Algorithm 1 SeedSubTrees(Γ)

```
for $rootC \in compSet$ do
    if (Empty($\Gamma_{rootC}$)) then
        continue;
    end if
    $level$ = [ ]; {dynamic array of failed components at subTree's current level}
    $nFailed = (0, 0, \ldots, 0)$; {counts failed components of each type}
    $BFHist = ((\,), (\,), \ldots, (\,))$; {an array of linked lists that keeps a breadth-first history of subTrees, array is indexed by component type, linked list for each component type stores parents in breadth-first order}
    add $rootC$ to $level$;
    $nFailed[rootC] = 1$;
    add @ to $BFHist[rootC]$; {signifies one component of type rootC has failed}
    AddSubTreeLevel($level$, $nFailed$, $BFHist$, 1, $rootC$);
end for
```

Algorithm 2 AddSubTreeLevel(*level*, *nFailed*, *BFHist*, *subTreeRate*, *rootC*)

```
1: $nextLevelPossibilities = \bigtimes_{i=1}^{|level|} \mathcal{P}(\Gamma_{level[i]})$;
   {Builds set of all possible nodes in next level as Cartesian product of powersets of Γ's}
2: for oneNextLevelPossibility ∈ nextLevelPossibilities do
3:     addedChildFlag = False;
4:     for parentC ∈ level do
5:         for childC ∈ $\Gamma_{parentC}$ do
6:             if childC ∈ oneNextLevelPossibility then
7:                 if nFailed[childC] == Redundancy(childC) then
8:                     goto line 3; {invalid subtree, requires more comps than available in system}
9:                 end if
10:                addedChildFlag = True;
11:                nFailed[childC] = nFailed[childC] + 1;
12:                add @ to BFHist[childC]; {signifies one component of type childC has failed}
13:                $subTreeRate = subTreeRate * \phi_{parentC,\, childC}$;
                   {update rate with $\phi$}
14:            else
15:                add parentC to BFHist[childC]; {signifies one component of type childC has not failed, but was present in $\Gamma_{parentC}$}
16:            end if
17:        end for
18:    end for
19:
20:    if addedChildFlag then
21:        AddSubTreeLevel(oneNextLevelPossibility, nFailed, BFHist, subTreeRate, rootC);
           {subTree can be grown further}
22:    else
23:        ComputeTreeRates(nFailed, BFHist, subTreeRate, rootC);
           {current subTree is completed because it cannot be grown further}
24:    end if
25: end for
```

Algorithm 3 ComputeTreeRates($\mathit{nFailed}$, BFHist, $\mathit{subTreeRate}$, rootC)

```
for $x \in Q$ do
    $e$ = Environment($x$);
    Initialize $y$ as a state with no components failed and environment $e$;
    for $\mathit{comp} \in \mathit{compSet}$ do
        $y[\mathit{comp}] = x[\mathit{comp}] + \mathit{nFailed}[\mathit{comp}]$;
    end for
    if $y$ is not a valid state then
        continue;
    end if
    $\mathit{rootFailureRate} =$ (Redundancy($\mathit{rootC}$) $- x[\mathit{rootC}]) * \lambda_{\mathit{rootC},\, e}$;
    $\mathit{prodNotFailedProb} = 1$; {cumulative probability of comps that could
    have failed but did not}
    for $\mathit{comp} \in \{\mathit{compSet}\}$ do
        $\mathit{compsAvailable}$ = Redundancy($\mathit{comp}$) $- x[\mathit{comp}]$;
        for $\mathit{parentC} \in \mathit{BFHist}[\mathit{comp}]$ do
            if $\mathit{parentC}$ == @ then
                $\mathit{compsAvailable} = \mathit{compsAvailable} - 1$;
            else if $\mathit{compsAvailable} > 0$ then
                $\mathit{prodNotFailedProb} = \mathit{prodNotFailedProb} * (1 - \phi_{\mathit{parentC},\, \mathit{comp}})$;
            end if
        end for
    end for
    $Q(x, y) = Q(x, y) + \mathit{rootFailureRate} * \mathit{subTreeRate} * \mathit{prodNotFailedProb}$;
end for
```