Supplementary material to Fast Parametric Model Checking through Model Fragmentation

Theorem 2. *Function* FRAGMENTATION *returns a valid fragmentation of the graph $\mathcal{G}(V, E)$ as restructured by* TRAVERSE.

Proof. We prove this result by showing that: (a) the set FS assembled by FRAGMENTATION comprises only valid fragments; (b) the fragments from FS are disjoint (i.e., no vertex from V is included in more than one fragment); and (c) the function terminates and returns a set of fragments FS that includes all the vertices from V.

To prove part (a), we note that new tuples are added to FS in three lines from Algorithms 1 and 2. In line 2 of Algorithm 1 and line 7 of Algorithm 2, FS is augmented with tuples that represent degenerate, single-state fragments according to the extended definition of a fragment from the first paragraph of Section IV-A. Finally, in line 23 of Algorithm 1, FS is augmented with a tuple that either passes the fragment-validity check from line 20, or is reduced to a degenerate, single-state fragment in line 21 before it is included in FS. As such, any tuple inserted into FS is a valid fragment.

To prove part (b), we note that the vertices from the reachability set R provided as an argument to FRAGMENTATION are each placed into a degenerate, single-state fragment in line 2 of Algorithm 1, and that the vertex set Z of any fragment $(Z, z_0, Z_{\mathsf{OUT}})$ added to FS comes from $V \setminus R$, where R is updated (in line 24 of Algorithm 1, and in line 8 of Algorithm 2) to include all the vertices of new fragments included in FS. To see that the vertices of all new fragments come from $V \setminus R$, observe that vertex z_0 added to Z in line 4 of the algorithm comes directly from $V \setminus R$; and vertex w added to Z in line 18 (or included into FS as the only vertex of a degenerate fragment in line 7 of Algorithm 2) comes from the stack T, which can only acquire vertices from $V \setminus R$ (as enforced by the if statements before lines 5 and 16 from Algorithm 2). Therefore, the fragments from FS are disjoint.

To prove part (c), we note that both algorithms terminate. TRAVERSE terminates because each of its statements (including the assembly of the vertex sets I and O, and its for loop) operate with a finite number of vertices. FRAGMENTATION terminates because: (i) each iteration of its for loop adds at least one vertex (i.e., z_0) to R in line 24 until $V \setminus R = \{\}$ in line 3 (since V is a finite set) and the loop terminates with all vertices from V included in fragments from FS; (ii) its while loop terminates since it iterates over the elements of stack T that can only contain one instance of vertices from the finite set V and is therefore finite; and (iii) RESTRUCTURE invocations can only add a finite number of vertices to V, as shown in Sect. IV-B. Thus, FRAGMENTATION terminates, returning a fragment set FS that includes all the vertices from V. □

Theorem 3. *The function* FRAGMENTATION *requires at most* $\mathsf{O}(n^2d^4)$ *steps, where n is the number of vertices in V and* $d = \max_{v \in V}\{indegree(v), outdegree(v)\}$.

Proof. The function RESTRUCTURING requires at most $\mathsf{O}(d^2)$ steps to process up to $outdegree(z) - 1$ outgoing edges of a vertex z (Fig. 6a), or up to $(indegree(z) - 1)outdegree(z)$ pairs of incoming-outgoing edges of z (Fig. 6b). Additionally, the model restructuring technique from Fig. 6a may add up to $outdegree(z) - 1$ new vertices for each vertex z in V, yielding a restructured graph with nd vertices in the worst-case scenario.

The function TRAVERSE requires at most $\mathsf{O}(d^3)$ steps, as assembling the set I and processing it in lines 2–11 requires the inspection of the $indegree(w)$ incoming edges of w, assembling the set O requires $outdegree(w)$ steps, and processing it involves up to $outdegree(w)$ executions of the $\mathsf{O}(d^2)$-step RESTRUCTURING.

In the worst-case scenario where the fragmentation produces only single-vertex fragments, the execution of FRAGMENTATION requires the execution of its for loop from lines 3–25 for each of the n vertices of the original graph (with any additional vertices created by restructuring included into the same fragment as the vertex that led to their creation, cf. Fig. 6a). Each iteration of this loop executes: (i) TRAVERSE in line 6 in $\mathsf{O}(d^3)$ steps; (ii) the while loop from lines 7–19, which has at most nd iterations (one for each vertex from the restructured graph) that may each invoke the $\mathsf{O}(d^3)$-step TRAVERSE or the $\mathsf{O}(d^2)$-step RESTRUCTURING; and (iii) the fragment validity check from line 20 which requires no more than $\mathsf{O}(nd)$ operations. Thus, each iteration of the for loop from lines 3–25 is completed in no more than $\mathsf{O}(nd^4)$ steps (due to the while loop from lines 7–19), and the entire FRAGMENTATION requires $\mathsf{O}(n^2d^4)$ steps in the worst-case scenario. □

Note that the coefficients associated with n and d from the big-O notation in Theorem 3 are typically well below 1. For instance, in our experiments the for loop from FRAGMENTATION was only executed for a small fraction of the vertices in V (since many fragments with multiple vertices are typically produced), and RESTRUCTURING was only executed sparingly. Furthermore, it is worth noting that pDTMCs typically sparsely connected graphs, and therefore d is relatively small.