Here is a algorithm for State Merge Modified Parallel Composition (SMMPC).

Algorithm1 is an overall picture. It starts from initial states of an environment model and testers. After generating an initial state of a game (line 3,4), it composes successor states, transitions and a valuation function according to the transition of environment state (line 7-18).

Algorithm 2 compose states(line 11) according to the rule of SMMPC. It gets successor states of testers according to the environment transition(line 5). If some of states are violation transitions, they are replaced with wild card states, which can be replaced with any other normal state in the tester (line 7). **Algorithm 2** also generate the set of property violations p_δ which is registered in valuation function in **Algorithm 1** (line 8).

Algorithm 3 search and connect to a state to which a violation state can be merged when the violation state is composed in **Algorithm 2** (line 4-6) . It generate a new state if there does not exist states which can be merged (line 11-14). When generating, it replaces the wild card states $*T_{\phi_i}$ to the initial states $s_{0T_{\phi_i}}$ of each tester (line 12) . **Algorithm 3** puts the generated state into the stack to continue composing successor states in **Algorithm1**(line 15).

Algorithm 1 Generating analysis space with merging state

1: INPUT: $E = (S_E, A, \Delta_E, s_{0E}, P_E, v_E)$,
$T_\Phi = \{T_{\phi_i} = (S_{T_{\phi_i}}, A, \Delta_{T_{\phi_i}}, s_{0T_{\phi_i}}, \{\neg\phi_i\}, v_{\phi_i}) | \phi_i \in \Phi\}, \Phi$
2: OUTPUT: $E_{\|'_*} = (S_{\|'_*}, A, \Delta_a, s_{0\|'_*}, \Phi, v_{\|'_*} : \Delta \times \Phi)$
3: $s_{0\|'_*} \leftarrow \{(s_{0E}, s_{0T_{\phi_1}}, s_{0T_{\phi_2}}, ...)\}$ such that $\forall |\Phi| \geq i \geq 1, \phi_i \in \Phi$
4: $S_{\|'_*} \leftarrow \{s_{0\|'_*}\}$
5: $stack_{\|'_*} \leftarrow \{s_{0\|'_*}\}$
6: **while** $stack_{\|'_*} \neq \emptyset$ **do**
7: **for all** $s_{\|'_*} \in stack_{\|'_*}$ **do**
8: $stack_{\|'_*} \leftarrow stack_{\|'_*} \backslash s_{\|'_*}$
9: **for all** $a \in A | s_E \in s_{\|'_*} \cap S_E, (s_E, a, s'_E) \in \Delta_E$ **do**
10: $p_\delta \leftarrow \{\}$
11: $s'_{\|'_*} \leftarrow composeState(s_{\|'_*}, a, p_\delta)$
12: $connectToMergiableState(s'_{\|'_*}, S_{\|'_*}, stack_{\|'_*})$
13: $S_{\|'_*} = S_{\|'_*} \cup \{s'_{\|'_*}\}$
14: $\delta_{\|'_*} = (s_{\|'_*}, a, s'_{\|'_*})$
15: $\Delta\|'_* \leftarrow \Delta_{\|'_*} \cup \{\delta_{\|'_*}\}$
16: $v_{\|'_*}(\delta_{\|'_*}) \leftarrow p_\delta$
17: **end for**
18: **end for**
19: **end while**
20: $E_{\|'_*} \leftarrow (S_{\|'_*}, A, \Delta_{\|'_*}, s_{0\|'_*}, \Phi, v_{\|'_*})$
21: **return** $E_{\|'_*}$

Algorithm 2 composeState

INPUT:$s_{\|'_*} \in S_{\|'_*}, a \in A, p_\delta$
OUTPUT:$s'_{\|'_*}$
for all $s_{T_{\phi_i}} \in s_{\|'_*} \cap S_{T_{\phi_i}} | \phi_i \in \Phi, |\Phi| \geq i \geq 1$ **do**
  if $\exists \delta_{T_{\phi_i}} \in \Delta_{T_{\phi_i}} | \delta_{T_{\phi_i}} = (s_{T_{\phi_i}}, a, s'_{T_{\phi_i}})$ **then**
    $s'_{T_{\phi_i}} \leftarrow \delta_{T_{\phi_i}}(s_{T_{\phi_i}}, a)$
    if $v_{T_{\phi_i}}(s'_{T_{\phi_i}}) = \{\neg\phi_i\}$ **then**
      $s'_{T_{\phi_i}} = *_{T_{\phi_i}}$
      $p_\delta \leftarrow p_\delta \cup \{\neg\phi_i\}$
    end if
  else
    $s'_{T_{\phi_i}} = s_{T_{\phi_i}}$
  end if
end for
$s'_{\|'_*} \leftarrow (s'_E, s'_{T_{\phi_1}}, s'_{T_{\phi_2}}, ..)$
return $s'_{\|'_*}$

Algorithm 3 connectToMergiableState

INPUT:$s_{\|'_*}, S_{\|'_*}, stack_{\|'_*}$
$s \leftarrow \emptyset$
for all $s_{\|'_*} \in S_{\|'_*}$ **do**
  if $\forall \phi_i \in \Phi | (s'_{T_{\phi_i}} = s_{T_{\phi_i}} \vee s'_{T_{\phi_i}} = *T_{\phi_i}) \wedge s_{T_{\phi_i}} \in s_{\|'_*} \wedge s'_{T_{\phi_i}} \in s'_{\|'_*}$ **then**
    $s \leftarrow s_{\|'_*}$
  end if
end for
if $s \neq \emptyset$ **then**
  $s'_{\|'_*} \leftarrow s$
else
  for all $s'_{T_{\phi_i}} \in s'_{\|'_*} | s'_{T_{\phi_i}} = *_{T_{\phi_i}}$ **do**
    $s'_{T_{\phi_i}} \leftarrow s_{0T_{\phi_i}}$
  end for
  $s'_{\|'_*} \leftarrow (s'_E, s'_{T_{\phi_1}}, s'_{T_{\phi_2}}, ..)$
  $stack_{\|'_*} \leftarrow stack_{\|'_*} \cup s'_{\|'_*}$
end if