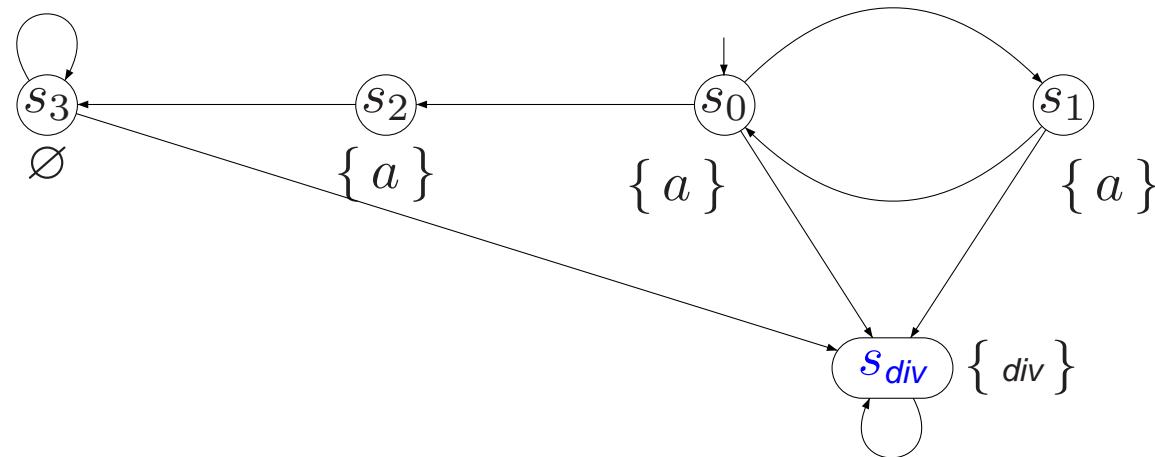
*Divergence-sensitive expansion* of finite  $TS = (S, Act, \rightarrow, I, AP, L)$  is:

$$\overline{TS} = (S \cup \{s_{div}\}, Act \cup \{\tau\}, \rightarrow, I, AP \cup \{\text{div}\}, \overline{L}) \quad \text{where}$$

- $s_{div} \notin S$
- $\rightarrow$  extends the transition relation of  $TS$  by:
  - $s_{div} \xrightarrow{\tau} s_{div}$  and
  - $s \xrightarrow{\tau} s_{div}$  for every state  $s \in S$  on a stutter cycle in  $TS$
- $\overline{L}(s) = L(s)$  if  $s \in S$  and  $\overline{L}(s_{div}) = \{\text{div}\}$

$s_{div} \not\approx s$  for any  $s \in S$  and  $s_{div}$  can only be reached from a  $\approx^{\text{div}}$ -divergent state

## Example



# Correctness

For finite transition system  $TS$ :

1.  $\overline{TS}$  is divergence-sensitive, and
2. for all  $s_1, s_2 \in S$ :  $s_1 \approx_{TS}^{div} s_2$  if and only if  $s_1 \approx_{\overline{TS}} s_2$

## Recipe for computing $TS/\approx^{\text{div}}$

### 1. Construct the divergence-sensitive expansion $\overline{TS}$

- determine the SCCs in  $G_{\text{stutter}}(TS)$ , and insert transitions  $s_{\text{div}} \rightarrow s_{\text{div}}$  and
- $s \rightarrow s_{\text{div}}$  for any state  $s$  in a non-trivial SCC of  $G_{\text{stutter}}$

### 2. Apply partition-refinement to $\overline{TS}$ to obtain $S/\approx_{TS}^{\text{div}} = S/\approx_{\overline{TS}}$

### 3. Generate $\overline{TS}/\approx$

- any  $C \in S/\approx^{\text{div}}$  that contains an initial state of  $TS$  is an initial state
- the labeling of  $C \in S/\approx^{\text{div}}$  equals the labeling of any  $s \in C$
- any transition  $s \rightarrow s'$  with  $s \not\approx_{TS}^{\text{div}} s'$  yields a transition between  $C_s$  and  $C_{s'}$

### 4. “Distill” $TS/\approx^{\text{div}}$ from $\overline{TS}/\approx$ :

- replace transition  $s \rightarrow s_{\text{div}}$  in  $\overline{TS}$  by the self-loop  $[s]_{\approx^{\text{div}}} \rightarrow [s]_{\approx^{\text{div}}}$
- delete state  $s_{\text{div}}$

## Time complexity

The quotient transition system  $TS/\approx^{\text{div}}$  can be determined  
with a worst-case time complexity in  $\mathcal{O}\left(|S| \cdot (|AP| + M)\right)$

# Summary

formal relation	trace equivalence	bisimulation	simulation
complexity	PSPACE-complete	$\mathcal{O}(M \cdot \log  S )$	$\mathcal{O}(M \cdot  S )$
logical fragment	LTL	CTL*	$\forall$ CTL*
preservation	strong	strong match	weak match

formal relation	stutter trace equivalence	divergence-sensitive stutter bisimulation
complexity	PSPACE-complete	$\mathcal{O}(M \cdot  S )$
logical fragment	$LTL_{\setminus \bigcirc}$	$CTL_{\setminus \bigcirc}^*$
preservation	strong	strong match