Stutter Equivalences
Lecture #4–#6 of Advanced Model Checking

Joost-Pieter Katoen
Software Modeling and Verification Group
affiliated to University of Twente, Formal Methods and Tools

October 27, 2016
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” transitions
  - 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

\[
i := y; \quad z := 1; \\
\text{while } i > 1 \text{ do} \\
\quad z := z \times i; \quad i := i - 1; \\
\text{od} \\
x := z;
\]

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 $A_0A_1A_2\ldots$ with $A_i \subseteq AP$ and  
  - natural numbers $n_0, n_1, n_2, \ldots, m_0, m_1, m_2, \ldots > 0$ such that:
    
    $\text{trace}(\pi_1) = \underbrace{A_0 \ldots A_0}_{n_0\text{-times}} \underbrace{A_1 \ldots A_1}_{n_1\text{-times}} \underbrace{A_2 \ldots A_2}_{n_2\text{-times}} \ldots$
    
    $\text{trace}(\pi_2) = \underbrace{A_0 \ldots A_0}_{m_0\text{-times}} \underbrace{A_1 \ldots A_1}_{m_1\text{-times}} \underbrace{A_2 \ldots A_2}_{m_2\text{-times}} \ldots$

$\Rightarrow \pi_1 \triangleq \pi_2$ if both their traces are of the form $A_0^+A_1^+A_2^+\ldots$ for $A_i \subseteq AP$
Semaphore-based mutual exclusion

\[ \langle n_1, n_2, y=1 \rangle \]

\[ \langle w_1, n_2, y=1 \rangle \]

\[ \langle n_1, w_2, y=1 \rangle \]

\[ \langle c_1, n_2, y=0 \rangle \]

\[ \langle w_1, w_2, y=1 \rangle \]

\[ \langle n_1, c_2, y=0 \rangle \]

\[ \langle c_1, w_2, y=0 \rangle \]

\[ \langle w_1, c_2, y=0 \rangle \]
Stutter equivalent traces

These infinite paths are stutter equivalent

$$\pi_1 = \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle w_1, w_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \langle n_1, w_2 \rangle \rightarrow \langle n_1, c_2 \rangle \rightarrow \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle w_1, w_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \ldots$$

$$\pi_2 = \langle n_1, n_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle c_1, n_2 \rangle \rightarrow \langle c_1, w_2 \rangle \rightarrow \langle n_1, n_2 \rangle \rightarrow \langle w_1, w_2 \rangle \rightarrow \langle w_1, c_2 \rangle \rightarrow \langle w_1, n_2 \rangle \rightarrow \langle c_1, n_2 \rangle \rightarrow \ldots$$

Hence, $\pi_1 \equiv \pi_2$, since for $AP = \{ \text{crit}_1, \text{crit}_2 \}$:

$$\text{trace}(\pi_1) = \varnothing^3 \{ \text{crit}_1 \} \varnothing \{ \text{crit}_2 \} \varnothing^3 \{ \text{crit}_1 \} \ldots \text{ and}$$

$$\text{trace}(\pi_2) = \varnothing^2 (\{ \text{crit}_1 \})^2 \varnothing^2 \{ \text{crit}_2 \} \varnothing \{ \text{crit}_1 \} \ldots$$
Pictorially
Stutter trace equivalence

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

$$TS_1 \equiv TS_2 \text{ if and only if } TS_1 \subseteq TS_2 \text{ and } TS_2 \subseteq TS_1$$

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

$$TS_1 \subseteq TS_2 \text{ iff } \forall \sigma_1 \in \text{Traces}(TS_1) \left( \exists \sigma_2 \in \text{Traces}(TS_2). \sigma_1 \equiv \sigma_2 \right)$$

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

$TS_1 \triangleq TS_2$, $TS_1 \not\preceq TS_3$ and $TS_2 \not\preceq TS_3$, but $TS_3 \preceq TS_2$ and $TS_3 \preceq TS_1$
The $\bigcirc$ operator

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

$\sigma_1 = A B B B \ldots$ and $\sigma_2 = A A A B B B B \ldots$ for $A, B \subseteq AP$ and $A \neq B$

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

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

$\Rightarrow$ a logical characterization of $\trianglelefteq$ can only be obtained by omitting $\bigcirc$ in fact, it turns out that this is the only modal operator that is not preserved by $\trianglelefteq$!
Stutter trace and LTL\(\bigcirc\) equivalence

For traces \(\sigma_1\) and \(\sigma_2\) over \(2^{\text{AP}}\) it holds:

\[ \sigma_1 \triangleq \sigma_2 \Rightarrow (\sigma_1 \models \varphi \text{ if and only if } \sigma_2 \models \varphi) \]

for any LTL\(\bigcirc\) formula \(\varphi\) over \(\text{AP}\)

LTL\(\bigcirc\) denotes the class of LTL formulas without the next operator \(\bigcirc\)
Stutter trace and LTL\(\Box\) equivalence

For transition systems \(TS_1, TS_2\) without terminal states:

(a) \(TS_1 \equiv TS_2\) if and only if \(\left( TS_1 \equiv_{LTL \Box} TS_2 \right)\)

(b) if \(TS_1 \preceq TS_2\) then for any LTL\(\Box\) formula \(\varphi\): \(TS_2 \models \varphi\) implies \(TS_1 \models \varphi\)
Semaphore-based mutual exclusion

This transition system is stutter trace-equivalent:

\[
\begin{align*}
\{ \text{crit}_1 \} & \rightarrow s_1 \\
\emptyset & \rightarrow s_0 \\
& \rightarrow s_2 \{ \text{crit}_2 \}
\end{align*}
\]
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 $s'_1 \in \text{Post}(s_1)$ with $(s'_1, s_2) \not\in \mathcal{R}$, then there exists a finite path fragment $s_2 u_1 \ldots u_n s'_2$ with $n \geq 0$ and $(s_1, u_i) \in \mathcal{R}$ and $(s'_1, s'_2) \in \mathcal{R}$

3. if $s'_2 \in \text{Post}(s_2)$ with $(s_1, s'_2) \not\in \mathcal{R}$, then there exists a finite path fragment $s_1 v_1 \ldots v_n s'_1$ 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

can be completed to

\[
\begin{align*}
\forall \alpha \in \{u_1, u_2, \ldots, u_n\}, \\
\forall s' \in S', \\
\forall s \in S, \\
\forall \eta \in \Delta, \\
s_1 &\approx s_2 \\
\downarrow & \\
s'_1 &\not\approx s_2 \\
\downarrow & \\
s_1 &\approx u_1 \\
\downarrow & \\
s_1 &\approx u_2 \\
\downarrow & \\
\vdots & \\
\downarrow & \\
s_1 &\approx u_n \\
\downarrow & \\
s'_1 &\approx s'_2 \\
\end{align*}
\]
Semaphore-based mutual exclusion

stutter-bisimilar states for $AP = \{ \text{crit}_1, \text{crit}_2 \}$
Stutter-bisimilar transition systems

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, $i = 1, 2$, 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 $TS = (S, \text{Act}, \rightarrow, I, AP, L)$ 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:

\{ crit_1 \} \overset{s_1}{\longrightarrow} \overset{s_0}{\longrightarrow} \{ crit_2 \}
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: $TS_1 \approx TS_2$ does not imply $TS_1 \equiv TS_2$!

- 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 $\equiv$ requires the existence of a stuttering equivalent trace
Stutter trace and stutter bisimulation are incomparable
Stutter bisimulation does not preserve LTL\[\text{∅}\]

\[TS_{left} \approx TS_{right}\] but \[TS_{left} \not\models \diamond a\] and \[TS_{right} \models \diamond a\]

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

<table>
<thead>
<tr>
<th>Formal Relation</th>
<th>Trace Equivalence</th>
<th>Bisimulation</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>PSPACE-complete</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>Logical Fragment</td>
<td>LTL</td>
<td>CTL*</td>
<td>∀CTL*</td>
</tr>
<tr>
<td>Preservation</td>
<td>strong</td>
<td>strong match</td>
<td>weak match</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formal Relation</th>
<th>Stutter Trace Equivalence</th>
<th>Divergence-Sensitive Stutter Bisimulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>PSPACE-complete</td>
<td>PTIME</td>
</tr>
<tr>
<td>Logical Fragment</td>
<td>LTL(\bigcirc)</td>
<td>CTL*(\bigcirc)</td>
</tr>
<tr>
<td>Preservation</td>
<td>strong</td>
<td>strong match</td>
</tr>
</tbody>
</table>
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 $s \ s_1 \ s_2 \ldots \in \text{Paths}(s)$ 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 $s_1 \approx_{TS}^{\text{div}} s_2$, if:

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

$\approx_{TS}^{\text{div}}$ 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:

$$\frac{s \xrightarrow{\alpha} s' \land s \not\approx^{\text{div}} s'}{[s]_{\text{div}} \xrightarrow{\tau} [s']_{\text{div}}}$$  
and

$$s \text{ is } \approx^{\text{div}} - \text{divergent} \quad \frac{[s]_{\text{div}} \xrightarrow{\tau} [s]_{\text{div}}}{[s]_{\text{div}} \xrightarrow{\tau} [s]_{\text{div}}}$$

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

Transition system \(TS\)

\[ [s_3] \approx [s_0] \approx \emptyset \]

\[ [s_3] \text{div} [s_2] \text{div} [s_0] \text{div} \]

Transition system \(TS/\approx\)

Transition system \(TS/\approx^{\text{div}}\)
Summary

stutter trace inclusion:
\[ TS_1 \trianglelefteq TS_2 \text{ iff } \forall \sigma_1 \in \text{Traces}(TS_1) \exists \sigma_2 \in \text{Traces}(TS_2). \sigma_1 \trianglelefteq \sigma_2 \]

stutter trace equivalence:
\[ TS_1 \triangleq TS_2 \text{ iff } TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1 \]

stutter bisimulation equivalence:
\[ TS_1 \approx TS_2 \text{ iff } \text{there exists a stutter bisimulation for } (TS_1, TS_2) \]

stutter bisimulation equivalence with divergence:
\[ TS_1 \approx^{\text{div}} TS_2 \text{ iff } \text{there exists a divergence-sensitive stutter bisimulation for } (TS_1, TS_2) \]
CTL$^*$ and CTL\$ equivalence vs $\sim^{div}$

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

$$s_1 \sim^{div}_{TS} s_2 \iff s_1 \equiv_{CTL^*} s_2 \iff s_1 \equiv_{CTL\$} s_2$$
CTL\(\_\)equivalence vs \(\approx_{\text{div}}\)

For finite transition system \(TS\) without terminal states, and \(s_1, s_2\) in \(TS\):

\[ s_1 \approx_{TS}^d s_2 \quad \text{iff} \quad s_1 \equiv_{\text{CTL}\_\_} s_2 \quad \text{iff} \quad s_1 \equiv_{\text{CTL}\_\_} s_2 \quad \text{iff} \quad s_1 \equiv_{\text{CTL}\_\_,U} s_2 \]
Equivalences and logical equivalence

\[
\begin{align*}
\text{CTL}^* \text{ equivalence} & \quad \text{LTL equivalence} \\
\text{bisimulation equivalence} & \quad \text{trace equivalence} & \quad \text{trace inclusion} \\
TS_1 \sim TS_2 & \quad \text{Traces}(T_1) = \text{Traces}(TS_2) & \quad \text{Traces}(T_1) \subseteq \text{Traces}(TS_2) \\
\text{divergence sensitive} & \quad \text{stutter trace-equivalence} & \quad \text{stutter trace inclusion} \\
\text{stutter bisimulation equivalence} & \quad TS_1 \triangleq TS_2 & \quad TS_1 \sqsubseteq TS_2 \\
\text{CTL}^* \text{ equivalence} & \quad \text{LTL} \text{ equivalence}
\end{align*}
\]
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 \( (TS_1 \models \Phi \iff TS_2 \models \Phi) \)
  - for any CTL\(^*\) (or CTL\(\setminus\)) formula \( \Phi \)

\( \therefore \) To check \( TS \models \Phi \), if 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 $TS/\approx^{div}$:
   - *transform* $TS$ into a (divergence-sensitive) transition system $\overline{TS}$
   - $\overline{TS}$ is divergent-sensitive, i.e., $\approx_{\overline{TS}}$ and $\approx^{div}_{\overline{TS}}$ coincide
   - determine $\overline{TS}/\approx$ using the quotienting algorithm for $\approx$
   - “distill” $TS/\approx^{div}$ 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: $\mathcal{R}_{\Pi_0} \supseteq \mathcal{R}_{\Pi_1} \supseteq \mathcal{R}_{\Pi_2} \supseteq \ldots \supseteq \mathcal{R}_{\Pi_i} = \approx_{TS}$
  - time complexity: maximally $|S|$ iterations needed
Theorem

\[ S/ \approx \text{ is the coarsest partition } \Pi \text{ of } S \text{ such that:} \]

(i) \( \Pi \) is finer than the initial partition \( \Pi_{AP} \), and

(ii) \( B \cap Pre^*_\Pi(C) = \emptyset \) or \( B \subseteq Pre^*_\Pi(C) \) for all \( B, C \in \Pi \)

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

\[ s \in Pre^*_\Pi(C) \text{ whenever } s = s_1 s_2 \ldots s_{n-1} s_n \in Paths(s) \]

\( \text{state } s \text{ can reach } C \text{ via a path that is completely in } B (= [s]_{\Pi}) \)
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 \( S/\approx \):
    \[
    \text{Refine}_{\approx}(\Pi, C) \text{ is finer than } \Pi \quad \text{and} \quad \text{Refine}_{\approx}(\Pi, C) \text{ is coarser than } S/\approx
    \]
  - \( \Pi \) is strictly coarser than \( S/\approx \) if and only if there exists a \textit{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 $S/\approx$

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

od
return $\Pi$

(* as before *)
Stutter cycles

• \( s_0 \ s_1 \ldots s_n \) is a stutter cycle if \( s_i \ s_{i+1} \) is a stutter step. One stutter cycle is

\[
\begin{align*}
    s_0 & \approx_{TS} s_1 \\
    s_1 & \approx_{TS} s_2 \\
    & \ldots \\
    s_{n-1} & \approx_{TS} s_n \\
    s_n & = s_0
\end{align*}
\]

• 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 \( B \):

\[
\text{Bottom}(B) = \left\{ s \in B \mid \text{Post}(s) \cap B = \emptyset \right\}
\]

• 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 $O\left(|S| \cdot (|AP| + M)\right)$.
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 $\sim$)

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

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

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

where

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

$s_{\text{div}} \not\approx s$ for any $s \in S$ and $s_{\text{div}}$ 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 $s_1, s_2 \in S$: $s_1 \approx_{\text{div}}^{TS} s_2$ 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** \( S/\approx^{\text{div}}_{TS} = S/\approx_{\overline{TS}} \)

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 \( s \not\approx^{\text{div}}_{TS} s' \) 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 $TS/\approx^{\mathsf{div}}$ can be determined with a worst-case time complexity in $O\left(|S| \cdot (|AP|+M)\right)$.
### Summary

<table>
<thead>
<tr>
<th>formal relation</th>
<th>trace equivalence</th>
<th>bisimulation</th>
<th>simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>complexity</td>
<td>PSPACE-complete</td>
<td>$O(M \cdot \log</td>
<td>S</td>
</tr>
<tr>
<td>logical fragment</td>
<td>LTL</td>
<td>CTL*</td>
<td>$\forall$CTL*</td>
</tr>
<tr>
<td>preservation</td>
<td>strong</td>
<td>strong match</td>
<td>weak match</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>formal relation</th>
<th>stutter trace equivalence</th>
<th>divergence-sensitive stutter bisimulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>complexity</td>
<td>PSPACE-complete</td>
<td>$O(M \cdot</td>
</tr>
<tr>
<td>logical fragment</td>
<td>LTL (\bigcirc)</td>
<td>CTL* (\bigcirc)</td>
</tr>
<tr>
<td>preservation</td>
<td>strong</td>
<td>strong match</td>
</tr>
</tbody>
</table>