

# Implementation of an Efficient Algorithm for Checking Bisimulation Equivalence †

Vinod Viswanath

Department of Electrical & Computer Engineering  
University of Texas at Austin  
vvinod@ece.utexas.edu

December 2, 2001

†Project Presentation for *EE382C Software Reliability* Fall 2001.

# Outline

- Bisimulations and Bisimulation Equivalence [Pel01]
- Relational Coarsest Partition Refinement Problem [PT87][KS80][Pel01]
- The Adapted Paige-Tarjan algorithm for calculating Bisimulation Equivalence [PT87]
- Other Algorithms[KS80][CC83][LY91]

# Notations

A labeled transition system is a quadruple  $S = \langle Q, A, T, q_0 \rangle$ , such that,

- $Q$  is a set of states,
- $A$  is a finite set of actions,
- $T \subseteq Q \times A \times Q$  is the transition relation ,
- $q_0$  is the initial state.

## Notations (continued ...)

For each  $\alpha \in A$ , the transition relation  $T_\alpha$  is considered to be either

- a binary relation on  $Q$  :  $T_\alpha = \{(p, q) \mid (p, \alpha, q) \in T\}$ , or
- a function  $Q \longrightarrow 2^Q$  :  $T_\alpha [p] = \{q \mid (p, \alpha, q) \in T\}$ .

We also use the following definitions/notations:

- $p \xrightarrow{\alpha} q$  for  $(p, \alpha, q) \in T$
- $T_\alpha^{-1} [q] = \{p \mid (p, \alpha, q) \in T\}$

## Notations (continued ...)

- $T_\alpha^{-1} [B] = \cup\{T_\alpha^{-1} [q] \mid q \in B\}$ , for  $B \subseteq Q$
- $|X|$  denotes the number of elements of the set  $X$
- $T$  is image-finite if  $\forall \alpha \in A. \forall q \in Q. T_\alpha^{-1} [q]$  is finite
- $n = |Q|$ , the cardinality of the set of states
- $m = |T|$ , the cardinality of the set of transition relations
- $c$  denotes the supremum for  $\alpha \in A$  and  $q \in Q$  of the image set sizes  $|T_\alpha [q]|$

# Bisimulation

**Definition 1** Given a labeled transition system  $S = (Q, A, T, q_0)$ , a binary relation  $\rho \subseteq Q \times Q$  is a *bisimulation* if and only if :

$$\forall (p_1, p_2) \in \rho . \forall \alpha \in A .$$

$$\forall r_1 . (p_1 \xrightarrow{\alpha} r_1 \Rightarrow \exists r_2 . (p_2 \xrightarrow{\alpha} r_2 \wedge (r_1, r_2) \in \rho)) \wedge$$

$$\forall r_2 . (p_2 \xrightarrow{\alpha} r_2 \Rightarrow \exists r_1 . (p_1 \xrightarrow{\alpha} r_1 \wedge (r_1, r_2) \in \rho))$$

Intuitively, two states  $p$  and  $q$  are bisimilar

if for each state  $p'$  reachable from  $p$  by execution of an action  $\alpha$  there is a state  $q'$ , reachable from  $q$  by execution of the same action  $\alpha$  such that  $p'$  and  $q'$  are bisimilar and vice versa.

## Bisimulation Equivalence ( $\sim_{bis}$ )

If  $\rho$  is a *bisimulation* then

- $\forall p \in Q, (p, p) \in \rho$  [reflexive],
- if  $(p_1, p_2) \in \rho$ , then  $(p_2, p_1) \in \rho$  [symmetric], and
- if  $(p_1, p_2) \in \rho$  and  $(p_2, p_3) \in \rho$ , then  $(p_1, p_3) \in \rho$  [transitive]

Therefore, *bisimulation* is an *equivalence relation* ( $\sim_{bis}$ ).

$\sim_{bis}$  can be used as *correctness criterion* for comparing two models  
(e.g. if an implementation and a design are bisimilar equivalent then correctness of the implementation is proved.)

## Calculating Bisimulation Equivalence [Pel01]

- Consider two states  $E$  and  $F$ .
- Construct the labeled transition system  $\langle Q, A, T, q_0 \rangle$ , where
  - $Q$  is the finite set of states that can evolve from either  $E$  or  $F$  and  $E, F \in Q$
  - $T$  is the transition relation between states,  $T \subseteq Q \times A \times Q$
- The algorithm repeatedly partitions  $Q$  into disjoint subsets.

## Calculating Bisimulation Equivalence ( $\rho$ )

- Start with one subset (e.g. all nodes in  $Q$ ) and refine the partition, *i.e.* break down subset of nodes into smaller subsets until each subset contains agents that are  $\sim_{bis}$  to each other  
 $\Rightarrow$  In each subset  $B \forall D_1, D_2 \in B, (D_1, D_2) \in \rho$ .
- It is possible to satisfy bisimulation equivalence on  $\langle Q, T, A, q_0 \rangle$  by more than one relation  $\rho$ .
- The maximal relation  $\rho$  includes the largest such bisimulation equivalence classes partitioning  $Q$ .
- To check for bisimulation equivalence between  $E$  and  $F$ , we need to find a maximal relation  $\rho$  and then check if  $E$  and  $F$  are in the same subset of the partition or not.

## Calculating Bisimulation Equivalence ( $\Psi(\rho)$ )

- This maximal relation  $\Psi(\rho)$  is the maximum element of the set of all bisimulations on  $Q$ , ordered by inclusion.
- $\Psi$  is a monotone operator under inclusion.
- $\rho \subseteq \Psi(\rho) \subseteq \Psi^2(\rho) \subseteq \Psi^3(\rho) \dots \subseteq \Psi^n(\rho) \dots \subseteq \rho_\Psi$ .
- By the *Tarski-Knaster* theorem  $\Psi(\rho)$  is the greatest fixed point  $\nu z. \Psi(z)$ .

## Calculating Bisimulation Equivalence ( $\Psi(\rho)$ )

- If  $T$  is image-finite then  $\Psi$  is  $\cap$ -continuous [Lar86]  
(i.e.  $\Psi(\bigcap_{i \in I} \rho_i) = \bigcap_{i \in I} \Psi(\rho_i)$  for each decreasing sequence  $\{\rho_i \mid i \in I\}$ )
- $\nu z. \Psi(z)$  is given by the limit of the sequence  $(\rho_r)_{r \in \mathbb{N}}$  such that  $\rho_0 = Q \times Q$  and  $\rho_{r+1} = \Psi(\rho_r)$

$$\begin{aligned} \rho_\Psi &= \lim_{n \rightarrow \infty} \rho_n = \lim_{n \rightarrow \infty} \bigcap_{i=0}^n \Psi^i(\rho_{n-1}) \dots = \lim_{n \rightarrow \infty} \bigcap_{i=0}^n \Psi^i(\rho_0) \\ &\implies \rho_\Psi = \bigcap_{i=0}^{\infty} \Psi^i(Q \times Q) \end{aligned}$$

## Calculating Bisimulation Equivalence ( $\Psi(\rho)$ )

- Therefore, the problem of finding the maximum Bisimulation Equivalence relation is reduced to the problem of finding  $\Psi(\rho)$  for a given labeled transition system  $\langle Q, A, T, q_0 \rangle$
- Finding a partition of  $Q$  with the relation  $\Psi(\rho)$  is defined as the *relational coarsest partition refinement* problem.

# Partition Refinement

Consider a labeled transition system  $S = \langle Q, A, T, q_0 \rangle$ .

- an equivalence relation  $\rho$  on  $Q$  is a partition  $\rho = \{B_1 \dots B_n\}$  where each  $B_i$  represents one of its equivalence classes  
(i.e.  $\forall x, y \in Q . (x, y) \in \rho$  if and only if  $\exists B_i . (x \in B_i \wedge y \in B_i)$ )
- A partition  $\rho'$  is a *refinement* of a partition  $\rho$ , (or  $\rho$  is *coarser* than  $\rho'$ )  
 $\rho' \sqsubseteq \rho$ , if and only if  $\forall B' \in \rho' . \exists B \in \rho . (B' \subseteq B)$

## Stability ( $\pi$ )

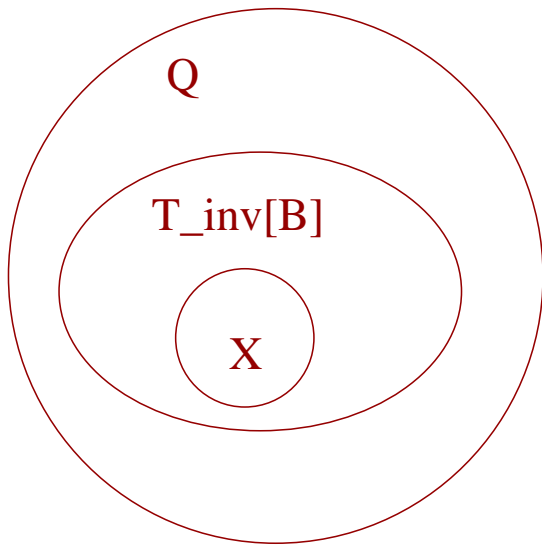
If  $X \subseteq Q$  and  $B \subseteq Q$  then  $X$  is *stable* with respect to  $B$  if

- $X$  is contained in the range of  $B$ ,  $X \subseteq T_\alpha^{-1}[B]$ , or
- $X$  is independent of  $B$ ,  $X \cap T_\alpha^{-1}[B] = \phi$

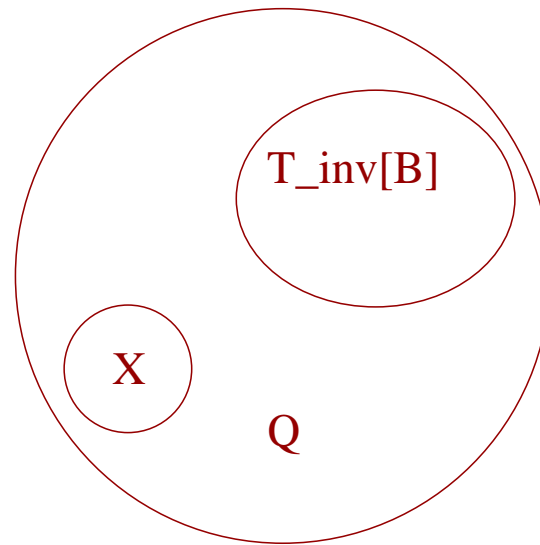
If  $\rho$  is a partition of  $Q$ ,  $\rho$  is *stable* with respect to  $B$  if all blocks of  $\rho$  are *stable* with respect to  $S$ .

$$\pi_{\alpha,B}(\rho) = \forall X \in \rho . (X \subseteq T_\alpha^{-1}[B] \vee X \cap T_\alpha^{-1}[B] = \phi)$$

## Stability ( $\pi$ )



completely contained and stable



$X$  is independent and hence stable

## Stability ( $\pi$ )

- $\rho$  is *stable* if it is *stable* with each of its own blocks.
- Stability is *inherited* under refinement  
(if  $\rho_1 \sqsubseteq \rho_2$ , and  $\rho_2$  is stable with respect to  $B$ , then  $\rho_1$  is also stable with respect to  $B$ .)
- Stability is *inherited* under union  
(if  $\rho_1$  is stable with respect to both  $B_1$  and  $B_2$ , then  $\rho_1$  is also stable with respect to  $B_1 \cup B_2$ .)

# Partition Refinement Problem

- Compute the stable partition of a graph automatically
- This is useful for two reasons –
  - Solving multiple verification problems (model checking)
  - Reducing individual components before composing them

# Relational Coarsest Partition Refinement Problem

Given a relation  $\rho$  and an initial partition  $\rho_I$  on a set  $Q$ , *find* the coarsest stable refinement of  $\rho_I$

(*i.e.* *find* the partition  $\Psi(\rho_I)$  such that every other stable partition is a refinement of  $\Psi(\rho_I)$  and  $\Psi(\rho_I)$  has the least number of equivalence classes (fewest blocks, coarsest).)

Calculating  $\Psi(\rho_I)$  gives the maximal Bisimulation Equivalence relation for a labeled transition system  $\langle Q, A, T, q_0 \rangle$ , with  $\rho_I = \{Q\}$ .

# Finding the Relational Coarsest Partition

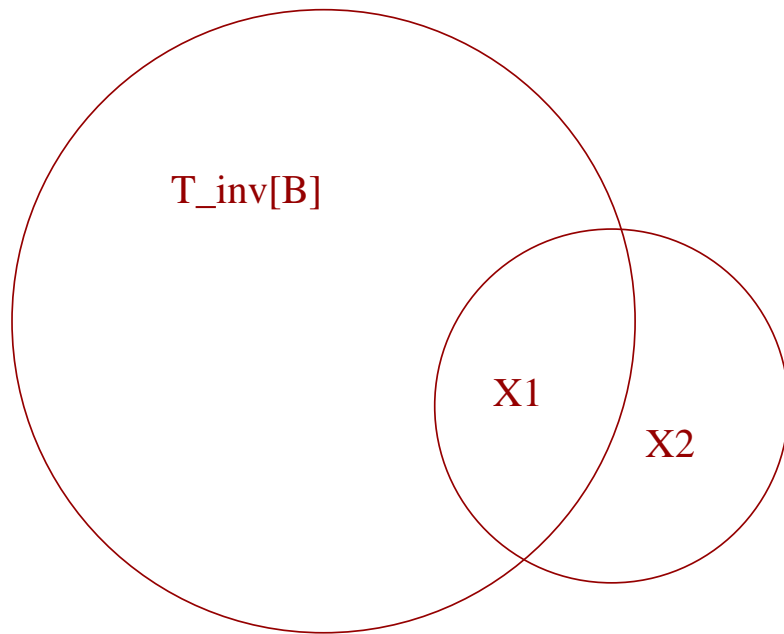
- Kannellakis Smolka Algorithm
  - $O((m + n)n)$  time complexity
  - $O(m + n)$  space complexity
- Paige Tarjan Algorithm
  - $O(m \log n)$  time complexity
  - $O(m)$  space complexity

## Split Function $Split(B, \rho)$

For any partition  $\rho$  and subset  $B \subseteq Q$ ,  
 $Split(B, \rho)$  is a refinement of  $\rho$  such that

- for every block  $X \in \rho$  such that  
 $X \subseteq T_\alpha^{-1}[B] \neq \phi$  and  $X \cap T_\alpha^{-1}[B] \neq \phi$
- replace  $X$  by  $X_1$  and  $X_2$  such that
  - $X_1 \subseteq T_\alpha^{-1}[B]$ , and
  - $X_2 \cap T_\alpha^{-1}[B] = \phi$
- $B$  is a *splitter* of  $\rho$  if  $split(B, \rho) \neq \rho$

## Simple Splitter $Split(B, X)$



$X1$  is in the preimage of  $B$

$X2$  is independent of preimage of  $B$

## Splitting (continued ...)

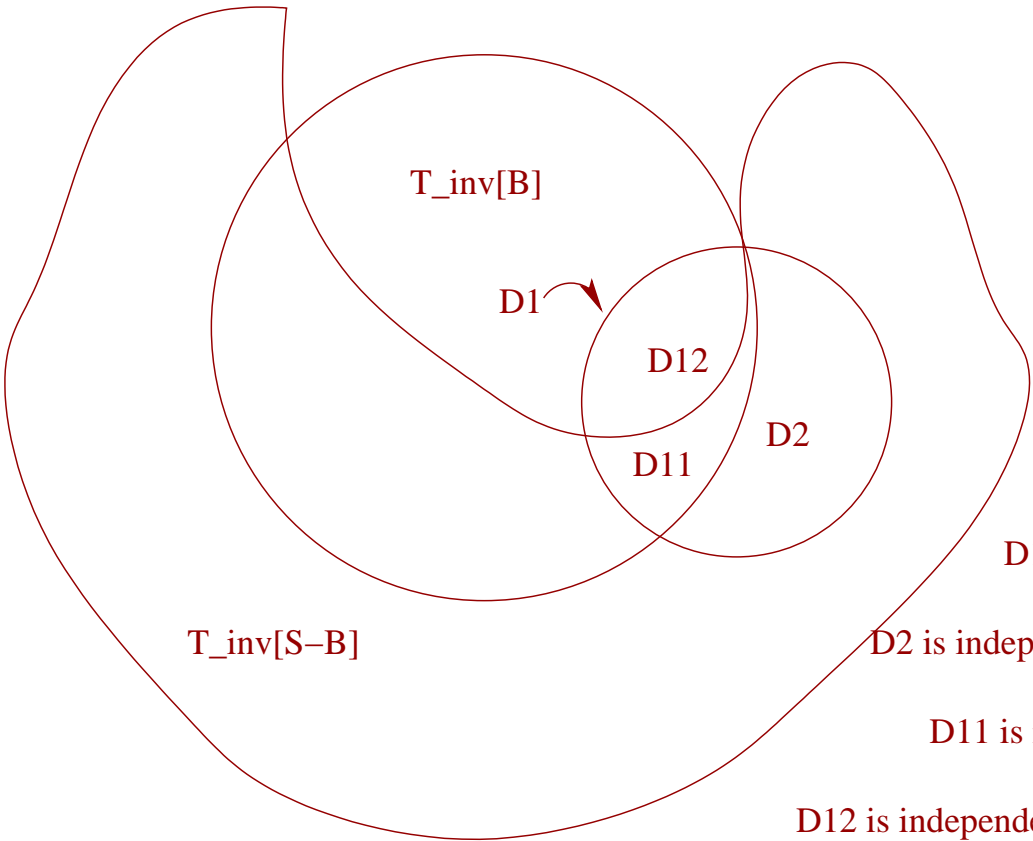
- $\rho$  is *unstable* with respect to  $B$  if and only if  $B$  is a splitter of  $\rho$
- *split* is monotone in its partition argument  
i.e. if  $B \in Q$  and  $\rho_1 \subseteq \rho_2$  then  $split(B, \rho_1) \subseteq split(B, \rho_2)$
- Function *split* is *commutative*, i.e., the coarsest partition of  $Q$  stable with respect to both  $\rho_1$  and  $\rho_2$  is  $split(\rho_1, split(\rho_2, Q)) = split(\rho_2, split(\rho_1, Q))$

## Compound Splitters

Suppose that partition  $\rho$  is stable with respect to a set  $S$  that is a union of some of the blocks of  $\rho$ . Suppose also that partition  $\rho$  is refined first with respect to a block  $B \subseteq S$  and then with respect to  $S - B$ . Then the following conditions hold :

- Refining  $\rho$  with respect to  $B$  splits a block  $D \in \rho$  into two blocks  $\mathcal{D}_1 = D \cap T^{-1}[B]$  and  $\mathcal{D}_2 = D - \mathcal{D}_1$  if and only if  $D \cap T^{-1}[B] \neq \phi$  and  $D - T^{-1}[B] \neq \phi$ .
- Refining  $split(\mathcal{B}, \rho)$  with respect to  $S - B$  splits  $\mathcal{D}_1$  into two blocks  $\mathcal{D}_{11} = \mathcal{D}_1 \cap T^{-1}[S - B]$  and  $\mathcal{D}_{12} = \mathcal{D}_1 - \mathcal{D}_{11}$  if and only if  $\mathcal{D}_1 \cap T^{-1}[S - B] \neq \phi$  and  $\mathcal{D}_1 - T^{-1}[S - B] \neq \phi$ .

# Compound Splitter $Split(\mathcal{S}, \rho)$



$D1$  is in the preimage of  $B$

$D2$  is independent of preimage of  $B$

$D11$  is in the preimage of  $S - B$

$D12$  is independent of preimage of  $S - B$

## Compound Splitters (continued ...)

- Refining  $\text{split}(\mathcal{B}, \rho)$  with respect to  $\mathcal{S} - \mathcal{B}$  does not split  $\mathcal{D}_2$  (because  $\mathcal{D}_2 \subseteq \mathcal{T}^{-1}[\mathcal{S} - \mathcal{B}]$ ).
- $\mathcal{D}_{12} = \mathcal{D}_1 \cap (\mathcal{T}^{-1}[\mathcal{B}] - \mathcal{T}^{-1}[\mathcal{S} - \mathcal{B}])$ .

# Adapted Paige-Tarjan Algorithm

Given an initial partition  $\rho_I$ , a system  $(Q, A, T, q_0)$ ,  
and an initial set of splitters  $W = \{Q\}$

Repeat for every splitter  $B \in W$  until  $W = \phi$ ,

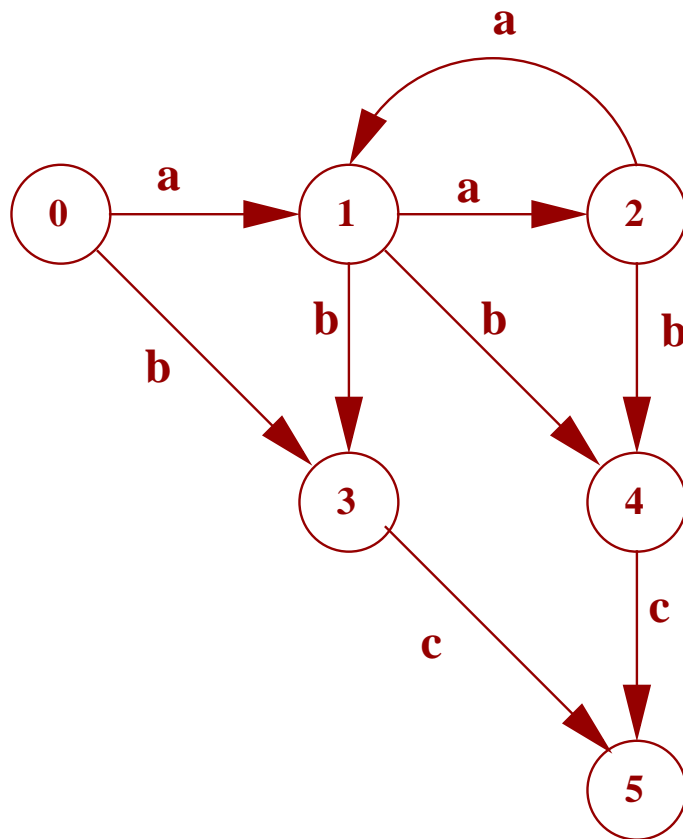
- if  $B$  is a simple splitter, remove  $B$  from  $W$
- for every  $\alpha \in A$ , do Simple\_Splitter\_Refinement
- if  $B$  is a compound splitter  $B_1 \cup B_2$ , remove  $B$  from  $W$
- for every  $\alpha \in A$ , do Compound\_Splitter\_Refinement

## Example

Consider the labeled transition system  $(Q, A, T, q_0)$  :

- $Q = \{0, 1, 2, 3, 4, 5\}$
- $A = \{a, b, c\}$
- $T_a[0] = \{1\}$ ,  $T_a[1] = \{2\}$ ,  $T_a[2] = \{1\}$
- $T_b[0] = \{3\}$ ,  $T_b[1] = \{3, 4\}$ ,  $T_b[2] = \{4\}$
- $T_c[3] = \{5\}$ ,  $T_c[4] = \{5\}$
- $\rho = \{B_0\}$ ,  $B_0 = \{0, 1, 2, 3, 4, 5\}$ ,  $W = \{B_0\}$

## Example – Transition Graph



## Simple\_Splitter\_Refinement

- compute the set  $I = \{X \mid \exists X \in \rho \wedge X_1 = X \cap T_\alpha^{-1}[B] \neq \phi\}$
- compute  $X_2 = X - T_\alpha^{-1}[B]$
- for every  $X_1$ , if  $X_2 = X_1$  do nothing
- if  $X_2 \neq X_1$ , add  $X_1$  to  $\rho$
- if  $X_2 \in W$ , and  $X_2$  is a simple splitter then add  $X_1$  to  $W$
- if  $X_2 \in W$ , and  $X_2$  is a leaf of a compound splitter, then create a compound splitter  $X_{12}$  with leaves  $X_1$  and  $X_2$  and add  $X_{12}$  to  $W$
- if  $X_2 \notin W$ , then create a compound splitter  $X_{12}$  with leaves  $X_1$  and  $X_2$  and add  $X_{12}$  to  $W$

## Example – refinement wrt $B_0$ (action $a$ )

action  $a$

- $T_a^{-1}[B_0] = \{0, 1, 2\}$
- $B_1 = \{0, 1, 2\}$
- $B_2 = \{3, 4, 5\}$
- $\rho = \{B_1, B_2\}$
- $W = \{(B_0, B_1, B_2)\}$

## Example – refinement wrt $B_0$ (action $b$ )

action  $b$

- $T_b^{-1}[B_0] = \{0, 1, 2\}$
- $\rho$  and  $W$  are not modified

## Example – refinement wrt $B_0$ (action $c$ )

action  $c$

- $T_c^{-1}[B_0] = \{3, 4\}$
- $B_3 = \{3, 4\}$
- $B_4 = \{5\}$
- $\rho = \{B_1, B_3, B_4\}$
- $W = \{(B_0, B_1, B_2), (B_2, B_3, B_4)\}$

## Compound\_Splitter\_Refinement

- compute the set  $I = \{X \mid \exists X \in \rho \wedge X_1 = X \subseteq T_\alpha^{-1}[B]\}$
- split compound splitter  $X$  into  $X_1$ ,  $X_2$ , and  $X_3$  using the original rule
- update  $\rho$  by replacing  $X$  with  $\{X_1, X_2, X_3\}$
- update  $W$  similar to the simple case
- create a compound splitter  $X_{23}$  pointing to  $X_2$  and  $X_3$ , and another compound splitter  $X_{123}$  pointing to  $X_1$  and  $X_{23}$

## Compound\_Splitter\_Refinement

To facilitate three-way compound splitting we need one more collection of records.

- For each block  $S$  of  $\rho$  and each element  $x \in T^{-1}[S]$ , we maintain a record containing the integer  $count(x, S) = |S \cap T[\{x\}]|$ .
- We will not distinguish this record from the count itself.
- Each edge  $(x, y) \in T$  such that  $y \in S$  contains a pointer to  $count(x, S)$ .

## Example – further refinement

- refinement with respect to  $(B_0, B_1, B_2)$   
 $\rho$  is not modified and  $W = \{(B_2, B_3, B_4)\}$
- refinement with respect to  $(B_2, B_3, B_4)$   
 $\rho$  is not modified and  $W = \phi$
- Final partition  $\Psi(\rho) = \{\{0, 1, 2\}, \{3, 4\}, \{5\}\}$

## Adapted PT algorithm – summary

$\rho'$  is the current partition.  $\rho$  is a previous partition.  $\rho' \sqsubseteq \rho$  ( $\rho'$  refines  $\rho$ ).  
 $W$  is the set of compound blocks of  $\rho$ .

- [Select a refining block]
  - Remove some block  $S$  from  $W$  (block  $S$  is a compound block of  $\rho$ ).
  - Examine the first two blocks in the list of blocks of  $\rho'$  contained in  $S$ . Let  $B$  be the smaller (break a tie arbitrarily).
- [update  $\rho$ ]
  - Remove  $B$  from  $S$  and create a new (simple) block  $S'$  of  $\rho'$  containing  $B$  as its only block of  $\rho$ .
  - If  $S$  is still compound, put  $S$  back into  $W$ .

## Adapted PT algorithm – summary (continued ...)

- [refine  $\rho'$  with respect to  $B$ ]
  - For each block  $D$  of  $\rho'$  containing some element of  $T^{-1}[B]$ , split  $D$  into  $\mathcal{D}_1 = D \cap \mathcal{E}^{-1}(B)$  and  $\mathcal{D}_2 = D - \mathcal{D}_1$ .
  - If  $\mathcal{D}_2$  is empty, remove it from  $\rho'$ .
  - If it is non-empty, and the block of  $\rho$  containing  $\mathcal{D}_1$  and  $\mathcal{D}_2$  has been made compound by the split, add it to  $W$ .

## Adapted PT algorithm – summary (continued ...)

- [refine  $\rho'$  with respect to  $S - B$ ]
  - Each block  $\mathcal{D}_\infty$  splits into two blocks  $\mathcal{D}_{11} = \mathcal{D}_1 \cap \mathcal{T}^{-1}[S - B]$  and  $\mathcal{D}_{12} = \mathcal{D}_1 - \mathcal{D}_{11}$ .
  - If  $\mathcal{D}_{12}$  is empty, remove it from  $\rho'$ .
  - If it is non-empty, and the block of  $\rho'$  containing  $\mathcal{D}_{11}$  and  $\mathcal{D}_{12}$  has been made compound by the split, add it to  $W$ .

## Data Structures used

Several Data Structures are required to represent – states, classes, and splitters.

- Each state  $p$  points to a list of tuples  $(\alpha, T_\alpha^{-1} [p])$
- Each class of  $\rho$  has an associated integer giving its size and points to its list of elements
- Each state points to its predecessor in its class and to the class containing it (this allows for deletion in  $O(1)$  time)

## Data Structures used (continued ...)

- $W$  is a set of splitters
- A compound splitter  $B$  is represented as a binary tree, with the *count* associated with the root, and has  $B_1$  and  $B_2$  as children if  $B = B_1 \cup B_2$
- Each class maintains an integer indicating whether the class itself is in  $W$  or if it is a leaf of a compound splitter.
- The space needed for representing all the data structures is  $O(m)$ , as is the initialization time.

## Hopcroft's process the smaller half

Consider the special case in which  $\mathcal{T}$  is a function,

- $|\mathcal{T}(\{x\})| = 1$  for all  $x \in Q$ .
- if  $\rho$  is a partition stable with respect to a set  $S$  that is a union of some of the blocks of  $\rho$ ,  
and  $\mathcal{B} \subseteq S$  is a block of  $\rho$ ,
- then  $split(\mathcal{B}, \rho)$  is stable with respect to  $S - \mathcal{B}$ ,
- since if  $\mathcal{B}_1$  is a block of  $split(\mathcal{B}, \rho)$ ,  $\mathcal{B}_1 \subseteq \mathcal{T}^{-1}[\mathcal{B}]$
- implies  $\mathcal{B}_1 \cap \mathcal{T}^{-1}[S - \mathcal{B}] = \phi$

## Hopcroft's process the smaller half

- also  $\mathcal{B}_1 \subseteq \mathcal{T}^{-1}[\mathcal{S}] - \mathcal{T}^{-1}[\mathcal{B}]$
- implies  $\mathcal{B}_1 \subseteq \mathcal{T}^{-1}[\mathcal{S} - \mathcal{B}]$ .
- This means that in each refinement step it suffices to replace  $\mathcal{Q}$  by  $\mathit{split}(\mathcal{B}, \mathcal{Q})$ ,
- since  $\mathit{split}(\mathcal{B}, \mathcal{Q}) = \mathit{split}(\mathcal{S} - \mathcal{B}, \mathit{split}(\mathcal{B}, \mathcal{Q}))$ .

This is the idea underlying Hopcroft's process the smaller half strategy [AHU74] [Hop71], the refining set  $\mathcal{B}$  is at most half the size of the stable set  $\mathcal{S}$  containing it.

## Time Complexity $O(m \log n)$

- The time spent in the refinement step is  $O(1)$  per edge scanned plus  $O(1)$  per vertex of  $\mathcal{B}$ , for a total of  $O(|\mathcal{B}| + \sum_{y \in \mathcal{B}} |\mathcal{T}^{-1}(\{y\})|)$ .
- A given element  $x \in Q$  is in at most  $\log(n + 1)$  different blocks  $\mathcal{B}$ .

This gives us an overall  $O(m \log n)$  time complexity for the algorithm.

Although this algorithm resembles the Hopcroft algorithm, and the Cardon-Crochemore algorithm, and the Kannellakis-Smolka algorithm for the bounded fanout case, this algorithm is simpler because, by refining with respect to old blocks rather than current ones, it is possible to avoid some updating. (The other algorithms must update  $\mathcal{L}$  each time a block in  $\mathcal{L}$  is split.)

## Bisimulation Minimizations

- Bisimulation Equivalence  $\sim_{bis}$  preserves all properties expressible in *LTL*, *CTL*, *CTL\**, and the  $\mu$  – calculus
- $\sim_{bis}$  can be automatically calculated and also yields the smallest model preserving all  $\mu$  – calculus formulae
- Therefore,  $\sim_{bis}$  is attractive for minimizing state-spaces in model checking
- *Bisimulation Minimization* provides an abstraction technique that preserves the truth and falsehood of all  $\mu$  – calculus properties
- Many model checking approaches represent state spaces symbolically. Efficient symbolic algorithms are therefore desirable in model checking.

# Bisimulation Minimization for Symbolic Model Checking

- This technique is particularly appealing in the case of symbolic model checking for two reasons
  - First, bisimulation can be computed as a fixed point of a simple boolean expression, so it is easily expressed symbolically
  - Second, unlike many other abstraction techniques, it can be computed automatically which is consistent with the automatic approach of model checking
- Bisimulation minimization therefore, can be used as a preprocessing phase to model checking, thereby reducing many of the resource requirement of model checking

# Reachable Partition Refinement

**Motivation:** In reality, we are interested only in the reachable portion of the quotient graph

- The reachable stable partition of  $\rho$ , denoted  $\Psi^{\mathcal{R}}(\rho)$ , is the reachable equivalence classes of  $\Psi(\rho)$
- The reachable subgraph of  $\Psi^{\mathcal{R}}(\rho)$  is called the  $\rho$ -minimal-reachable quotient
- If  $\sigma^T$  be a block of  $\rho$ , then, the answer to the reachability problem  $(G, \sigma^T)$  is **true** iff  $\sigma^T \cap \tau$  is nonempty for some  $\tau \in \Psi^{\mathcal{R}}(\rho)$
- Reachable-partition-refinement problem:
  - Input: a transition graph  $G$  and initial partition  $\rho_I$
  - Output: reachable stable partition  $\Psi^{\mathcal{R}}(\rho)$

# Refinement and Reachability

- First solution: compute the reachable region  $\sigma^R$  of  $G$ , and then apply partition refinement algorithm
- Reachable region may be too large (even infinite) while  $\psi^{\mathcal{R}}(\rho_I)$  is small
- Second solution: Compute the coarsest stable refinement  $\Psi(\rho_I)$ , and then, analyze it for reachability
- May cause unnecessary splitting:  $\Psi(\rho_I)$  may be too large (even infinite) while  $\psi^{\mathcal{R}}(\rho)$  is small
- Can we do partition-refinement and reachability analysis simultaneously?
- Goal: Split only those regions that are known to be reachable, maintain reachability between regions

# Conclusions

- We have formally established a description of bisimulation equivalence in terms of the relational coarsest partition problem.
- We have presented an adaptation of the Paige & Tarjan algorithm and its implementation.
- The new algorithm provides an efficient decision procedure for other equivalence relations requiring the computation of bisimulation equivalence.
- Implemented the algorithm in C and tested on simple test cases.

## References

- [AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA 1974., 1974.
- [CC82] A. Cardon and M. Crochemore. Partitioning a graph in  $O(|A| \log |V|)$ . *Theoretical Computer Science*, 19:85–98, 1982.
- [FV98] Kathi Fisler and Moshe Y. Vardi. Bisimulation Minimization in an Automata-Theoretic Verification Framework. In *Formal Methods in Computer-Aided Design*, pp:115-132, 1998.
- [Hop71] J. E. Hopcroft. An  $n \log n$  algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, Ed. by Zvi Kohavi and Azaria Paz, Academic Press. 1971.
- [KS80] P. Kannellakis and S. Smolka. CCS Expressions. *Information and Computation*, 86:43–68, 1990.

## References (continued ...)

- [Lar86] K. G. Larsen. *Context-dependent bisimulation between processes*. PhD thesis, 1986.
- [Mil80] R. Milner. *A calculus of communication systems*, 1980.
- [Par81] D. Park. *Concurrency and automata on infinite sequences*, 1981.
- [Pel01] Doron A. Peled. *Software Reliability Methods*. Springer Verlag, New York City, New York, 2001.
- [PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.