Efficient Counterexample Generation Through Permutation of Independent Transitions

Zhen Zhang and Landon Taylor

DRAFT - NOTES ONLY

Contents

1 Counterexample Commuting **2**

2 Very Different Counterexamples **5**

2.1 Using Single Enabled Transitions 5

2.2 Disabling a Transition 5

1 Counterexample Commuting

Automated Threshold Selection for Predicate Abstraction for CTMCs

Use the toggle swith model as an example (`https://github.com/fluentverification/Critical-Values-Protot` `blob/master/ToggleSwitch/control.prism`).

1. Starting with

Given a CTMC model $\mathcal{M}$ and the following transient CSL property $\mathsf{P}_{=?}(\Diamond^{[0,T]}\,\Phi)$, this method aims at efficiently providing the estimated lower and upper probability bound, $prob_{min}()$ and $prob_{max}(,)$ for the path formula $\Diamond^{[0,T]}\,\Phi$, respectively. The proposed method is expected to work well for highly concurrent model $\mathcal{M}$.

Procedure for generating paths reaching target states.

This procedure can provide a lower probability bound for reaching a state satisfying Φ.

1. Generate the first shortest counterexample path from the equivalent non-deterministic model $\mathcal{M}'$ of $\mathcal{M}$ using PDR. This path starts from the initial state s_0: $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots s_i \xrightarrow{t_i} s_{i+1} \dots s_{n-1} \xrightarrow{t_{n-1}} s_n$ and the formula Φ holds in its last state $last(\rho) = s_n$, i.e., $last(\rho) \models \Phi$.

2. Construct the set of independent transitions (including empty set), $indp(\rho)$, for path ρ as $indp(\rho) = enabled(s_0) \cap enabled(s_1) \cap \dots \cap enabled(s_i) \cap \dots \cap enabled(s_n)$. Denote $enabled(s_i)$ as the set of enabled transitions in state s_i. Essentially, each transition $t_\alpha \in indp(\rho)$ is enabled in *every* state, i.e., $s_0, s_1, \dots, s_n$ of path ρ. Therefore, t_α is independent of transitions t_0 through t_{n-1}. Also, we need to confirm that executing t_α from the last state of ρ reaches a target state: $last(\rho) = s_n \xrightarrow{t_\alpha} s_{n+1}$ and $s_{n+1} \models \Phi$.

3. If $indp(\rho) \neq \emptyset$, consider each transition $t_\alpha \in indp(\rho)$ and path $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots s_i \xrightarrow{t_i} s_{i+1} \dots s_{n-1} \xrightarrow{t_{n-1}} s_n$. Figure 1 illustrates all possible path interleavings between t_α and transitions in ρ. There are a total of $(n+1)$ unique paths in this figure from s_0 to u. Figure 2 shows the full interleaving of two mutually independent transitions t_α, t_β and transitions in ρ. Nodes with the same identifier (e.g., w_0, w_1, etc.) represent the same state. There are a total of $(n+2)(n+1)$, i.e., $n^2 + 3n + 2$, unique paths in this figure from s_0 to w.

   - How to efficiently enumerate all paths from s_0 to u illustrated in both figures? To calculate the sum of probabilities of all these paths, do we have to enumerate all such paths and simulate each one at a time? Note that each transition's rate (and hence probability) is dependent on its source state. For example, executing t_α in s_0 can have different rate (and therefore probability) than executing it in s_n.
   - How can the method from the previous step be generalized to k mutually independent transitions in $indp(\rho)$?

4. Otherwise, i.e., $indp(\rho) = \emptyset$, construct $indp(\rho)$ from the last k transitions of ρ: $indp(\rho) = enabled(s_i) \cap enabled(s_{i+1}) \cdots \cap enabled(s_{i+(k-1)})$ where $s_{i+(k-1)} = last(\rho)$ and $enabled(s_{i-1}) \cap enabled(s_i) \cap enabled(s_{i+1}) \cdots \cap enabled(s_{i+(k-1)}) = \emptyset$. Then permute every transition in $indp(\rho)$ from state s_i to $s_{i+(k-1)}$. Need to add more details.

5. When generating the next shortest counterexample path ρ', choose an outgoing transition t'_0 from the initial state $s_0$that is *not* in $indp(\rho)$. If t'_0 does not exist, try to find such a transition in the next state s_1, and so on, until such a transition is found.

6. To model check CSL property with upper time bound, consider Riley's algorithm to essentially limit the length of generated counterexample paths.

7. Repeat the above steps.

To improve efficiency in reconstructing paths, we consider transitions beyond those in $indp(\rho)$. Determine independence for a finite sequence $t_{i_1}, \ldots, t_{i_n}$ w.r.t. all transitions along a path returned by PDR, even if not every transition in the sequence $t_{i_1}, \ldots, t_{i_n}$ is independent of all transitions in $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_i \xrightarrow{t_i} s_{i+1} \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n$, as long as the first transition t_{i_1} is an independent transition w.r.t. every transition in $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_i \xrightarrow{t_i} s_{i+1} \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n$. For example, in the six-reaction model, transition sequence t_{R_1}, t_{R_2} can be executed at any state in the following shortest path returned by PDR, such that $s_n \models \Phi$:
$s_0 \underbrace{\xrightarrow{t_{R_4}} s_1 \xrightarrow{t_{R_6}} s_2 \xrightarrow{t_{R_4}} s_3 \xrightarrow{t_{R_6}} s_4 \ldots}_{t_{R_4} \text{ and } t_{R_6} \text{ repeatedly execute 9 times}} \xrightarrow{t_{R_4}} s_n$. Transition t_{R_1} is enabled in every state along this path, but t_{R_2} is not. However, executing t_{R_1} enables t_{R_2} and then executing t_{R_2} *does not* alter the enabledness of t_{R_4} or t_{R_6} in every state along this path. Similarly sequence t_{R_1}, t_{R_3} is also such a sequence. Consider building a dependency graph for syntactic transitions in the PRISM model, as defined in Def. 4.2 of Valmari2011_CanStubbornSetsBeOptimal_journal.pdf.

Procedure for generating paths reaching non-target states.

This procedure attempts to provide an *upper* probability bound for reaching the target state set. Since the CSL property $\mathsf{P}_{=?}(\Diamond^{[0,T]}\,\Phi)$ is of interest, then this procedure finds and reconstructs paths that reach states satisfying $\neg\Phi$. Using the steps above, we can find the lower probability bound for $prob(\Diamond\neg\Phi)$, $prob_{min}(\Diamond\neg\Phi)$. From the principle of duality, we know that $\Diamond\Phi = \neg\Box\neg\Phi$. If one execution (i.e., a path) satisfies $\Diamond\Phi$, then it does not satisfy $\Box\neg\Phi$, and vice versa. Therefore, $prob(\Diamond\Phi) = 1 - prob(\Box\neg\Phi)$. Therefore, $prob_{max}(\Diamond\Phi) = 1 - prob_{min}(\Box\neg\Phi)$. We can use the procedure for generating paths reaching target states described above with the target state set satisfying $\Box\neg\Phi$. Then we can potentially calculate $prob_{min}(\Box\neg\Phi)$ to obtain $prob_{max}(\Diamond\Phi)$.

Our assumption is that $\Diamond^{[0,T]}\,\Phi$ is a rare event, so evaluating $\mathsf{P}_{=?}(\Diamond^{[0,T]}\,\Phi)$ directly in STAMINA may incur challenges. However, since $prob(\Diamond\Phi) = 1 - prob(\Box\neg\Phi)$, we can use STAMINA to obtain a probability window for $\mathsf{P}_{=?}(\Box^{[0,T]}\,\neg\Phi)$: $[prob_{min}(\Box\neg\Phi), prob_{max}(\Box\neg\Phi)]$.

$$
\begin{tikzcd}
s = s_0 \arrow[r, "t_0"] \arrow[d, "t_\alpha"'] & s_1 \arrow[r, "t_1"] \arrow[d, "t_\alpha"'] & s_2 \arrow[r, "t_2"] \arrow[d, "t_\alpha"'] & \cdots \arrow[r, "t_{n-2}"] & s_{n-1} \arrow[r, "t_{n-1}"] \arrow[d, "t_\alpha"'] & s_n \arrow[d, "t_\alpha"'] \\
u_0 \arrow[r, "t_0"] & u_1 \arrow[r, "t_1"] & u_2 \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & u_{n-1} \arrow[r, "t_{n-1}"] & u_n = u
\end{tikzcd}
$$

Figure 1: Interleaving of t_α and transitions in ρ, where $t_\alpha \in indp(\rho)$ and $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_i \xrightarrow{t_i} s_{i+1} \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n$.

$$
\begin{tikzcd}
w_0 \arrow[r, "t_0"] & w_1 \arrow[r, "t_1"] & w_2 \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & w_{n-1} \arrow[r] & w_n = w \\
v_0 \arrow[u, "t_\alpha"] \arrow[r, "t_0"] & v_1 \arrow[u, "t_\alpha"] \arrow[r, "t_1"] & v_2 \arrow[u, "t_\alpha"] \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & v_{n-1} \arrow[u, "t_\alpha"] \arrow[r] & v_n \arrow[u, "t_\alpha"] \\
s = s_0 \arrow[u, "t_\beta"] \arrow[d, "t_\alpha"] \arrow[r, "t_0"] & s_1 \arrow[u, "t_\beta"] \arrow[d, "t_\alpha"] \arrow[r, "t_1"] & s_2 \arrow[u, "t_\beta"] \arrow[d, "t_\alpha"] \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & s_{n-1} \arrow[u, "t_\beta"] \arrow[d, "t_\alpha"] \arrow[r, "t_{n-1}"] & s_n \arrow[u, "t_\beta"] \arrow[d, "t_\alpha"] \\
u_0 \arrow[d, "t_\beta"] \arrow[r, "t_0"] & u_1 \arrow[d, "t_\beta"] \arrow[r, "t_1"] & u_2 \arrow[d, "t_\beta"] \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & u_{n-1} \arrow[d, "t_\beta"] \arrow[r, "t_{n-1}"] & u_n \arrow[d, "t_\beta"] \\
w_0 \arrow[r, "t_0"] & w_1 \arrow[r, "t_1"] & w_2 \arrow[r, "t_2"] & \cdots \arrow[r, "t_{n-2}"] & w_{n-1} \arrow[r, "t_{n-1}"] & w_n = w
\end{tikzcd}
$$

Figure 2: Interleaving of t_α, t_β, and transitions in ρ, where $t_\alpha, t_\beta \in indp(\rho)$ and $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_i \xrightarrow{t_i} s_{i+1} \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n$. Note that t_α and t_β are also assumed to be independent of each other.

Optimize PDR's counterexample generation with permutation of independent transitions.

For the 300 traces PDR generated for the six-reaction model, many of them can be produced by permuting transition r_one at different locations of the first (i.e., the shortest) path returned by PDR.

Can independent transition relation help inductive invariant generation for PDR and/or PrIC3?

Need to look into whether and how an independent transition informs about one step relative inductiveness.

Assume-guarantee view of chemical reaction network?

Use the six-reaction model as an example. Based on the initial condition and the property of interest Φ, we first identify transition(s) that can help to reach a target state satisfying Φ.

If we model each reaction as a separate object in IVy, can we use assume-guarantee reasoning to check for reachability and generate paths?

Utilize mutual induction (like the ping-pong example) between R4 and its environment, i.e., composition of other five objects, up to k steps?

2 Very Different Counterexamples

2.1 Using Single Enabled Transitions

Consider a trace $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_i \xrightarrow{t_i} s_{i+1} \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n$. At any state s_k, it is likely that several transitions are enabled. Using that knowledge, we propose the following process to find a very new path after commuting once:

1. Starting with state s_k, take one of the available transitions. This transition should not be the independent transition or the transition in the existing path. Then, you will be in s_{k+1}'.
2. Set s_{k+1}' as the new initial state in an IVy model. Use PDR to generate a path from s_{k+1}' to s_n (such that $s_n \models \Phi$, s_n is a target state)
3. Use the newly generated path as the seed path. Starting with either $k = 0$ or $k = n$, generate and commute more paths.

2.2 Disabling a Transition

Consider the following process:

1. In the IVy model, disable a single transition t_k in some way. Keep the rest of the model the same, including the initial state and property to check. Ways to disable t_k could include:
   - Using an invariant and flag variable, never allow the transition to have been fired (probably the slowest option)
   - Comment out the transition entirely (probably the fastest option)
   - Permanently set the transition's guard to false
2. Check the IVy model. Either result can give useful information to a user:
   - If we get a trace back, we know that t_k is not required to reach the target from the initial state. We also have a fresh (and very different) seed path to try our commuting algorithm on.
   - If we get *unreachable* back, the model will not ever reach the target state without using t_k. We do not get a new path, but we can inform the user that the model will never reach the target without using t_k, so modifying the probability or existence of t_k can help modify the probability of reaching the target
3. Repeat for all $t_k \in T$.