Model Checking
Lecture #15: Bisimulation Quotienting
[Baier & Katoen, Chapter 7.2–7.6]

Joost-Pieter Katoen
Software Modeling and Verification Group
Model Checking Course, RWTH Aachen, WiSe 2019/2020

Overview

1. Bisimulation Equivalence
2. Quotient Transition System
3. Bisimulation Quotienting
4. Simulation Pre-Order
5. Checking Simulation Pre-order

State Spaces Can Be Gigantic

State Spaces Can Be Gigantic

Treating Gigantic Models?

- Use compact data structures
- Make models smaller prior to (or: during) model checking
- Try to make them even smaller
- If possible, try to obtain the smallest possible model
- While preserving the properties of interest
- Do this all algorithmically and possibly fast
Abstraction

Reduce a huge TS to a small $\overline{TS}$ prior or during model checking.

Relevant issues:

- What is the formal relationship between TS and $\overline{TS}$?
- Can $\overline{TS}$ be obtained algorithmically and efficiently?
- Which logical fragment (of LTL, CTL, CTL*) is preserved?
- And in what sense?
  - "strong" preservation: positive and negative results carry over
  - "weak" preservation: only positive results carry over
  - "match": logic equivalence coincides with formal relation

Bisimulation Equivalence

Definition: bisimulation relation

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP, L_i)$, $i=1,2$, be transition systems. The symmetric relation $R \subseteq (S_1 \times S_2 \cup S_2 \times S_1)$ is a bisimulation for $(TS_1, TS_2)$ whenever:

1. for all initial states $s_1 \in I_1$, $(s_1, s_2) \in R$ for some $s_2 \in I_2$
2. for all states $(s_1, s_2) \in R$ it holds:
   2.1 $L_1(s_1) = L_2(s_2)$, and
   2.2 $s'_1 \in Pos(s_1)$ implies $(s'_1, s'_2) \in R$ for some $s'_2 \in Pos(s_2)$.  

Overview

- Bisimulation Equivalence
- Quotient Transition System
- Bisimulation Quotienting
- Simulation Pre-Order
- Checking Simulation Pre-order
Bisimulation Equivalence

Visually

\[ s_1 \rightarrow s'_1 \]
\[ R \quad \text{can be completed to} \quad R \]
\[ s_2 \rightarrow s'_2 \]

and by symmetry

\[ s_1 \rightarrow s'_1 \]
\[ R \quad \text{can be completed to} \quad R \]
\[ s_2 \rightarrow s'_2 \]

Definition: bisimulation equivalence

\[ TS_1 \text{ and } TS_2 \text{ are bisimulation equivalent (short: bisimilar), denoted } \sim \text{ if there exists a bisimulation for } (TS_1, TS_2). \text{ That is:} \]

\[ \sim = \bigcup \{ R \mid R \text{ is a bisimulation on } (TS_1, TS_2) \}. \]

Bisimilarity (\( \sim \)) is an equivalence relation.

Overview

1. Bisimulation Equivalence
2. Quotient Transition System
3. Bisimulation Quotienting
4. Simulation Pre-Order
5. Checking Simulation Pre-order

Quotient Transition System

Bisimulation on States

Definition: bisimulation/bisimilarity on states

Symmetric relation \( R \subseteq S \times S \) is a bisimulation on \( TS \) (with state space \( S \)) if for any \( (s_1, s_2) \in R \):

1. \( L(s_1) = L(s_2) \)
2. \( s'_1 \in \text{Post}(s_1) \text{ then } (s'_1, s'_2) \in R \text{ for some } s'_2 \in \text{Post}(s_2). \)

The states \( s_1 \) and \( s_2 \) are bisimilar, denoted \( s_1 \sim_{TS} s_2 \), if \( (s_1, s_2) \in R \) for some bisimulation \( R \) for \( TS \).

\[ s_1 \sim_{TS} s_2 \text{ if and only if } TS_{s_1} \sim TS_{s_2} \text{ where } TS_{s_i} \text{ denotes the transition system } TS \text{ in which } s_i \text{ is the only initial state.} \]
### Coarsest Bisimulation

The relation $\sim_{TS}$ is a bisimulation, an equivalence, and the coarsest bisimulation for $TS$.

**Proof.**

### Property

For every transition system $TS$ it holds: $TS \sim TS/\sim_{TS}$.

**Proof.**

### Coarsest Bisimulation

The relation $\sim_{TS}$ is a bisimulation, an equivalence, and the coarsest bisimulation for $TS$.

**Proof.**

### Quotient Transition System

**Definition: quotient transition system**

For $TS = (S, Act, \rightarrow, I, AP, L)$ and bisimulation $\sim_{TS} \subseteq S \times S$ on $TS$, let the quotient transition system

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

the quotient of $TS$ under $\sim_{TS}$

where

- $S' = S/\sim_{TS} = \{[s]_\sim \mid s \in S\}$ with $[s]_\sim = \{s' \in S \mid s \sim_{TS} s'\}$
- $\rightarrow'$ is defined by:
  $$s \xrightarrow{\alpha} s' \iff [s]_\sim \xrightarrow{\alpha} [s']_\sim$$
- $I' = \{[s]_\sim \mid s \in I\}$
- $L'([s]_\sim) = L(s)$
(Simplified) Lamport’s Bakery Algorithm

Thread 1:

```
while true {
  ....
  n1 : x1 := x2 + 1;
  w1 : wait until x1 = 0 || x1 < x2 {
    c1 : .... critical section ....
    x1 := 0;
  }
  ....
}
```

Thread 2:

```
while true {
  ....
  n2 : x2 := x1 + 1;
  w2 : wait until x1 = 0 || x2 < x1 {
    c2 : .... critical section ....
    x2 := 0;
  }
  ....
}
```

This algorithm can be applied to arbitrarily many processes.

Example Bakery Algorithm Run

<table>
<thead>
<tr>
<th>thread P_1</th>
<th>thread P_2</th>
<th>x_1</th>
<th>x_2</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_1</td>
<td>n_2</td>
<td>0</td>
<td>0</td>
<td>P_1 requests access to critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>w_2</td>
<td>1</td>
<td>0</td>
<td>P_2 requests access to critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>w_2</td>
<td>1</td>
<td>2</td>
<td>P_1 enters the critical section</td>
</tr>
<tr>
<td>c_1</td>
<td>w_2</td>
<td>1</td>
<td>2</td>
<td>P_1 leaves the critical section</td>
</tr>
<tr>
<td>n_1</td>
<td>w_2</td>
<td>0</td>
<td>2</td>
<td>P_1 requests access to critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>w_2</td>
<td>3</td>
<td>2</td>
<td>P_2 enters the critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>c_2</td>
<td>3</td>
<td>2</td>
<td>P_2 leaves the critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>n_2</td>
<td>3</td>
<td>0</td>
<td>P_2 requests access to critical section</td>
</tr>
<tr>
<td>w_1</td>
<td>w_2</td>
<td>3</td>
<td>4</td>
<td>P_2 enters the critical section</td>
</tr>
</tbody>
</table>

Counters may grow unboundedly large.

Bakery Algorithm Transition System

Infinite state space due to possible unbounded increase of counters.

Bisimulation Relation

Let function $f$ map a reachable state of $TS_{Bak}$ onto a state in $TS_{Bak}^{abs}$

Let $s = \langle \ell_1, \ell_2, x_1 = b_1, x_2 = b_2 \rangle \in TS_{Bak}$ with $\ell_i \in \{ n_i, w_i, c_i \}$ and $b_i \in \mathbb{N}$

Then:

$$f(s) = \begin{cases} 
\langle \ell_1, \ell_2, x_1 = 0, x_2 = 0 \rangle & \text{if } b_1 = b_2 = 0 \\
\langle \ell_1, \ell_2, x_1 = 0, x_2 > 0 \rangle & \text{if } b_1 = 0 \text{ and } b_2 > 0 \\
\langle \ell_1, \ell_2, x_1 > 0, x_2 = 0 \rangle & \text{if } b_1 > 0 \text{ and } b_2 = 0 \\
\langle \ell_1, \ell_2, x_1 > 0, x_2 > 0 \rangle & \text{if } b_1 > b_2 > 0 \\
\langle \ell_1, \ell_2, x_1 > 0, x_2 > 0 \rangle & \text{if } b_2 > b_1 > 0 
\end{cases}$$

It follows: $R = \{ (s, f(s)) \mid s \in S \}$ is a bisimulation for $(TS_{Bak}, TS_{Bak}^{abs})$ for any subset of $AP = \{ \text{noncrit}_i, \text{wait}_i, \text{crit}_i \mid i = 1, 2 \}$. 
Quotient Transition System

Quotient of Bakery Algorithm

\[ TS_{\text{Bak}}^{\text{abs}} = TS_{\text{Bak}} / \sim \text{ for } AP = \{ \text{noncrit}, \text{wait}i, \text{crit}i \mid i = 1, 2 \} \]

Overview

1. Bisimulation Equivalence
2. Quotient Transition System
3. Bisimulation Quotienting
4. Simulation Pre-Order
5. Checking Simulation Pre-order

Partitions

- A partition \( \Pi = \{ B_1, \ldots, B_k \} \) of \( S \) satisfies:
  - \( B_i \) is non-empty; \( B_i \) is called a block
  - \( B_i \cap B_j = \emptyset \) for all \( i, j \) with \( i \neq j \)
  - \( B_1 \cup \ldots \cup B_k = S \)

- \( C \subseteq S \) is a super-block of partition \( \Pi \) of \( S \) if
  \[ C = B_{i_1} \cup \ldots \cup B_{i_m} \text{ for } B_{i_j} \in \Pi \text{ for } 0 < j \leq m \]

- Partition \( \Pi \) (of \( S \)) is finer than partition \( \Pi' \) (of \( S \)) if:
  \[ \forall B \in \Pi. \ (\exists B' \in \Pi'. B \subseteq B') \]

- each block of \( \Pi' \) equals the union of a set of blocks in \( \Pi \)

- \( \Pi \) is strictly finer than \( \Pi' \) if it is finer than \( \Pi' \) and \( \Pi \neq \Pi' \)

Partitions and Equivalences

- \( \mathcal{R} \) is an equivalence on \( S \) \( \Rightarrow \) \( S/\mathcal{R} \) is a partition of \( S \)

- Partition \( \Pi = \{ B_1, \ldots, B_k \} \) of \( S \) induces the equivalence relation
  \[ \mathcal{R}_\Pi = \{ (s, t) \mid \exists B_i \in \Pi. s \in B_i \land t \in B_i \} \]
  where it holds: \( S/\mathcal{R}_\Pi = \Pi \).

There is a one-to-one relationship between partitions and equivalences.
Partition Refinement

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

- Iteratively compute a partition of $S$
- Initially: $\Pi_0$ equals $\Pi_{AP} = \{ (s, t) \in S \times S \mid L(s) = L(t) \}$
- Repeat until no change: $\Pi_{i+1} \coloneqq \text{Refine}(\Pi_i)$
  - loop invariant: $\Pi_i$ is coarser than $S/\sim$ and finer than $\{ S \}$
- Return $\Pi_i$
  - termination is ensured:
    $$S \times S \supseteq R_{\Pi_0} \supseteq R_{\Pi_1} \supseteq R_{\Pi_2} \supseteq \ldots \supseteq R_{\Pi_i} = \sim_{TS}$$
  - time complexity: maximally $|S|$ iterations needed

Refinement Operator

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

  $\text{Refine}(B, C) = \{ B \cap \text{Pre}(C), \; B \setminus \text{Pre}(C) \} \setminus \{ \emptyset \}$

- Basic properties:
  - for $\Pi$ finer than $\Pi_{AP}$ and coarser than $S/\sim$:
    $\text{Refine}(\Pi, C)$ is finer than $\Pi$ and $\text{Refine}(\Pi, C)$ is coarser than $S/\sim$
  - $\Pi$ is strictly coarser than $S/\sim$ if and only if there exists a splitter for $\Pi$

Theorem

$S/\sim$ is the coarsest partition $\Pi$ of $S$ such that:
1. $\Pi$ is finer than the initial partition $\Pi_{AP}$, and
2. for all $B, C \in \Pi$ it holds\footnote{In fact, this also holds for all $B \in \Pi$ and all super-blocks $C$ of $\Pi$.}:
   $$B \cap \text{Pre}(C) = \emptyset \text{ or } B \subseteq \text{Pre}(C).$$

Proof. \square

Splitters

- Let $\Pi$ be a partition of $S$ and $C$ a super-block of $\Pi$
- $C$ is a splitter of $\Pi$ if for some $B \in \Pi$:
  $$B \cap \text{Pre}(C) \neq \emptyset \quad \text{and} \quad B \setminus \text{Pre}(C) \neq \emptyset$$
- Block $B$ is stable wrt. $C$ if
  $$B \cap \text{Pre}(C) = \emptyset \quad \text{and} \quad B \setminus \text{Pre}(C) = \emptyset$$
- $\Pi$ is stable w.r.t. $C$ if every $B \in \Pi$ is stable wrt. $C$
Algorithm Skeleton

Input: finite transition system $TS$ over $AP$ with state space $S$
Output: bisimulation quotient space $S/\sim$

$\Pi := \Pi_{AP}$
\[\textbf{while} \text{ there exists a splitter for } \Pi \text{ do} \]
\[\text{choose a splitter } C' \text{ for } \Pi; \]
$\Pi := \text{Refine}(\Pi, C')$;  (* Refine$(\Pi, C')$ is strictly finer than $\Pi$ *)
\[\textbf{od} \]
return $\Pi$

Which Splitter to Take?

How to determine a splitter for partition $\Pi_{i+1}$?

1. Simple strategy:
   $O(|S| \cdot M)$
   use any block of $\Pi_i$ as splitter candidate

2. Advanced strategy:
   $O(\log |S| \cdot M)$
   use only "smaller" blocks of $\Pi_i$ as splitter candidates and apply "a ternary" refinement

Advanced Selection Strategy

- Not necessary to refine with respect to all blocks $C \in \Pi_{old}$

  ⇒ Consider only the "smaller" subblocks of a previous refinement

  - Step $i$: refine $C'$ into $C_1 = C' \cap \text{Pre}(D)$ and $C_2 = C' \setminus \text{Pre}(D)$

  - Step $i+1$: use the smallest $C \in \{ C_1, C_2 \}$ as splitter
    - let $C$ be such that $|C| \leq |C'|/2$, thus $|C| \leq |C' \setminus C|$
    - combine the refinement steps with respect to $C$ and $C' \setminus C$
    - $\text{Refine}(\Pi, C, C' \setminus C) = \text{Refine}(\text{Refine}(\Pi, C), C' \setminus C)$ where $|C| \leq |C' \setminus C|$

    the decomposed blocks are stable with respect to $C$ and $C' \setminus C$
The Ternary Refinement Operator

Let: \( \text{Refine}(\Pi, C, C' \setminus C) = \bigcup_{B \in \Pi} \text{Refine}(B, C, C' \setminus C) \)

where \( \text{Refine}(B, C, C' \setminus C) = \{ B_1, B_2, B_3 \} \setminus \{ \emptyset \} \) with:

- \( B_1 = B \cap \text{Pre}(C) \cap \text{Pre}(C' \setminus C) \) to both \( C \) and \( C \setminus C' \)
- \( B_2 = (B \cap \text{Pre}(C)) \setminus \text{Pre}(C' \setminus C) \) only to \( C \)
- \( B_3 = (B \cap \text{Pre}(C' \setminus C)) \setminus \text{Pre}(C) \) only to \( C' \setminus C \)

\( \Rightarrow \) blocks \( B_1, B_2, B_3 \) are stable with respect to \( C \) and \( C' \setminus C \)

\[ \begin{array}{c}
\text{block } B \\
\end{array} \]

Complexity

The bisimulation quotient of finite transition system \( TS \) can be computed in \( O(N \cdot \log M) \) where \( N \) and \( M \) are the number of states and transitions in \( TS \) respectively.

Checking bisimilarity is PTIME-complete.

Proof.
Reduction from the direct circuit value problem. Outside the scope of this lecture.
Simulation Relation

**Definition: simulation relation**
Relation $R \subseteq S \times S$ is a simulation relation on $TS$ if for any $(s_1, s_2) \in R$:
- $L(s_1) = L(s_2)$, and
- if $s_1' \in \text{Post}(s_1)$ then $(s_1', s_2') \in R$ for some $s_2' \in \text{Post}(s_2)$.

State $s_2$ simulates $s_1$, written $s_1 \preceq_{TS} s_2$ if $(s_1, s_2) \in R$ for some simulation relation $R$ on $TS$.

$TS_1 \preceq \ TS_2$ iff $\forall s_1 \in I_1, \exists s_2 \in I_2. s_1 \preceq_{TS_1} s_2$.

$\preceq_{TS}$ is a preorder and the coarsest simulation for $TS$.

Abstraction Function

**Definition: abstraction function**
$f : S \rightarrow \hat{S}$ is an abstraction function if $f(s) = f(s') \Rightarrow L(s) = L(s')$.

$S$ are “concrete” states and $\hat{S}$ are “abstract” states, mostly $|\hat{S}| < |S|$.

Abstraction functions are useful for:
- data abstraction: abstract from values of program or control variables
- $f : \text{concrete data domain} \rightarrow \text{abstract data domain}$
- predicate abstraction: use predicates over the program variables
- $f : \text{state} \rightarrow \text{valuations of the predicates}$
- localization reduction: program variables are visible or invisible
- $f : \text{all variables} \rightarrow \text{visible variables}$

Abstract Transition System

**Definition: abstract transition system**
For $TS = (S, \text{Act, } \rightarrow, I, AP, L)$ and abstraction function $f : S \rightarrow \hat{S}$ let:

$$TS_f = (\hat{S}, \text{Act, } \rightarrow_f, I_f, AP, L_f),$$

the abstraction of $TS$ under $f$ where

- $\rightarrow_f$ is defined by: $s \xrightarrow{a} s'$ if $f(s) \xrightarrow{a} f(s')$.
- $I_f = \{ f(s) \mid s \in I \}$ and $L_f(f(s)) = L(s)$.

The relation $R = \{(s, f(s)) \mid s \in S \}$ is a simulation for $(TS, TS_f)$.

**Proof.**
By checking all conditions of a simulation relation. Straightforward. 

Visually

$\begin{align*}
\text{s}_1 & \rightarrow \text{s}_1' \\
R & \text{can be completed to} \ R & R \\
\text{s}_2 & \rightarrow \text{s}_2' \\
\text{but not necessarily:} & \\
\text{s}_1 & \rightarrow \text{s}_1' \\
\text{R} & \text{can be completed to} \ R & R \\
\text{s}_2 & \rightarrow \text{s}_2' 
\end{align*}$
**Simulation Pre-Order**

**Simulation Equivalence**

**Definition: simulation equivalence**

Transition systems \( TS_1 \) and \( TS_2 \) are **simulation equivalent**, denoted \( TS_1 \sim TS_2 \) if \( TS_1 \preceq TS_2 \) and \( TS_2 \preceq TS_1 \).

1. Bisimilarity implies simulation equivalence; not the converse.
2. Simulation equivalence implies trace equivalence; not the converse.
3. For \( AP \)-deterministic\(^2 \) transition systems, simulation, bisimulation and trace equivalence coincide.

\(^2\) \( TS \) is \( AP \)-deterministic if all initial states are labelled differently, and this also applies to all direct successors of any state in \( TS \).

**Logical Characterisation**

- Negation of formulas is problematic as \( \preceq_{TS} \) is not symmetric.
- Let \( L \) be a fragment of CTL\(^*\) which is closed under negation.
- And assume \( L \) weakly matches \( \preceq_{TS} \), that is:
  
  \[ s_1 \preceq_{TS} s_2 \iff \text{ for all state formulae } \Phi \text{ of } L: \ s_2 \models \Phi \implies s_1 \not\models \Phi. \]

- Let \( s_1 \preceq_{TS} s_2 \). Then, for any state formula \( \Phi \) of \( L \):
  
  \[ s_1 \models \Phi \implies s_2 \not\models \Phi \implies s_2 \models \Phi. \]

- Hence, \( s_2 \preceq_{TS} s_1 \) which requires \( \preceq_{TS} \) to be symmetric. Contradiction.
Universal Fragment of CTL*

Definition: universal fragment of CTL*

\( \forall \text{CTL}^* \) state-formulas are formed according to:

\[ \Phi ::= \text{true} \mid \text{false} \mid a \mid \neg a \mid \Phi_1 \land \Phi_2 \mid \Phi_1 \lor \Phi_2 \mid \forall \varphi \]

where \( a \in \text{AP} \) and \( \varphi \) is a path-formula. \( \forall \text{CTL}^* \) path-formulas are formed according to:

\[ \varphi ::= \Phi \mid \Box \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 U \varphi_2 \mid \varphi_1 R \varphi_2 \]

where \( \Phi \) is a state-formula, and \( \varphi, \varphi_1 \) and \( \varphi_2 \) are path-formulas.

\( \forall \text{CTL} \) does not contain (general) negation and no existential path quantifier.

Simulation and CTL

Theorem: Simulation equivalence, CTL and and CTL*

Let \( TS \) be a finitely branching transition system and \( s, s' \) states in \( TS \). The following statements are equivalent:

1. \( s \preceq_{TS} s' \)
2. for any \( \forall \text{CTL}^* \)-formula \( \Phi \): \( s' \vDash \Phi \) implies \( s \vDash \Phi \)
3. for any \( \forall \text{CTL} \)-formula \( \Phi \): \( s' \vDash \Phi \) implies \( s \vDash \Phi \)
4. for any \( \forall \text{CTL} \setminus U, R \)-formula \( \Phi \): \( s' \vDash \Phi \) implies \( s \vDash \Phi \)

Proof.

Along similar lines as the proof for the corresponding theorem for bisimilarity and CTL*, CTL and CTL -equivalence.
Algorithm Skeleton

Input: finite transition system $TS$ over $AP$ with state space $S$
Output: simulation order $\preceq_{TS}$

$\mathcal{R} := \{(s_1, s_2) \mid L(s_1) = L(s_2)\}$

while $\mathcal{R}$ is not a simulation do
  let $(s_1, s_2) \in \mathcal{R}$ such that $s_1 \rightarrow s_1'$ and $\forall s_2', s_2 \rightarrow s_2'$ implies $(s_1', s_2') \notin \mathcal{R}$;
  $\mathcal{R} := \mathcal{R} \setminus \{(s_1, s_2)\}$;
end

return $\mathcal{R}$

The number of iterations is bounded above by $|S|^2$, since:

$S \times S \supseteq \mathcal{R}_0 \supseteq \mathcal{R}_1 \supseteq \mathcal{R}_2 \supseteq \ldots \supseteq \mathcal{R}_n = \preceq_{TS}$

Time complexity

The time complexity of computing $\preceq_{TS}$ is $O(M \cdot N^2)$.

Proof.

In the worst case, there are $N^2$ iterations as there are $N^2$ pairs of states. For each pair of states in the worst case all transitions have to be examined.

The best known algorithm\(^4\) has complexity $O(M \cdot N)$. It removes several pairs in each iteration at a time and uses efficient data structures for the sets $Sim_{\mathcal{R}_i}(s)$.

\(^4\)Due to Henzinger, Henzinger and Kopke.

Algorithm

for all $s_1 \in S$ do
  $Sim(s_1) := \{ s_2 \in S \mid L(s_1) = L(s_2) \}$;
end

while $\exists (s_1, s_2) \in S \times Sim(s_1), \exists s'_1 \in Post(s_1)$ with $Post(s_2) \cap Sim(s'_1) = \emptyset$ do
  choose such a pair of states $(s_1, s_2)$;
  $Sim(s_1) := Sim(s_1) \setminus \{s_2\}$;
end

return $\{(s_1, s_2) \mid s_2 \in Sim(s_1)\}$

Next Lecture

Thursday December 19, 10:30