Foundations of Dependable Systems Seminar : Bisimulation and Model Checking

Noé Zufferey, Sylvain Julmy

December 18, 2017

Model Checking 1

The paper focus on the checking of invariance properties. A logical proposition is either true or false for each state. Can the analysed state system reach a system that does not correspond to this logical proposition ?

Model Checking 2

A model checker has to analyse the entire transition state spaces. Some methods are used to reduce the size of the state space. One of them is bisimilution minimization.

The paper study a model checking method which uses bisimilution minimization. Minimized sytems are faster to check.

Model Checking 3

A formal model for symbolic model checking can be :

$\langle S, R, AP, L, init \rangle$

- S : set of states
- R : the transitions over S
- AP : a set of atomic prposition
- L : $S \times AP$, set of pairs atomic prppositions and the states they are true with.
- $init$: the initial state

Bisimilution minimization

The purpose is to reduce the number of states. Bisimulation minimization algorithms analyse a transition state space to find set of states that are equivalent with a given logical proposition.
Both state systems, the first one and the minimization are bisimilar in respect with a logical proposition. They can simulate each other for the model checking.

Bisimilution minimization 2

A common way to Minimize state systems is to split the state space in classes in which states have the main behaviour regarding the logical prposition.
Then each class is repeatidly split into new classes until all states of a class agree on the type of their successor.

Bisimilution minimization 3

A naïve way to compute a bisimultation minimization is to repeatidly look for states that are not bisimilar and/or their succesor are not bisimilar.

$$
\begin{aligned}
&\exists ap \in AP \neg(L(s_1, ap) \Leftrightarrow L(s_2, ap)) \vee \\
&\exists s_1' R(s_1, s_1') \wedge \neg(\exists s_2' (R(s_2, s_2') \wedge B(s_1', s_2'))) \vee \\
&\exists s_2' R(s_2, s_2') \wedge \neg(\exists s_1' (R(s_1, s_1') \wedge B(s_1', s_2')))
\end{aligned}
$$

Bisimilution minimization 4

The first line :

$\exists ap \in AP \neg(L(s_1, ap) \Leftrightarrow L(s_2, ap))$

looks for states that are not directly bisimilar in respect with the logical prposition.
The two last lines :

$\exists s'_1 R(s_1, s'_1) \wedge \neg(\exists s'_2(R(s_2, s'_2) \wedge B(s'_1, s'_2))) \vee$
$\exists s'_2 R(s_2, s'_2) \wedge \neg(\exists s'_1(R(s_1, s'_1) \wedge B(s'_1, s'_2)))$

look for states whose successors are not bisimilar.

Bisimilution minimization 5

The naïve algorithm takes into account the unreachable states.

Terminology

The paper difines a number of terms that are usefull when speaking of bisimulation minimization algorithms.

Terminology 2

- block : set of states
- block's representative : the state that represent its whole blocks
- partition : set of blocks
- initial partition : partition of the two block (good and bad) that are used at the beggining of the algorithms
- good block : initial block composed by states that satisfy the given invariant property
- bad block : initial block composed by states that do not satisfy the given invariant property

Terminology 3

- refine : a partition P_1 refines a partition P_2 iff each blck of P_1 is contained in blocks of P_2
- reachable : a state is reachable if their is an existing path between de initial state and this state. A block is reachable if containing a reachable state.
- stable : a block B_1 is stable with a block B_2 iff every state in B_1 have transition with at least one state of B_2 or if no state in B_1 have transition with states in B_2
- splitter : a splitter of a block is a block that the first black is not stable with

Invariance checking

The paper studies invariance checking. An invariance property is true iff every reachable state satisfies it. Invariance checking can be executed in two ways

- forward reachability
- backward reachability

Backward reachability (BR) is closely related to bisimilution minimization.

Backward reachability

BR has two set of states. The frontier states F that represent the new discovered states and the explored states S. BR iterates the following equations :

$F_0 = Bad$
$\mathrm{F}_{i+1} = pre(F_i) - S_i$

$S_0 = Bad$
$\mathrm{S}_{i+1} = pre(F_i) \cup S_i$

The iteration stop either if $F_i = \emptyset$ or if $init \in F_i$.

Backward reachability 2

The lower bound for BR is $n \cdot (M + U + D + 2E + I)$, with n the number of iteration to terminate.
n is the length of th shortest path between the initial state and a bad state or n is the length of the longest acyclic state from a good state to a bad state.

Algorithms

The paper study 3 diferent algorithms that apply bisimulation minimization for symbolic model checking :

- PT
- BFH
- LY

Each algorithm has been modified for model checking purpose. The authors added termination condition since the invariance has been proven true or false.

PT algorithm

PT stands for Paige-Tarjan.

PT stabilizes every block (reachable and unreachable). It selects splitters to stabize the system instead of block to split.

Two partition are used in PT. The current partition Q and the previous partition X. Q refines X. That basically means that Q contains more blocks.

The algorithm repeatidly look for blocks of X that contains states from multiple blocks of Q. This kind of blocks is called of compound block. Then, the compound block is splitted.

A Block is marked if it contains a bad state. The algorithm fail if init is marked.

PT algorithm 2

The marking had been added to support early termination. If a splitter B is marked, every block that reach B is marked. So, this PT algorithm contains at most 1 marked block (and if zero, it stop with violation) Furthermore, the algorithm never split a marked block, split a marked block is not helpful

PT algorithm 3

- Select a refining block.
- Update X
- Split the block containing elements that are predecessors of the splitter

PT algorithm 4

initialize X to $\{\mathcal{U}\}$
initialize Q to the initial partition $\{Good, Bad\}$
set the initial block to the block containing the initial state
mark Bad
while $Q \neq X$ and the initial block is not marked **do**
select some compound block S from X
let B be the block in S with the smallest number of states
remove B from S and add block S' containing only B to X
$E_1 = pre(B)$
$E_2 = E_1 - pre(S - B)$
foreach block D of Q that contains an element of E_1 and is unmarked
$D_1 = D \cap E_1$
$D_2 = D - D_1$
Replace D in Q with block D_1; re-direct pointer to D in X to D_1
if B is marked **then** mark D_1 **endif**
if D_2 is non-empty **then**
add D_2 to Q
if B is not marked **then** mark D_2 **endif**

PT algorithm 5

add a pointer to D_2 within block of X containing D_1
endif
if D_1 contains an element of E_2 **then**
$D_{11} = D_1 \cap E_2$
$D_{12} = D_1 - D_{11}$
Replace D_1 in Q with block D_{11}; re-direct pointer to D_1 in X to D_{11}
if B is marked or D_1 is marked **then** mark D_{11} **endif**
if D_{12} is non-empty **then**
add D_{12} to Q
add a pointer to D_{12} within block of X containing D_{11}
mark D_{12}
endif
endif
if D was the initial block **then**
set the initial block to the block containing the initial state
endfor
endwhile
if the initial block is marked **then** signal safety violation

PT algorithm 5

The lower bound for PT is $n \cdot (2M + D + I + E)$, with n the number of while iterations

The Lee-Yannakakis algorithm

LY - idea

- Stabilize only reachable blocks.
- Reachable block use a representative that has to be reachable.
- The first state is the representative for the initial block.
- To find new reachable state, we look for transition from representative of reachable state to state from unreachable block.

LY - idea

Two loops :

- Search new, reachable blocks
- Stabilize reachable but unstable blocks

LY - termination

With the exception of the initial block, all new blocks created by the algorithm have paths to the bad block.

LY - termination

Therefore, when a second block becomes reachable, the algorithm should raise a violation and terminate.

LY - new algorithm

Basic idea[1] :

- Search new reachable blocks.
- Stabilize reachable but unstable blocks.
- When a second block becomes reachable $\rightarrow$ raise a violation.

[1] Very similar to BR

LY - search

To search for new reachable block, the algorithm is searching from all the successors of the initial state if one of those is in a different block.

The algorithm also determine if the initial block has to be stabilized or not.

LY - search

$D := post(B)$
for all $\langle C, q \rangle \in post(init)$ **do**
    if $B \neq C$ **then**
        raise violation
    end if
    if $B \cap pre(C) \neq B$ **then** $\triangleright$ Not all predecessors of B are in B
        B is not stable
    end if
    $D := D - C$
end for
if $D \neq \emptyset$ **then** $\triangleright$ $post(init) = \emptyset$
    B is not stable
end if

LY - search

$queue := \emptyset$

$partition = \{B, Bad\}$

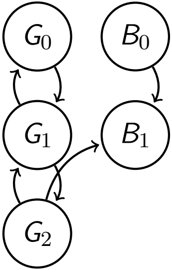
$init = G_0$

$B = \{G_0, \ldots, G_2\}$

$Bad = \{B_0, B_1\}$

$block_{init} = \langle B, init \rangle$

$D = post(B) = \{B, Bad\}$



LY - search

$post(init) = \{B\}$

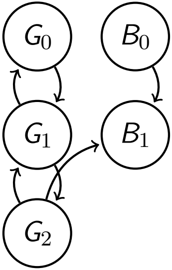
$\langle C, q\rangle = \langle B, init\rangle$

$pre(C) = \{B\}$

$\{B\} \cap pre(C) = \{B\} == \{B\}$

$D = \{B, Bad\} - \{B\} = Bad$

$D \neq \emptyset \rightarrow enqueue(\langle B, init\rangle)$



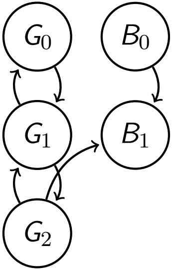
LY - stabilization

while B is not stable **do**
    Mark B as stable
    Compute the frontier of B
    Let B' the state of B that can only reach B
    Let B'' the state of B that can reach a bad block
    if $\emptyset \neq B' \cap pre(B') \neq B'$ or $\emptyset \neq B' \cap pre(B'') \neq B'$ **then**
        Mark B as unstable
    end if
end while

LY - stabilization

Iteration 1

$$init = G_0$$
$$partition = \{B, Bad\}$$
$$B = \{G_0, G_1, G_2\}$$
$$pre(B) = \{B\}$$
$$post(B) = \{B, Bad\}$$



LY - stabilization

Iteration 1

$B_1' = B \cap pre(B) = \{B\}$

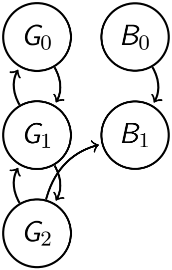
$B_2' = pre(post(B) - B) = pre(\{Bad\}) = \{B\}$

$B' = B_1' - B_2' = \emptyset$

$B'' = B - B' = B$

$partition = \{B, Bad, B\}$

$B := B' = \emptyset$



LY - stabilization

Iteration 1

$B := B' = \emptyset$

$pre(B) = \emptyset$

$B'' = B$

$pre(B'') = B$

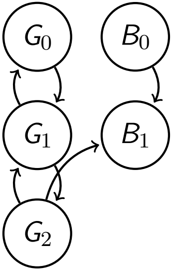
$B \cap pre(B) = \emptyset$

$B \cap pre(B'') = \emptyset$

no enqueue !

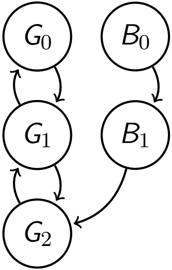
$post(init) \cap B'' = \{B\} \cap \{B\} = \{B\}$

$\rightarrow$ raise safety violation !



LY - search (2)

$queue := \emptyset$
$partition = \{B, Bad\}$
$init = G_0$
$B = \{G_0, \ldots, G_2\}$
$Bad = \{B_0, B_1\}$
$block_{init} = \langle B, init \rangle$
$D = post(B) = \{B\}$



LY - search (2)

$post(init) = \{B\}$

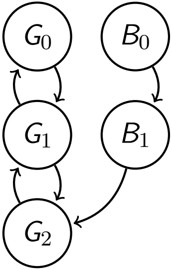
$\langle C, q\rangle = \langle B, init\rangle$

$pre(C) = \{B, Bad\}$

$\{B\} \cap pre(C) = \{B\} == \{B\}$

$D = \{B\} - \{B\} = \emptyset$

no safety violation, terminate



LY - complexity

$$(n-1) * 5M + 4I + 3D + 4E$$

where

- n : number of BR iterations
- M : number of image iterations
- I : number of intersection operations
- D : number of set difference operations
- E : number of equality check
- U : number of union operations

BFH

The Bouajjani-Fernandez-Halbwachs algorithm

BFH - idea

- BFH, like LY, selects reachable blocks to stabilize but differ in how to stabilize a block.
- BFH stabilize a block w.r.t. all the other blocks (either reachable or unreachable).
- The algorithm become simpler but unnecessary work is done.

BFH - termination

As in LY, BFH could terminate when a second block becomes reachable. The algorithm correctly determine violations of invariants but not as soon as they occur.

BFH - termination

The algorithm may traverse a path from the bad block to the initial state before the initial block becomes stable.
Thus, the algorithm takes more iterations to terminate.

BFH - new algorithm

Mark the bad block
$I = [init]_p$
while I is not marked **do**
    $N := split(I, p)$
    if $N = \{I\}$ **then**
        if $post(I) - I \neq \emptyset \rightarrow$ violation, **else** break
    else
        $p := (p - \{I\}) \cup N$
        $I := [init]_p$
    end if
end while
if I is marked **then**
    Signal safety violation
end if

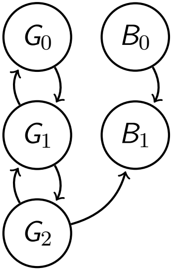
BFH - new algorithm (split)

```
function SPLIT(X : block, p : partition)
    N = {X}
    for all Y : block ∈ p do
        M := ∅
        for all W : state ∈ N do
            W_1 = W ∩ pre(Y)
            if W_1 = W or W_1 = ∅ then
                M := M ∪ {W}
            else
                M := M ∪ {W_1, W − W_1}
            end if
        end for
    end for
    return N
end function
```

BFH - example

Init

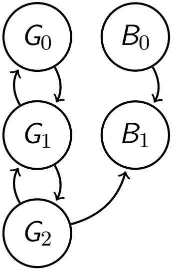
$$I = \{B\}$$
$$p = \{B, Bad\}$$
$$init = G_0$$



BFH - example

Iteration 1

$$N = split(I, p) = ???$$



BFH - example

Iteration 1 - split(1)

$X = B, p = \{B, Bad\}$

$N = \{B\}$

foreach $Y \in p \rightarrow$

$Y = B, M = \emptyset$

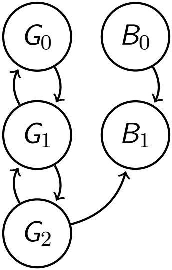
foreach $W \in N \rightarrow$

$W = B$

$W_1 = W \cap pre(Y) = \{B\}$

$\rightarrow M := M \cup \{W\} = \emptyset \cup B = \{B\}$

$N := M = \{B\}$



BFH - example

Iteration 1 - split(2)

$X = B, p = \{B, Bad\}$

$N = \{B\}$

foreach $Y \in p \rightarrow$

$Y = Bad, M = \emptyset$

foreach $W \in N \rightarrow$

$W = B$

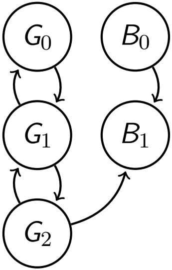
$W_1 = W \cap pre(Y) = B$

Y is marked $\rightarrow$ B is marked

$\rightarrow M := M \cup \{W\} = \emptyset \cup \{B\} = \{B\}$

$N := M = \{B\}$

$return(B)$



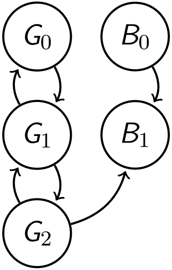
BFH - example

Iteration 1

$N = split(I, p) = \{B\}$

$N = \{I\} \rightarrow post(I) - \{I\} = \{Bad\} \neq \emptyset$

$\rightarrow$ raise safety violation !



BFH - complexity

$$(M + I + 2E) * \frac{n^2 + 3n}{2} + n * D$$

where

- n : number of BR iterations
- M : number of image iterations
- I : number of intersection operations
- D : number of set difference operations
- E : number of equality check
- U : number of union operations

Experimental comparisons

Experimental comparisons

Experimental comparisons

Lower bounds

- BR : $n * (M + U + D + 2E + I)$
- PT : $n * (2M + D + I + E)$
- LY : $(n - 1) * (5M + 4I + 3D + 4E)$
- BFH : $(M + I + 2E) * \frac{n^2+3n}{2} + n * D$

Results

	State Vars	BR			LY		BFH		PT		
		Iter	Time	Mem	Time	Mem	Time	Mem	Iter	Time	Mem
gigamax	16	6	1.8	5.59	2.3	5.60	2.1	5.59	6	2.0	5.58
eisenberg*	17	19	1.1	3.80	3.3	3.99	9.3	4.48	270	9.3	4.28
abp	19	11	0.9	3.81	2.3	3.85	2.8	3.82	19	2.0	3.86
bakery*	20	58	1.3	3.70	6.5	3.87	126.9	9.67	212	7.8	4.55
treearb4	23	24	3.5	4.28	16.9	5.18	99.0	6.14	232	118.3	6.06
elev23	32	1	3.9	8.45	4.2	8.54	4.4	8.51	1	4.0	8.43
coherence1	37	5	3.6	6.28	85.5	22.0	33.0	20.0	23	29.5	8.55
coherence2	37	14	9.3	7.81	279.3	31.0	174.8	21.0	166	567.4	18
coherence3*	37	5	6.5	7.96	84.2	20.0	24.4	11.0	9	7.9	7.89
coherence4*	37	5	7.3	8.58	78.2	18.0	34.4	11.0	685	13.8H	68
elev33	45	1	7.0	11.0	444.5	17.0	443.8	17.0	1	7.2	11
elev43	56	1	11.9	15.0	1590.1	42.0	1661.0	39.0	1	12.2	15
tcp*	80	1	3.1	7.83	3.6	8.06	3.0	8.08	1	3.2	7.83

Figure: Experimental comparison of the various algorithms.

Experimental comparisons

- BR has a better time and memory usage (in almost all cases).
- PT does surprisingly well compared to LY and BFH

PT Optimisation

	Split Marked			Don't Split Marked			Savings		
	Iter	Time	Mem	Iter	Time	Mem	Iter	Time	Mem
gigamax	271	9.2	5.66	6	2.0	5.58	98%	78%	1%
eisenberg*	587	16.5	4.67	270	9.3	4.28	54%	44%	8%
abp	134	6.3	3.96	19	2.0	3.86	85%	68%	3%
bakery*	2279	121.5	6.58	212	7.8	4.55	91%	94%	31%
treearb4	-	(24H)	(54)	232	118.3	6.06			
elev23	3	4.0	8.44	1	4.0	8.43	67%	0%	0%
coherence1	181	70.1	11.0	23	29.5	8.55	87%	58%	22%
coherence2	4376	2320.2	20.0	166	567.4	18	96%	76%	10%
coherence3*	3533	1003.8	18.0	9	7.9	7.89	100%	99%	56%
coherence4*	-	(33H)	(62)	685	13.8H	68			
elev33	3	7.3	11.0	1	7.2	11	67%	1%	0%
elev43	1	11.8	15.0	1	12.2	15	0%	-3%	0%
tcp*	1	3.2	7.84	1	3.2	7.83	0%	0%	0%

Figure: The effect of not splitting marked blocks for PT.

Experimental comparisons

"That PT performs so well compared to BFH and LY suggests that minimisation algorithms tailored to verification settings should pay attention to choosing. "

Conclusion

- Bisimulation is very useful : minimisation technique for model checking, equivalence between transition systems, collapsing infinite-state systems.
- Assumption : the big problem is from the algorithm, not from the representation.
- Minimisation and backward reachability are similar in the spirit of testing invariant properties.
- Creation of three new algorithms : PT, LY and BFH.

What to retain ?

Bisimulation and Model Checking require more resources than Model Checking alone.