Algorithm 1 Matrix-based Deadlock-resolving Verification Algorithm

Input: Expanded Vertex Simplification R_i of RDLT R, Contraction Path Algorithm, Find Escape Contraction Path Algorithm
Output: Boolean; $True$, otherwise $False$
Matrices: Adjacency Matrix RV_adj, Constraint matirx RV_C

$P \leftarrow ContractionPathGenerationAlgorithm(R_i)$
Deadlock Identification: Vertices not in P, but are adjacent to P, and are PODs
$dV \leftarrow \emptyset$ ▷ Initialize deadlock set
$unreachable\_vertices \leftarrow V_i/P$
for all vertex $x \in unreachable\_vertices$ **do**
  Let l be the index of vertex x in V_i
  $constraints\_stack \leftarrow \emptyset$
  $constraints\_counter \leftarrow 0$
  for all row k in RV_C **do**
    $c \leftarrow RV_C[k][l]$
    if $c \neq \emptyset \wedge c \notin \{\epsilon\} \cup constraints\_stack$ **then**
      $constraints\_stack \leftarrow constraints\_stack \cup \{c\}$
      $constraint\_counter \leftarrow constraint\_counter + 1$
    end if
  end for
  if $constraints\_counter \geq 2$ **then**
    $dV \leftarrow x_l \cup dV$
  end if
end for
Enumeration of Parents of Deadlocks: Reachable Parents of Identified Deadlocks
for all vertex $x \in dV$ **do**
  $parents(x) \leftarrow \emptyset$
  Let l be the index of vertex x in V_i
  for all row k in RV_{adj} **do**
    if $RV_{adj}[k][l] = 1 \wedge x_k \in P$ **then** ▷ The parent vertex needs to be reachable, hence included in the contraction path
      $parent(x) \leftarrow x_k \cup parent(x)$
    end if
  end for
  for all parent vertex $w \in parent(x)$ **do**
    $escapePathExists \leftarrow findEscapePath(w, f', P, RV)$
    if $escapePathExists = false$ **then**
      return $False$
    end if
  end for
end for
return $True$

Algorithm 2 Matrix-based Contraction Path Generation

Given RDLT R; Pre-processing Steps:
Input: Expanded vertex simplification R_i of RDLT R
Output: Contraction Path P
Matrices: RV_{adj}^t and RV_{C}^t

Initialize C-Attribute Matrix RV_{C}^0 of R_i
Let $s' \in V_i$ be the source and $f' \in V_i$ be the sink
Let $x = s'$
Initialize $P = \{x\}$
Let $t = 1$
while $\exists y \in V_i \mid RV_{\mathrm{adj}}^{t-1}(x, y) \geq 1$ **do**
  $\mathcal{Y} \leftarrow \{y \in V_i \mid RV_{\mathrm{adj}}^{t-1}(x, y) \geq 1\}$
  Select any $y \in \mathcal{Y}$
  Let LHS $= RV_C^{t-1}(x, y) \cup \{\epsilon\}$
  $\mathcal{U} \leftarrow \{u \in V_i \mid u \neq x \wedge (RV_{\mathrm{adj}}^{t-1}(u, y) \geq 1)\}$
  Let RHS $= \bigcup_{u \in \mathcal{U}} RV_C^{t-1}(u, y)$
  if LHS $\supseteq$ RHS **then**
    Update $RV_C^{t-1}(u, y) = \epsilon, \forall (u, y) \in E_i, u \neq x$
    for all $u \in \mathcal{U}$ **do**
      $RV_C^{t-1}(u, y) = \epsilon$
    end for
    Let $z = x \wedge y$
    Let $z = xy =$ Matrix Addition of rows (columns) x and y in RV_{adj}^{t-1}
    for all $w \in V_i$ **do**
      RowMerge_Adj: $RV_{\mathrm{adj}}^t(z, w) = RV_{\mathrm{adj}}^{t-1}(x, w) + RV_{\mathrm{adj}}^{t-1}(y, w)$
      ColMerge_Adj: $RV_{\mathrm{adj}}^t(w, z) = RV_{\mathrm{adj}}^{t-1}(w, x) + RV_{\mathrm{adj}}^{t-1}(w, y)$
    end for
    Let $z = xy =$ Element-wise Set Union of rows (columns) x and y in RV_C^{t-1}
    for all $w \in V_i$ **do**
      RowMerge_C: $RV_C^t(z, w) = RV_C^{t-1}(x, w) \cup RV_C^{t-1}(y, w)$
      ColMerge_C: $RV_C^t(w, z) = RV_C^{t-1}(w, x) \cup RV_C^{t-1}(w, y)$
    end for
    $V_i = (V_i \setminus \{x, y\}) \cup \{z\}$
    Create RV_{adj}^t as an $m \times m$ matrix where $m = n - t$ and as the submatrix of RV_{adj}^{t-1} with rows and columns indexed by updated vertex set V_i of R_i
    Create RV_C^t as an $m \times m$ matrix where $m = n - t$ and as the submatrix of RV_C^{t-1} with rows and columns indexed by updated vertex set V_i of R_i
    Let $x = z$
    Let $P = P \cup \{y\}$
    Let $t = t + 1$
  end if
end while
return P

Algorithm 3 **Matrix-Based Finding Escape Contraction Path Algorithm**

Given: RDLT R_i; Preprocessing
Input: Parent Vertex w, Sink f', Contraction Path P, Adjacency Matrix of R_i RV_{adj}
Output: Boolean, $True$, otherwise $False$

Initialize $Visited[0...n-1] \leftarrow false$ ▷ for each vertex in V_i
Initialize $Q \leftarrow$ empty queue
$Q.queue(w)$
$Visited[w] \leftarrow true$
while $Q \neq \emptyset$ **do**
    $current \leftarrow Q.dequeue()$
    Let k be the index of $current$ vertex in V_i
    if $current = f'$ **then**
        return $True$
        for all vertex x in P **do**
            Let l be the index of x in V_i
            if $RV_{adj}[k][l] = 1 \wedge Visted[l] = false$ **then**
                $Q.enqueue(x)$
                $Visited[l] \leftarrow true$
            end if
        end for
    end if
end while
return $False$