Verify Functional Correctness of Tarjan's **S**trongly **C**onnected **C**omponent Algorithm

周李韬

April 2021

What is SCC?

Definition 1.1 (Strong Connectivity). Let G be a directed graph. If for each pair of vertices v, w in G, v is reachable to w and w is reachable to v. (i.e. exists path $p_1 : v \rightarrow^* w$ and $p_2 : w \rightarrow^* v$. Then G is said to be strongly connected.

Definition 1.2 (Strongly Connected Components). Let G be a directed graph. Define an equivalence relation for two vertices if they are mutually reachable to each other. Let the distinct equivalence classes under this relation to be $\mathcal{V}_i, 1 \leq i \leq n$. Let $G_i = (\mathcal{V}_i, \mathcal{E}_i), where \mathcal{E}_i = \{(v, w) \in \mathcal{E} | v, w \in V_i\}$, then

a) Each G_i is strongly connected
b) No G_i is a proper subgraph of a strongly connected components of G

- An SCC Algorithm should map each vertex to a component C_i.
- The correctness of an SCC algorithm is based on two properties
  - Strongly Connected

    $\forall v, w \in C_i \Rightarrow v \leftrightarrow^* w$
  - Maximal

    $\forall v$, if $\forall w \in C_i \Rightarrow v \leftrightarrow^* w$,
    then $v \in C_i$

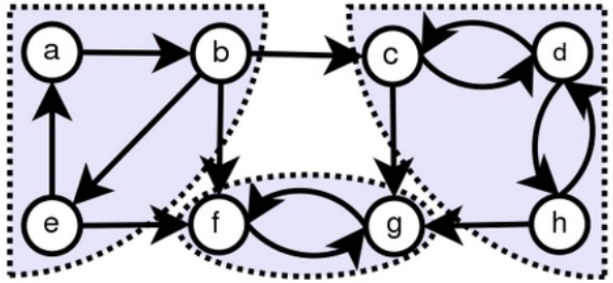
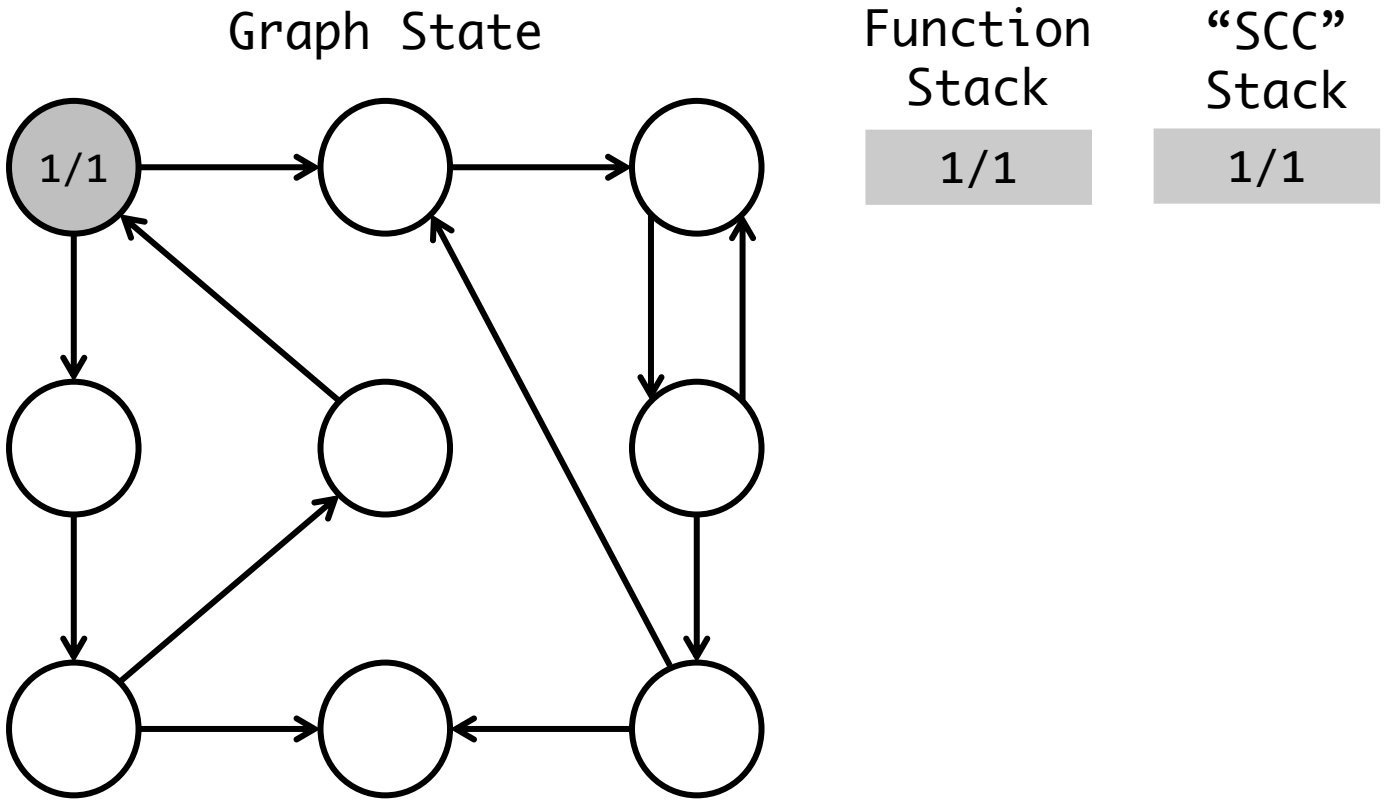
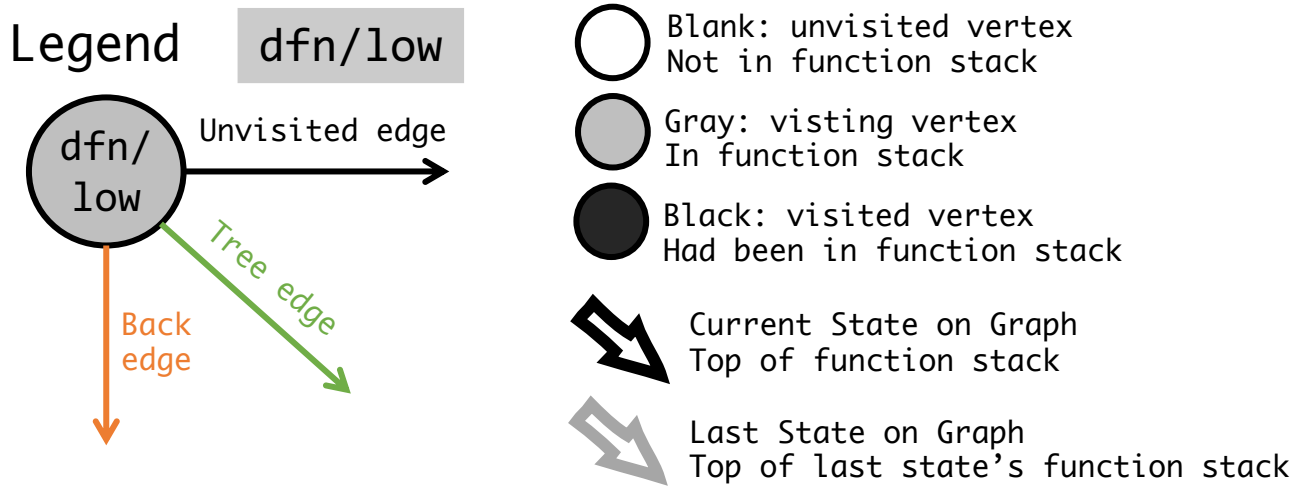


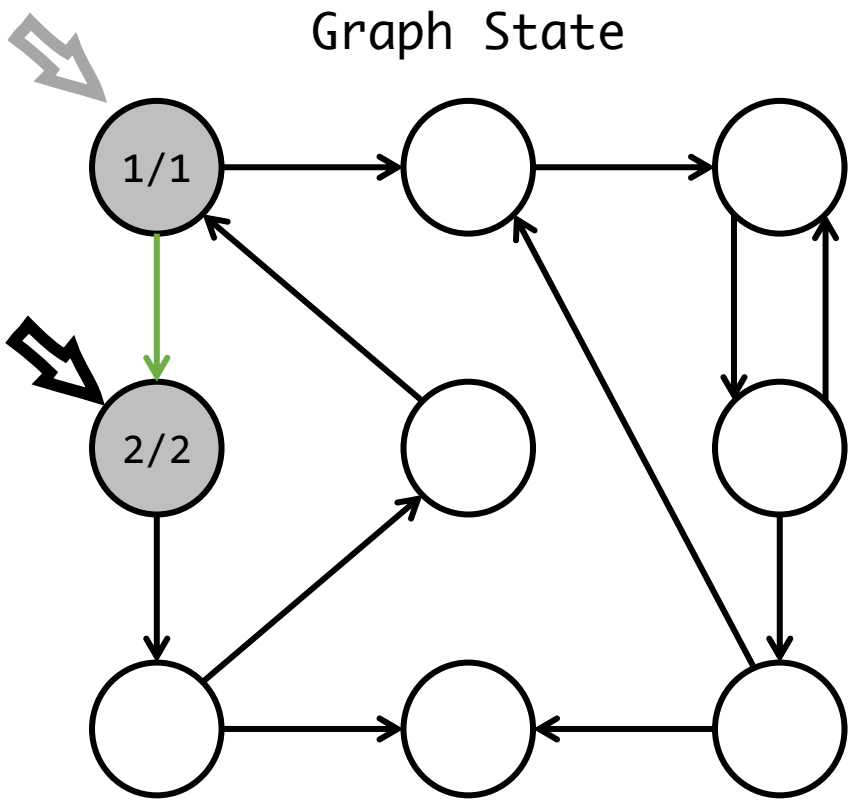
Figure 1. An example of Strongly Connected Components in a Directed Graph





How Tarjan's Algorithm works?

- Depth First Traverse the graph in one pass
- $Dfn: Vertex \rightarrow Nat$
  - visiting order of DFS
- $Low: Vertex \rightarrow Nat$
  - the smallest vertex's DFN which is reachable by passing at most one back edge
- $Scc\_Stack: list\ Vertex$
  - vertices which have been reached during the search but which have not yet been placed in a component are stored on a stack.

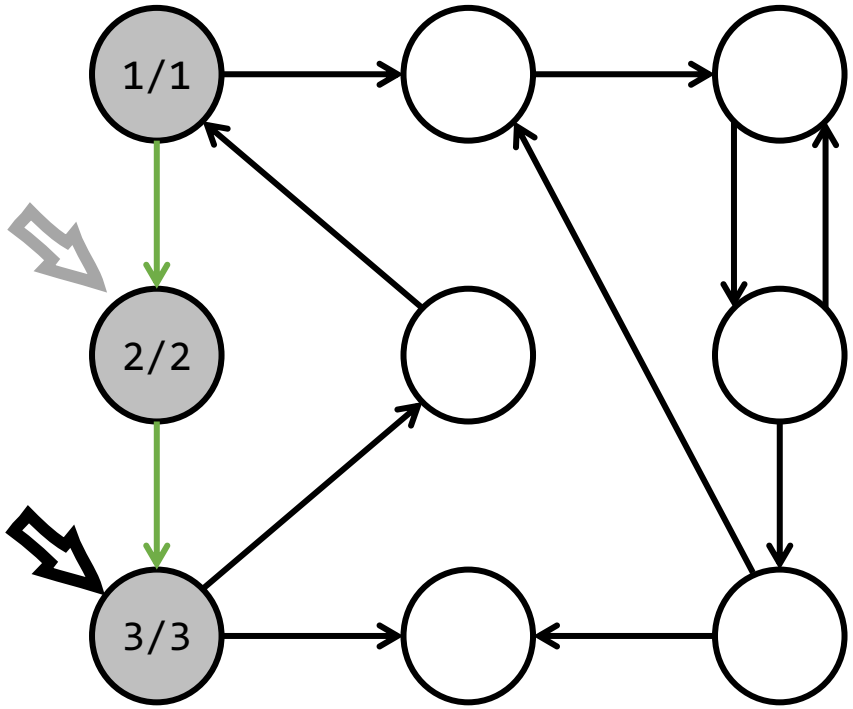


Function Stack	"SCC" Stack
1/1	1/1
2/2	2/2

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge

Graph State

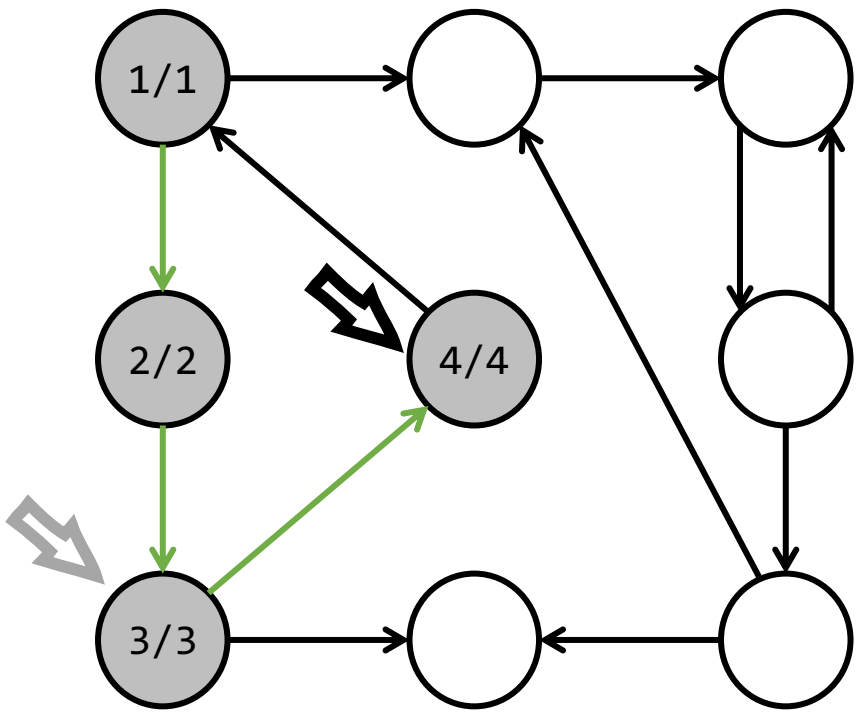


Function Stack	"SCC" Stack
1/1	1/1
2/2	2/2
3/3	3/3

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge

Graph State

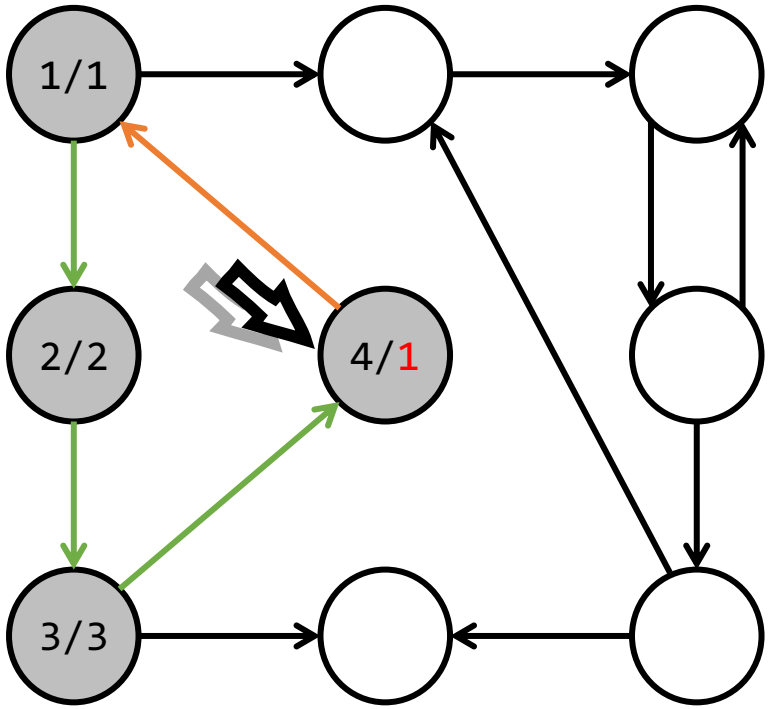


Function Stack	"SCC" Stack
1/1	1/1
2/2	2/2
3/3	3/3
4/4	4/4

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge

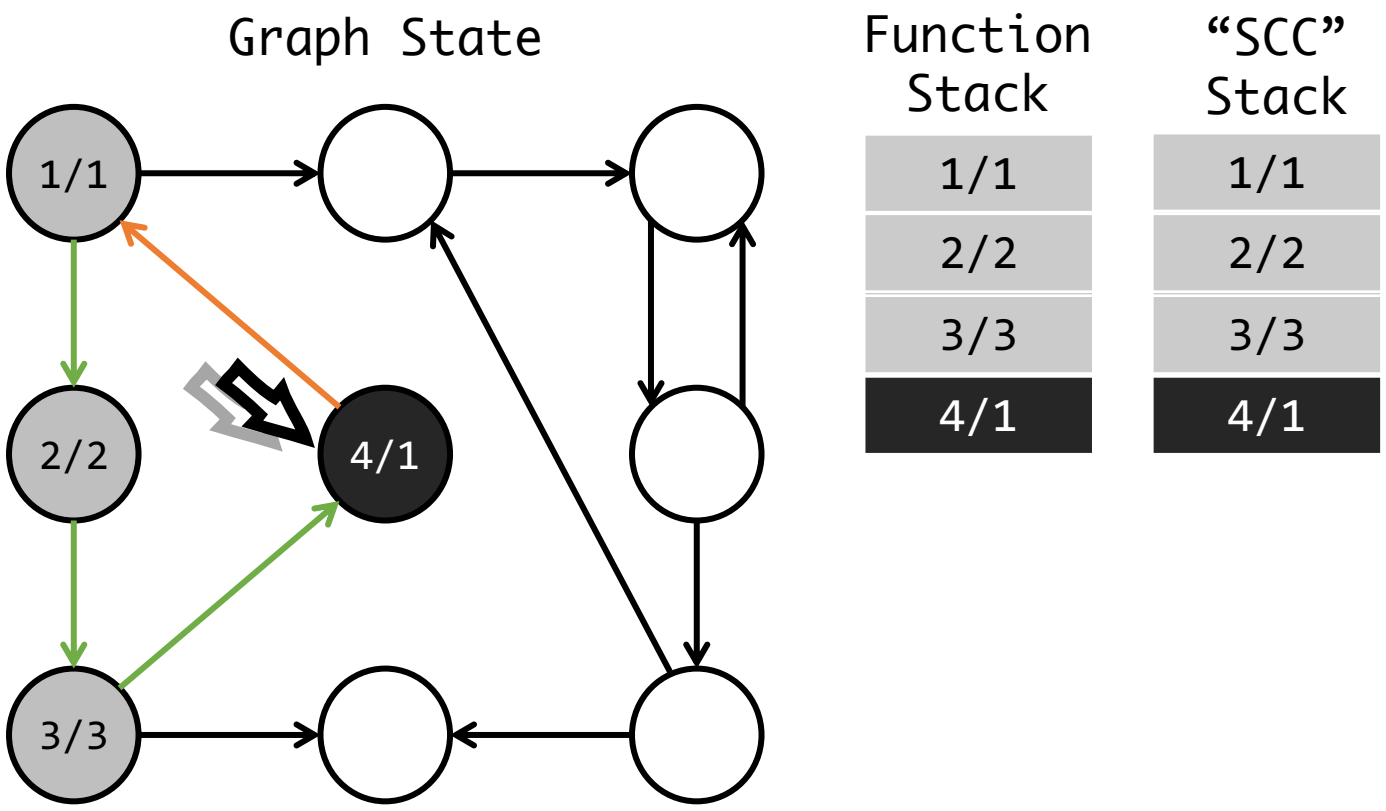
Graph State



Function Stack	"SCC" Stack
1/1	1/1
2/2	2/2
3/3	3/3
4/1	4/1

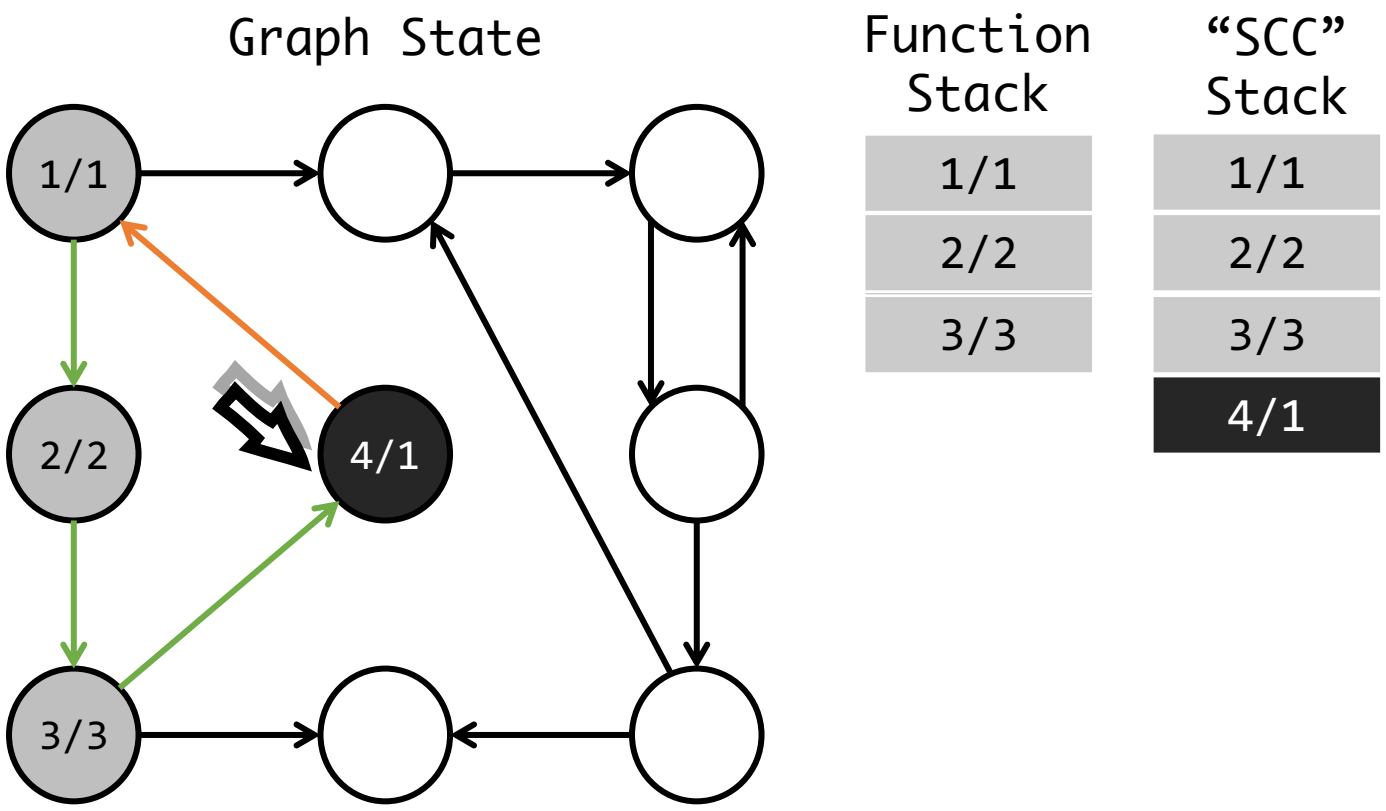
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- **Forward-Visited Step**
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge



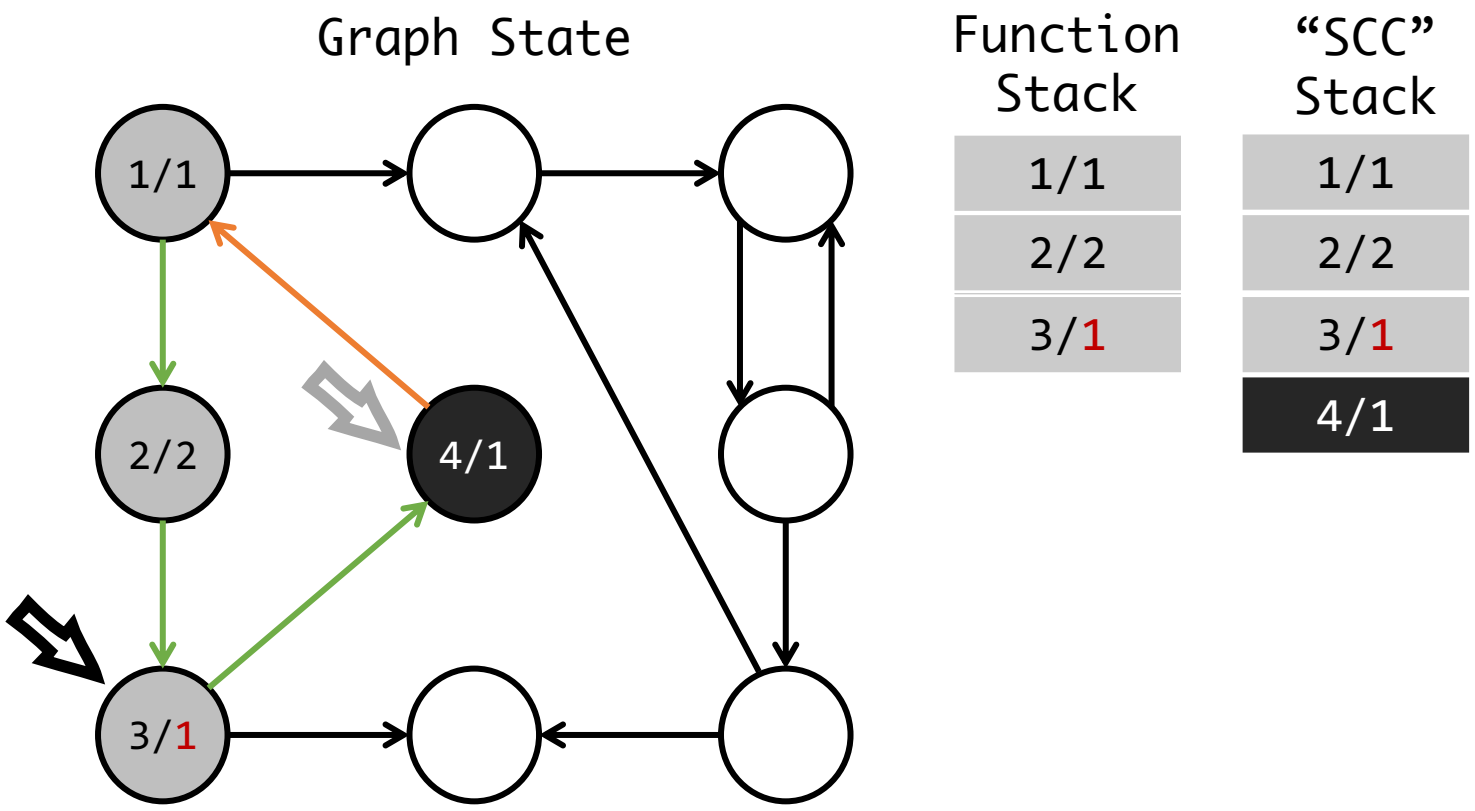
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**



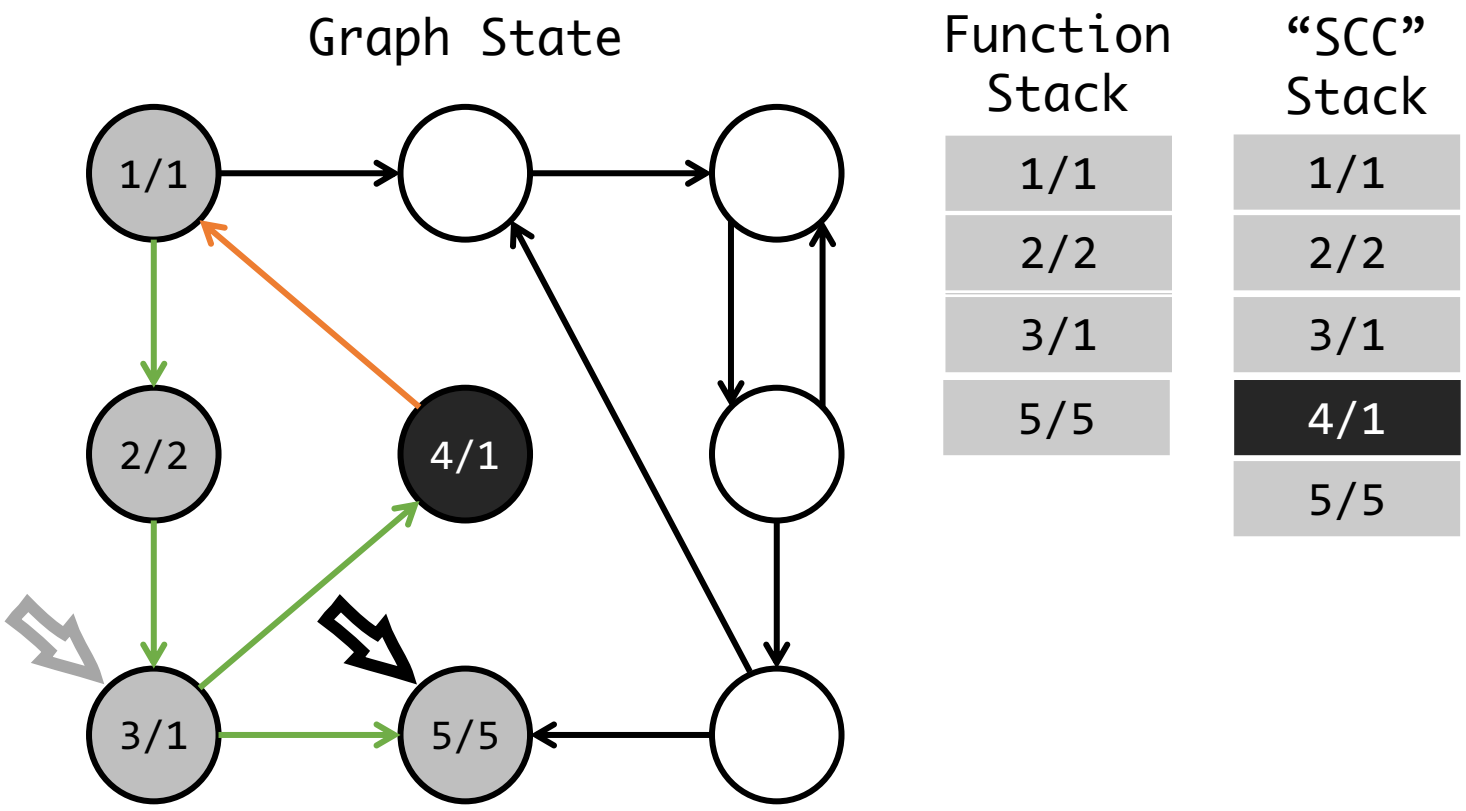
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**



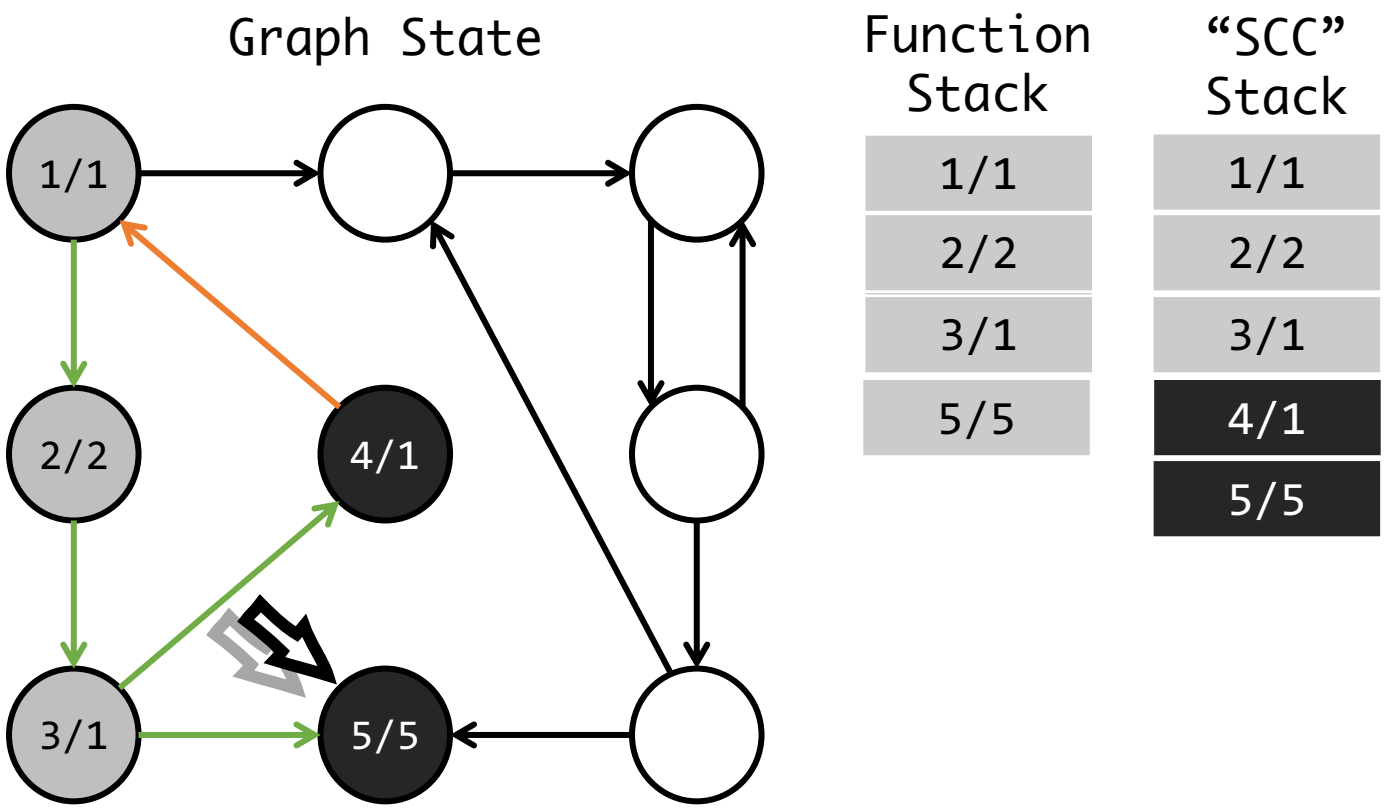
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



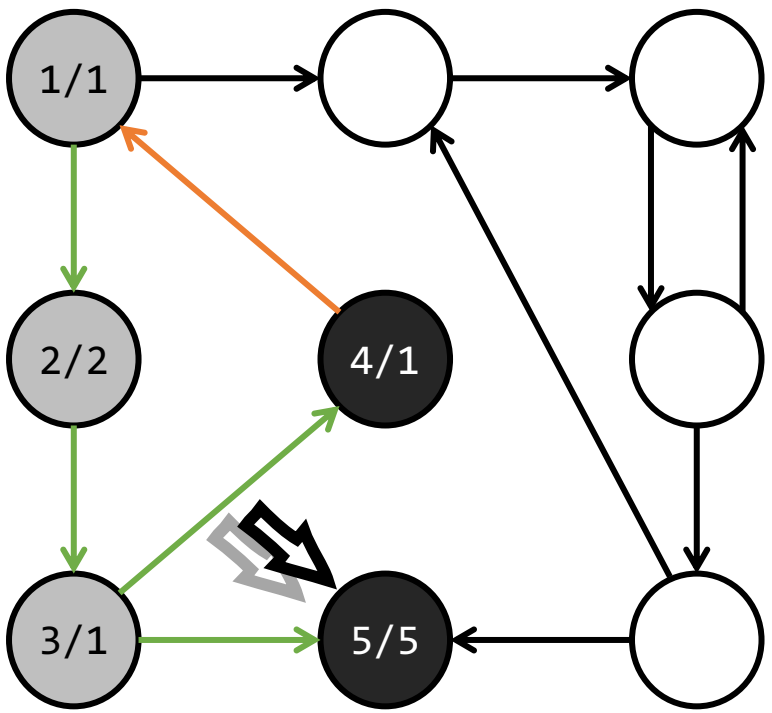
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

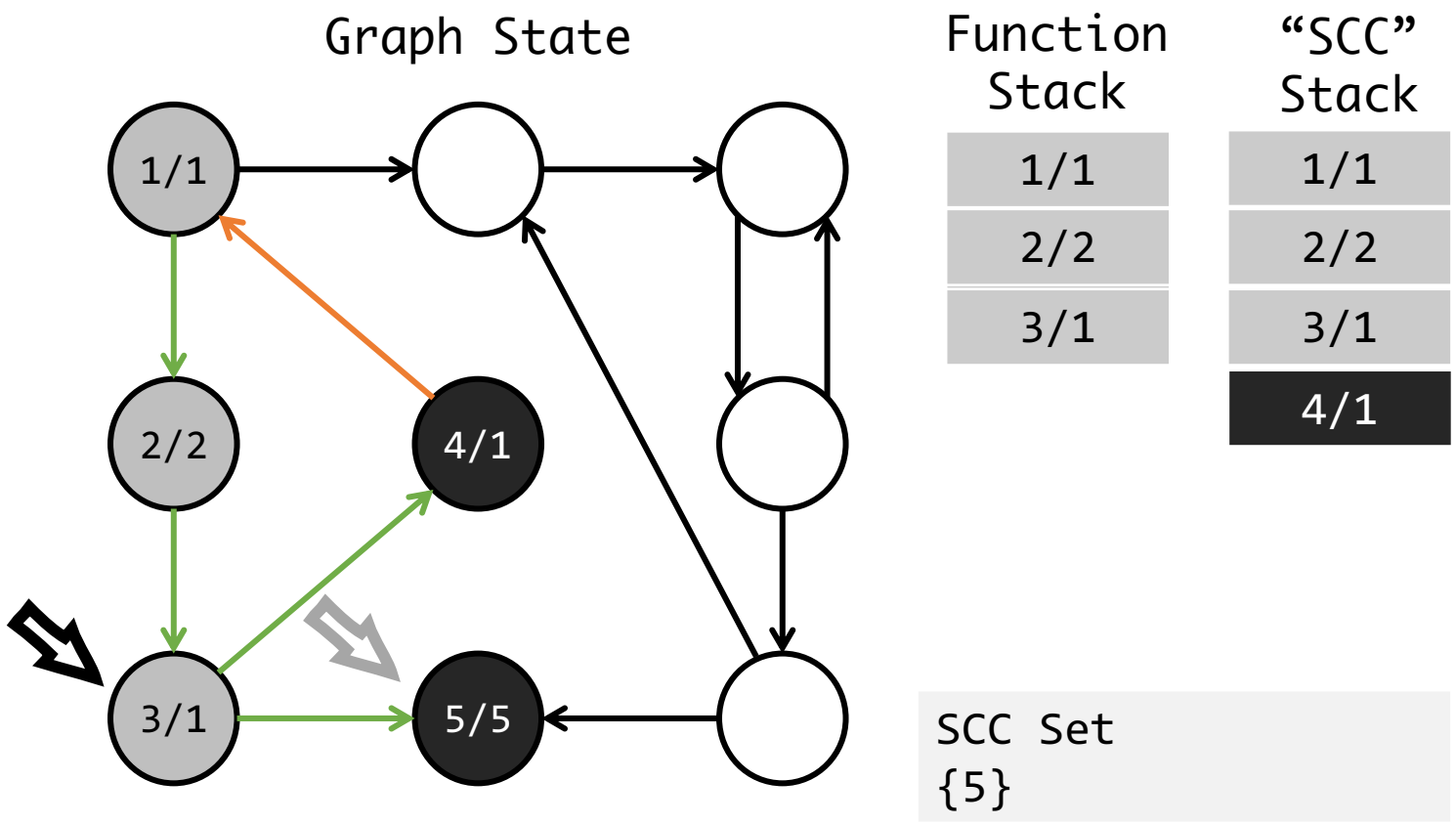


Function Stack	"SCC" Stack
1/1	1/1
2/2	2/2
3/1	3/1
5/5	4/1

SCC Set
{5}

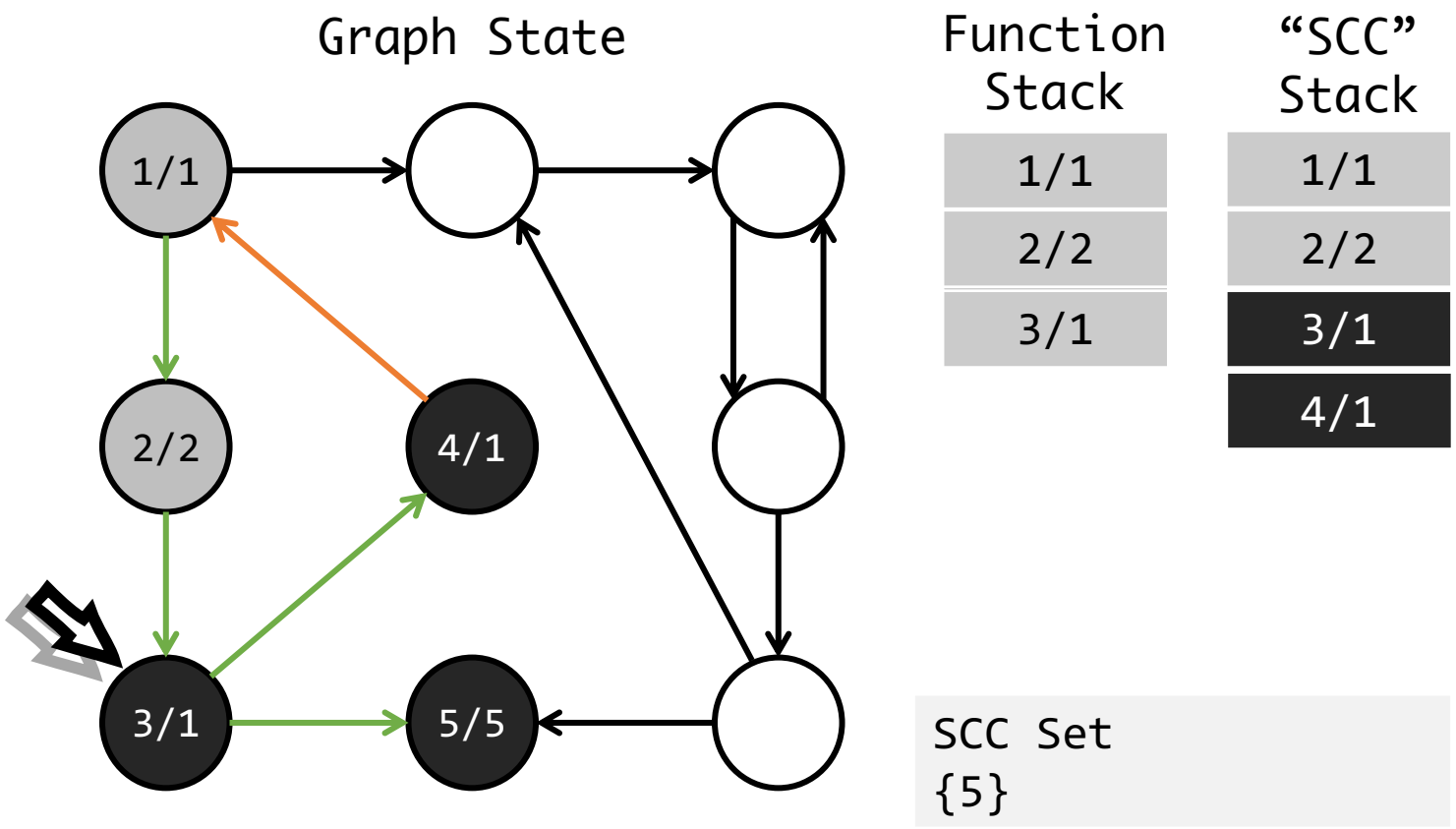
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



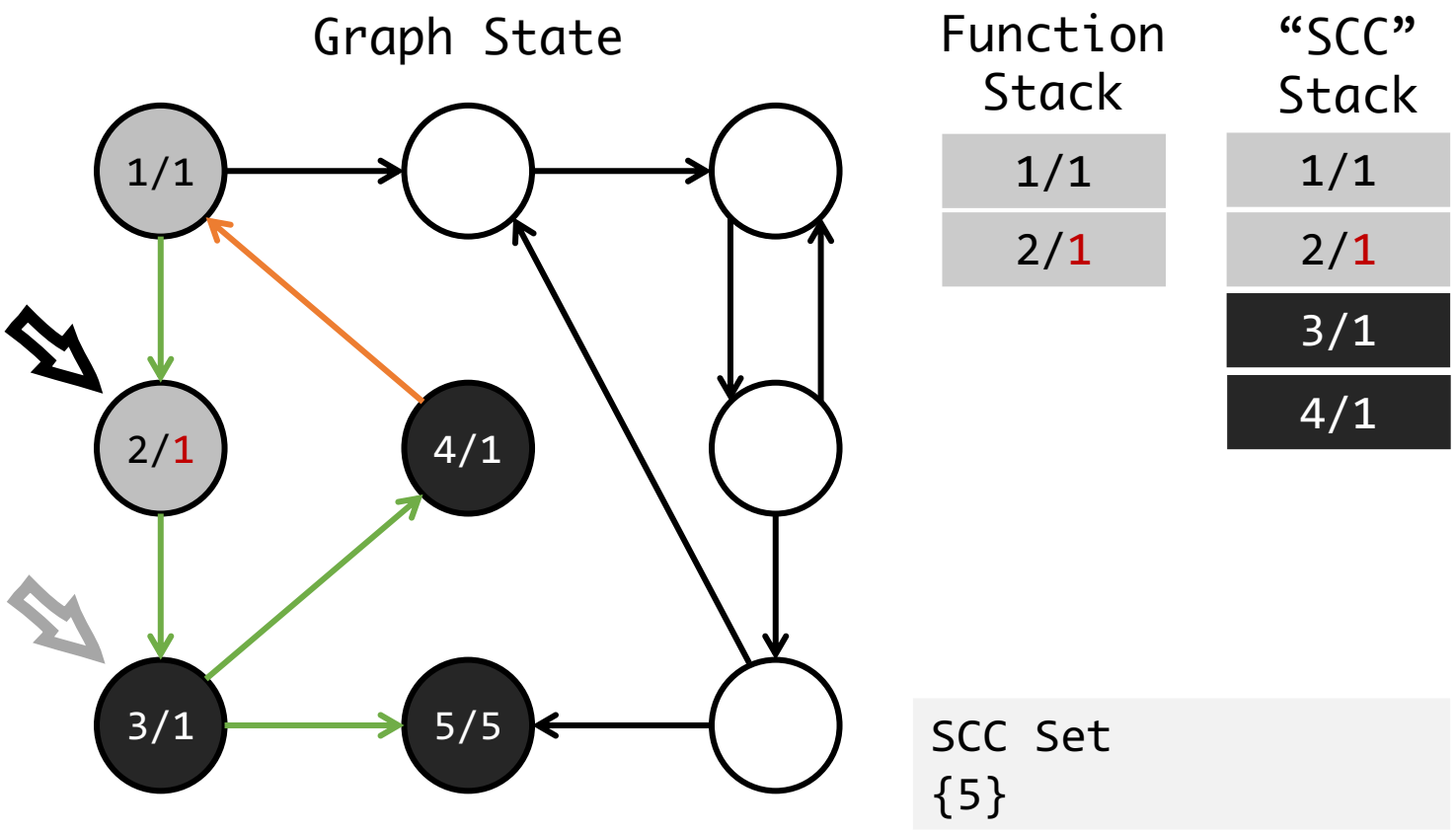
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



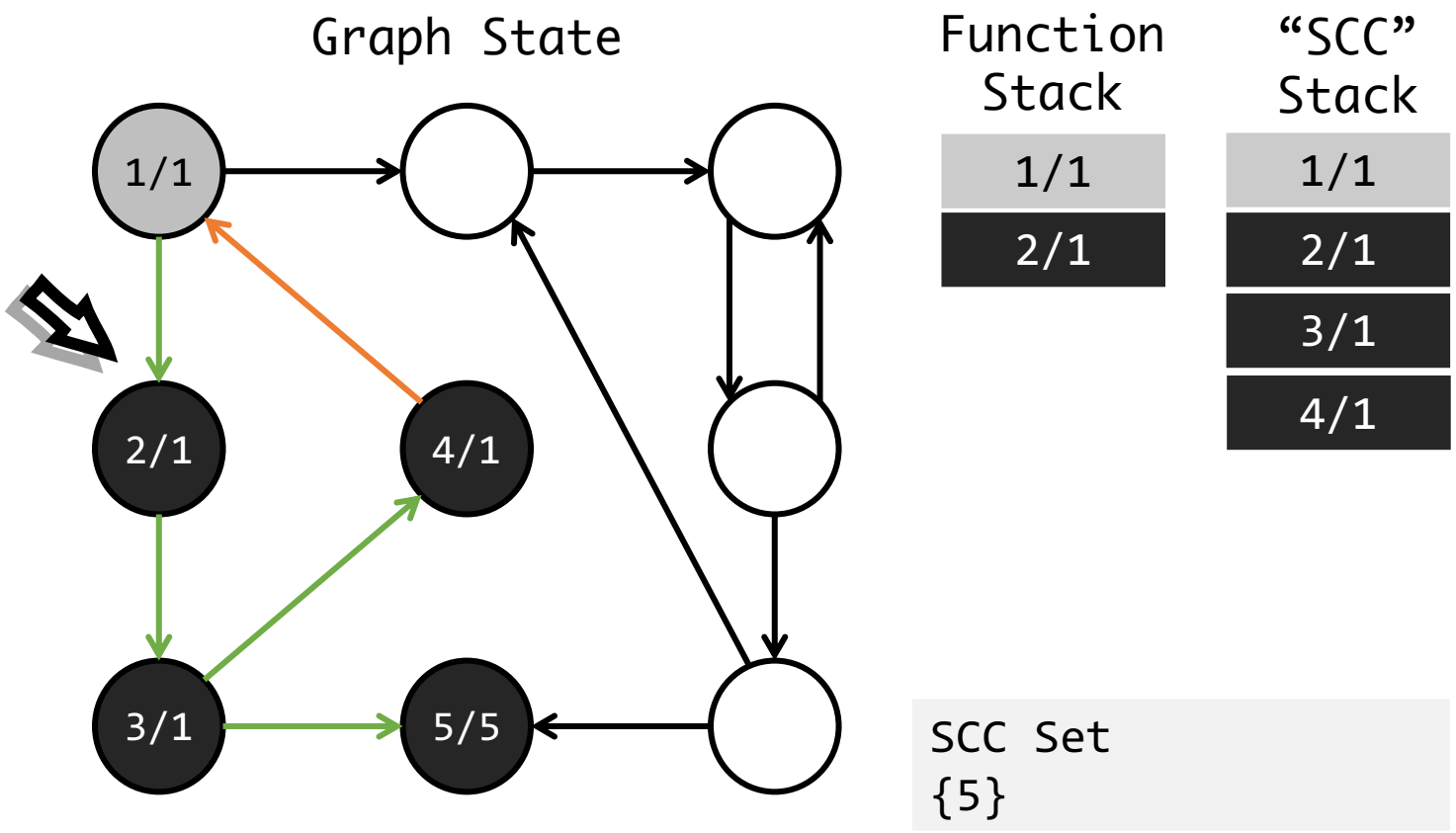
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



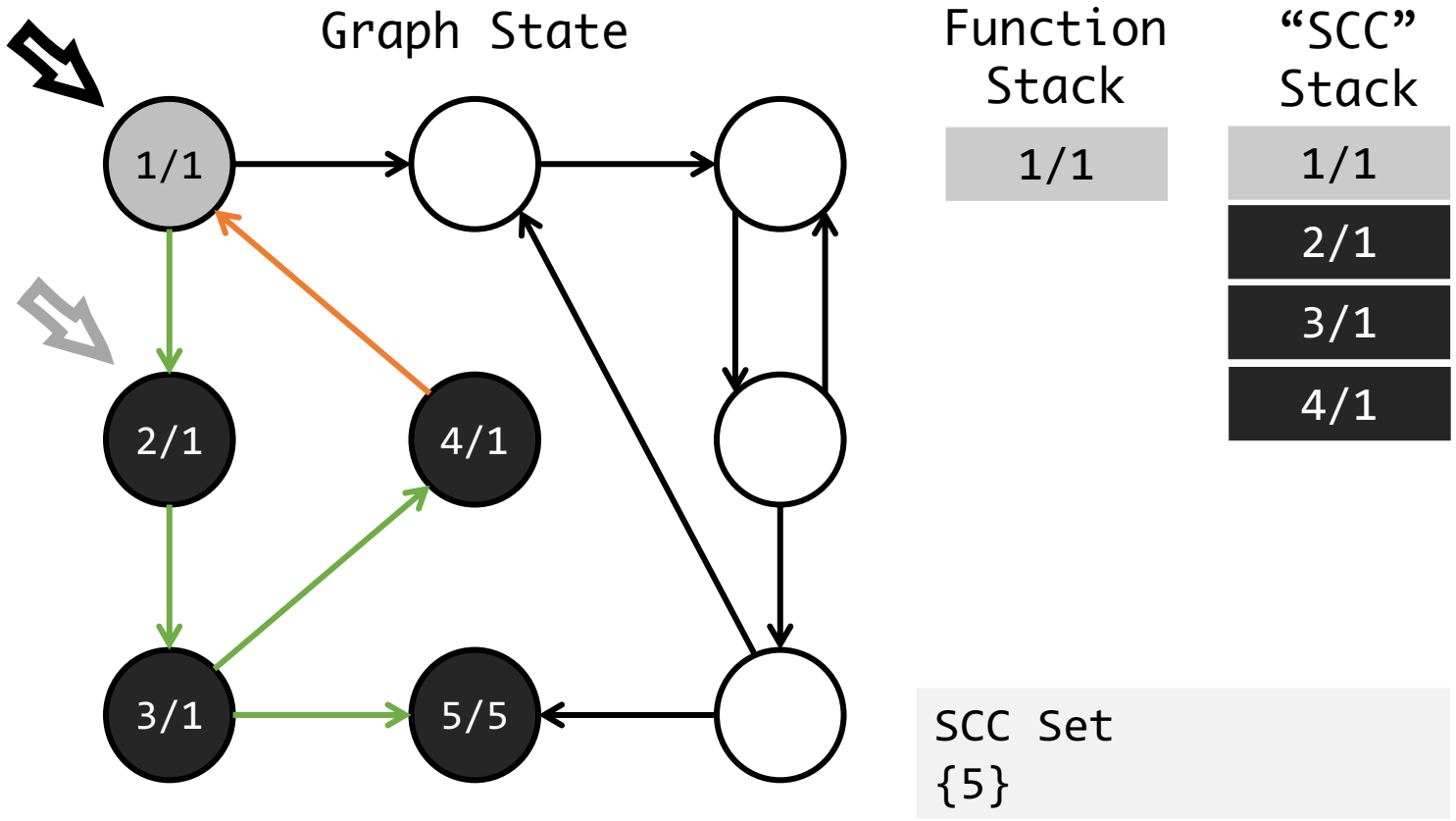
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



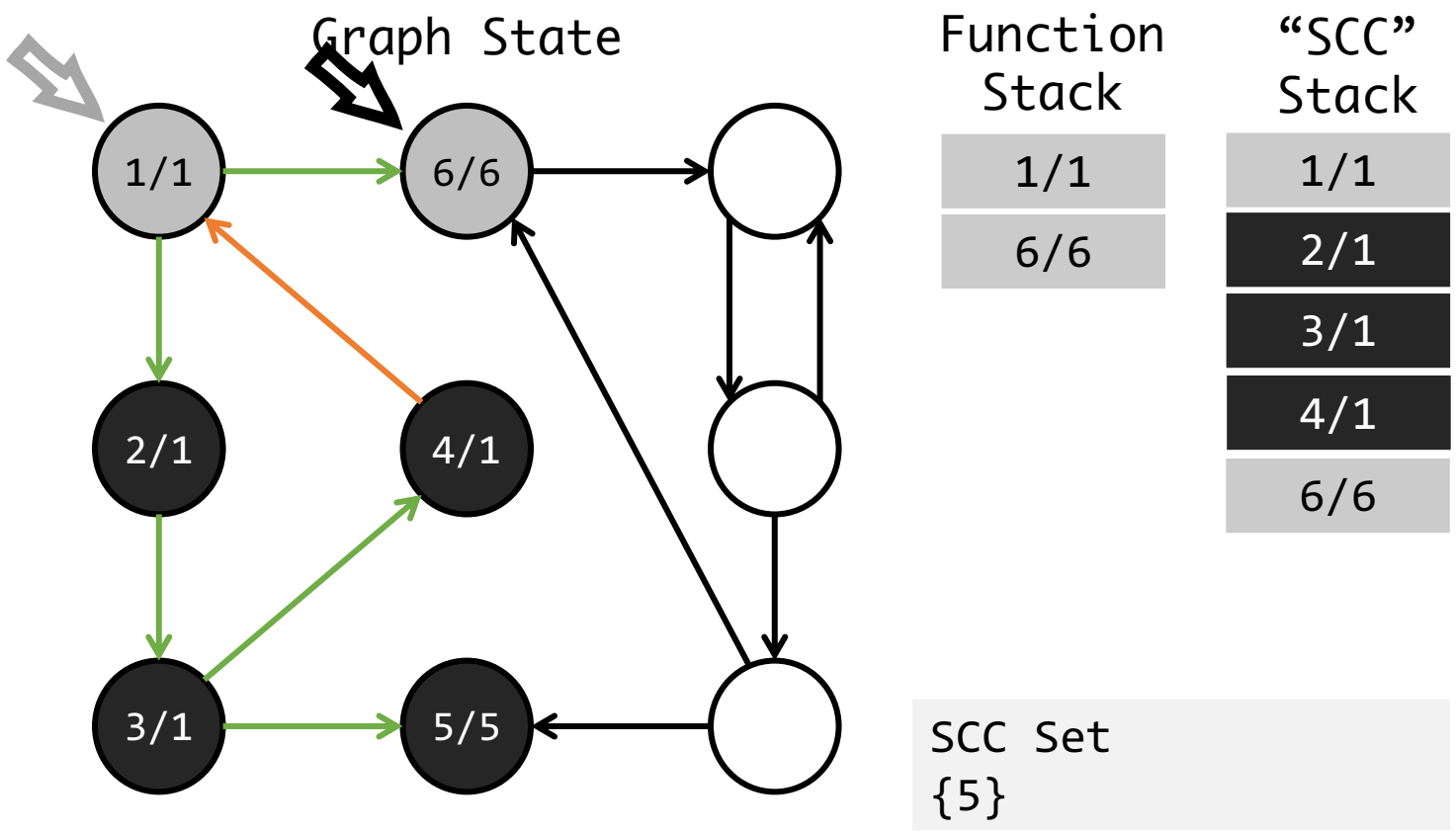
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



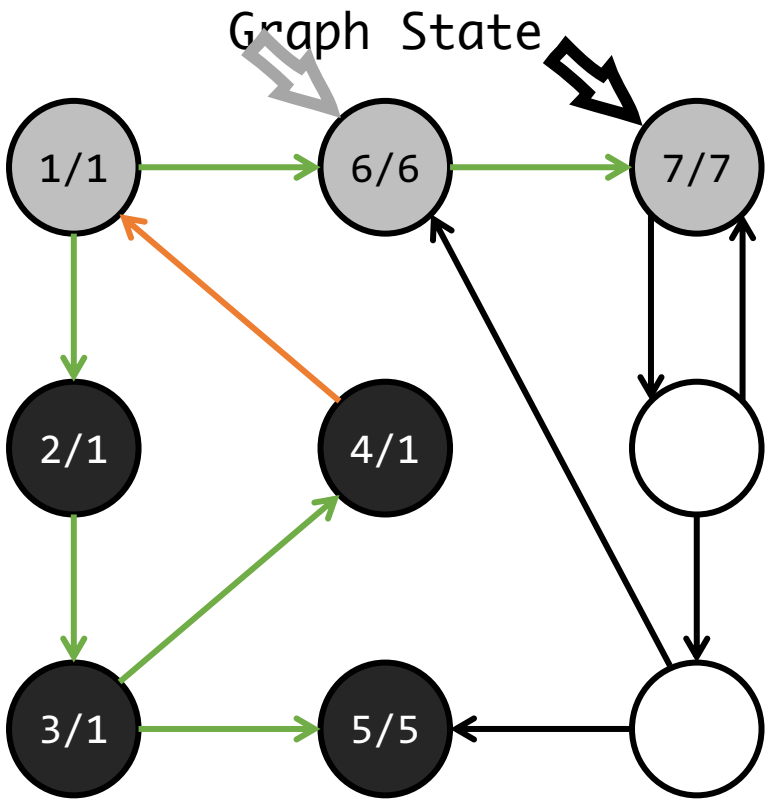
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



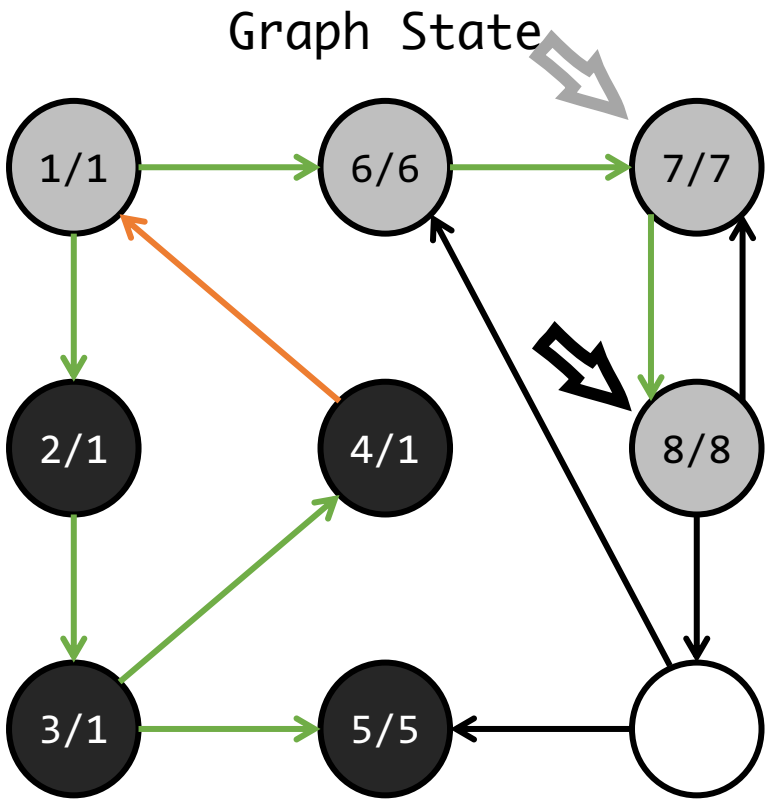
Function Stack
1/1
6/6
7/7

"SCC" Stack
1/1
2/1
3/1
4/1
6/6
7/7

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



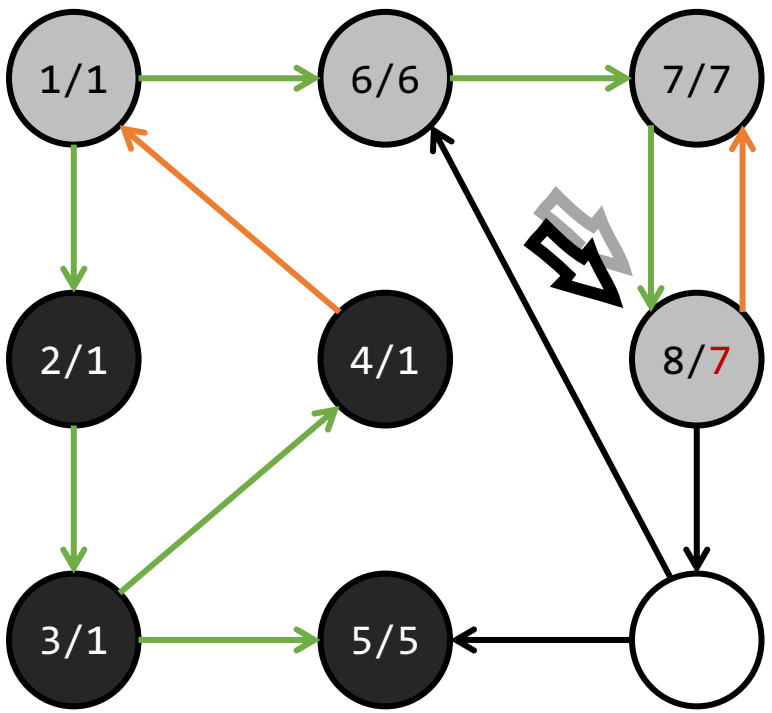
Function Stack
1/1
6/6
7/7
8/8

"SCC" Stack
1/1
2/1
3/1
4/1
6/6
7/7
8/8

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



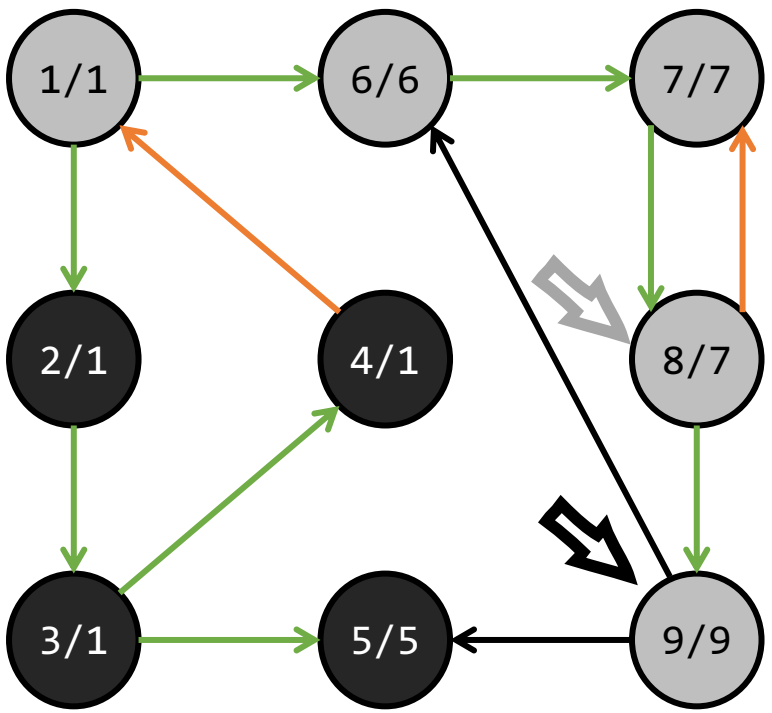
Function Stack
1/1
6/6
7/7
8/7

"SCC" Stack
1/1
2/1
3/1
4/1
6/6
7/7
8/7

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- **Forward-Visited Step**
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



Function Stack
1/1
6/6
7/7
8/7
9/9

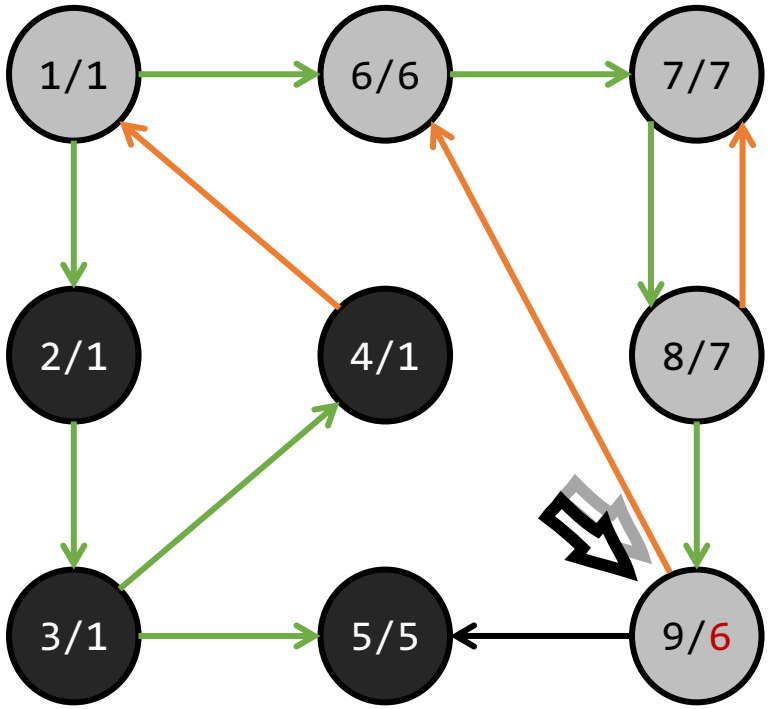
"SCC" Stack
1/1
2/1
3/1
4/1
6/6
7/7
8/7
9/9

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- **Forward Step**
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

Graph State

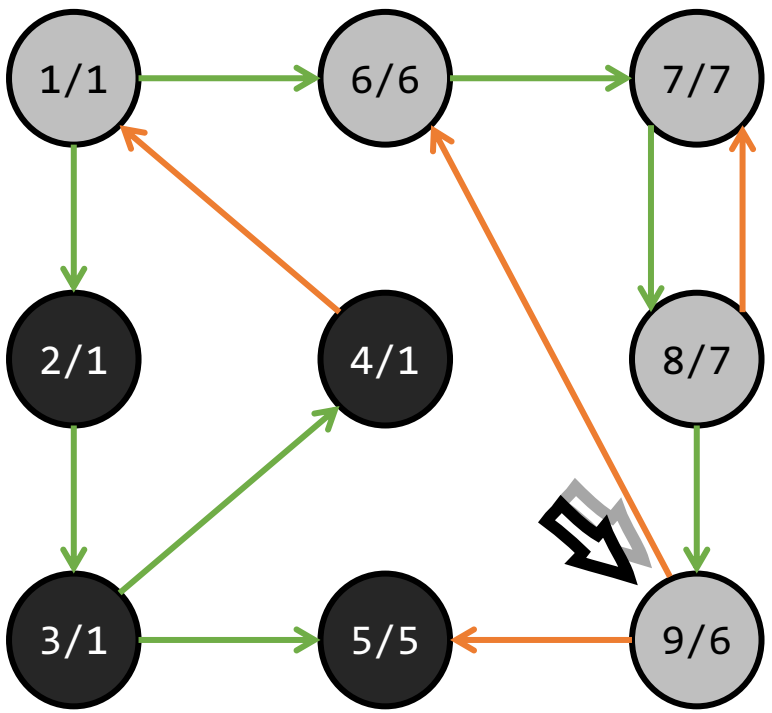


Function Stack	"SCC" Stack
1/1	1/1
6/6	2/1
7/7	3/1
8/7	4/1
9/6	6/6
	7/7
	8/7
	9/6

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- **Forward-Visited Step**
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

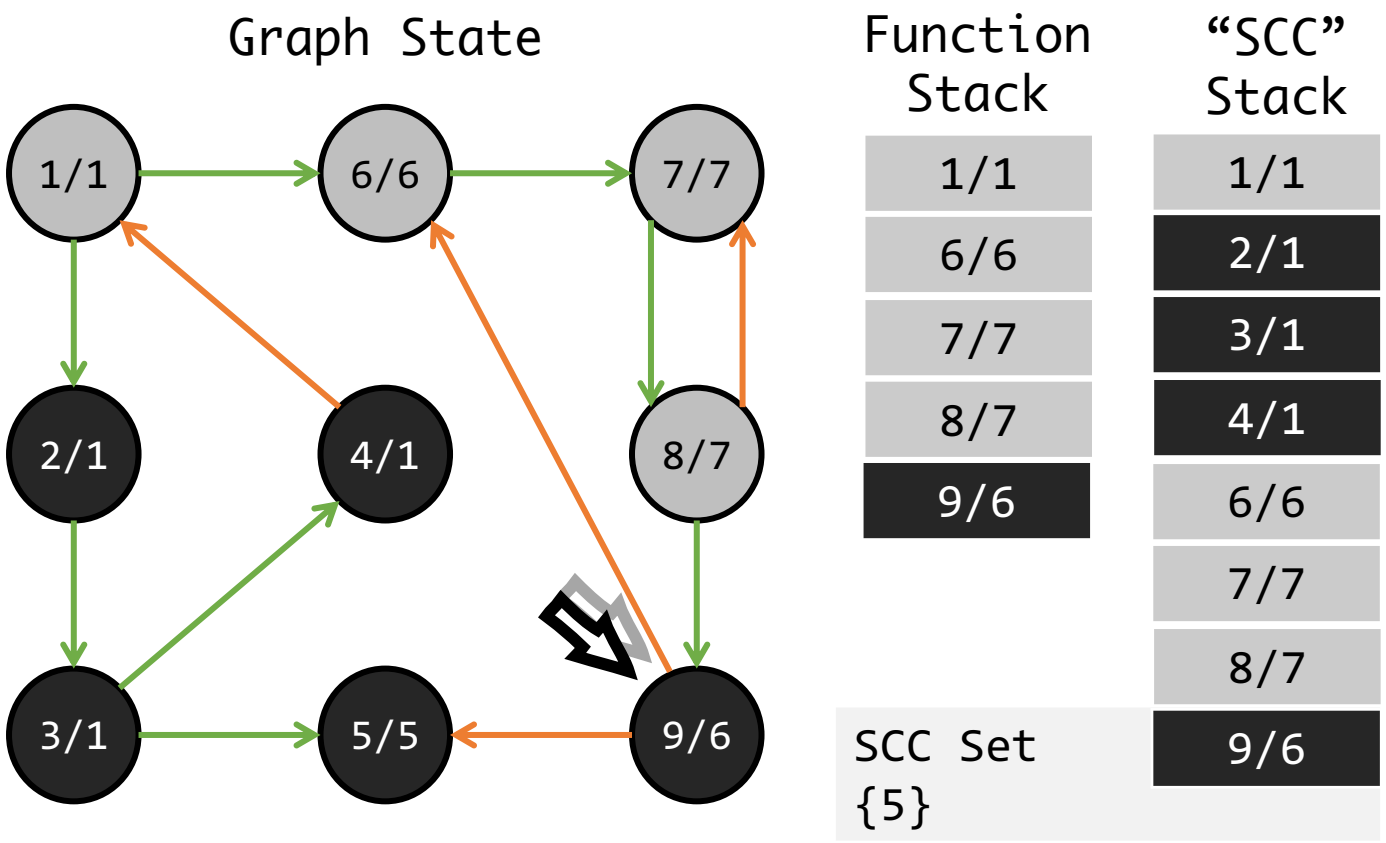


Function Stack	"SCC" Stack
1/1	1/1
6/6	2/1
7/7	3/1
8/7	4/1
9/6	6/6
	7/7
	8/7
	9/6

SCC Set
{5}

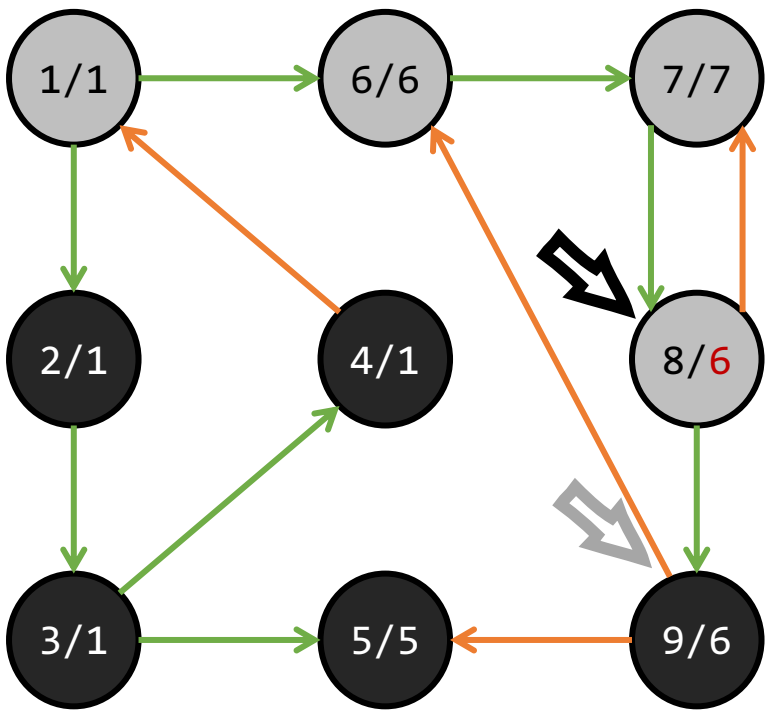
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



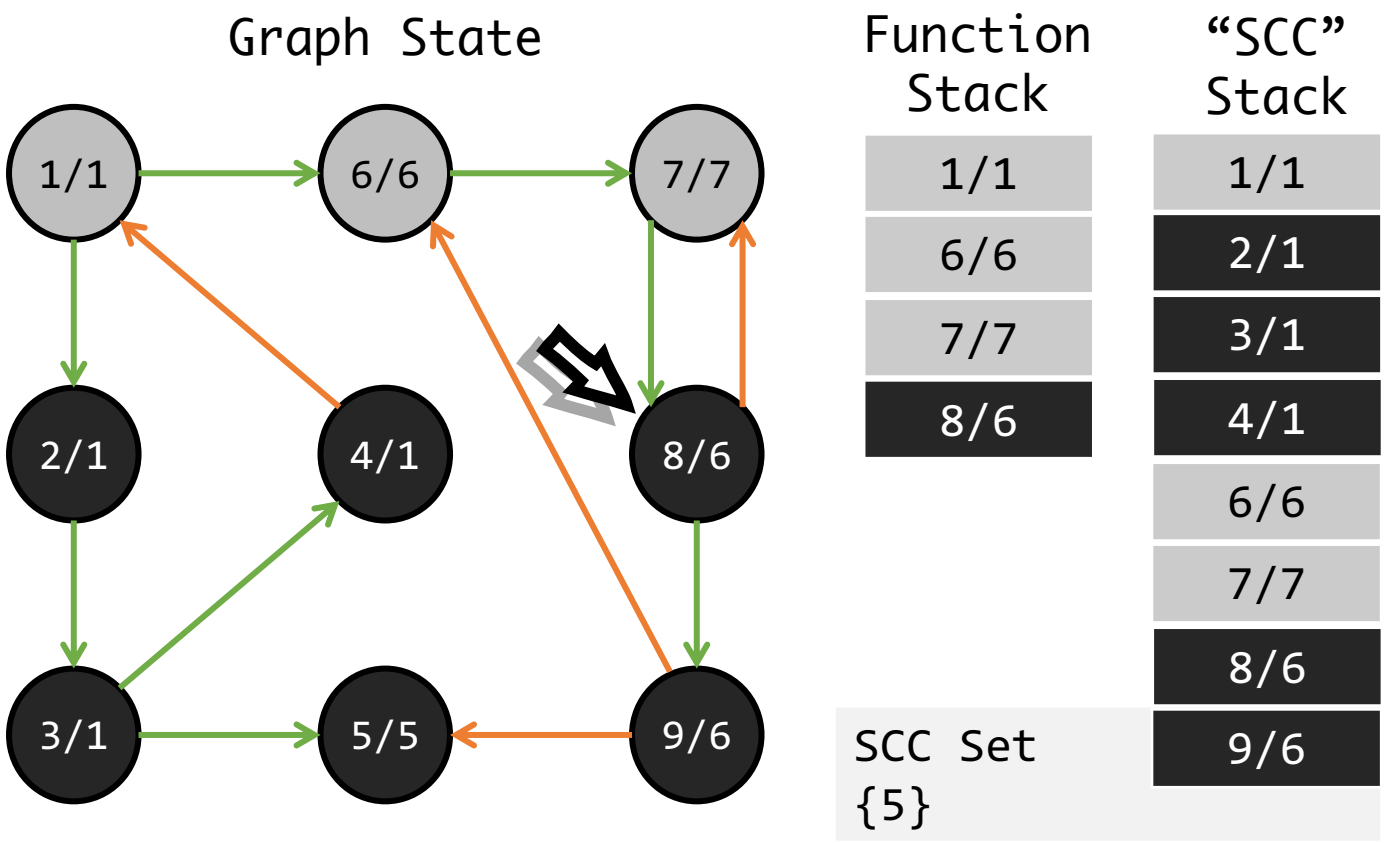
Function Stack
1/1
6/6
7/7
8/6

"SCC" Stack
1/1
2/1
3/1
4/1
6/6
7/7
8/6
9/6

SCC Set
{5}

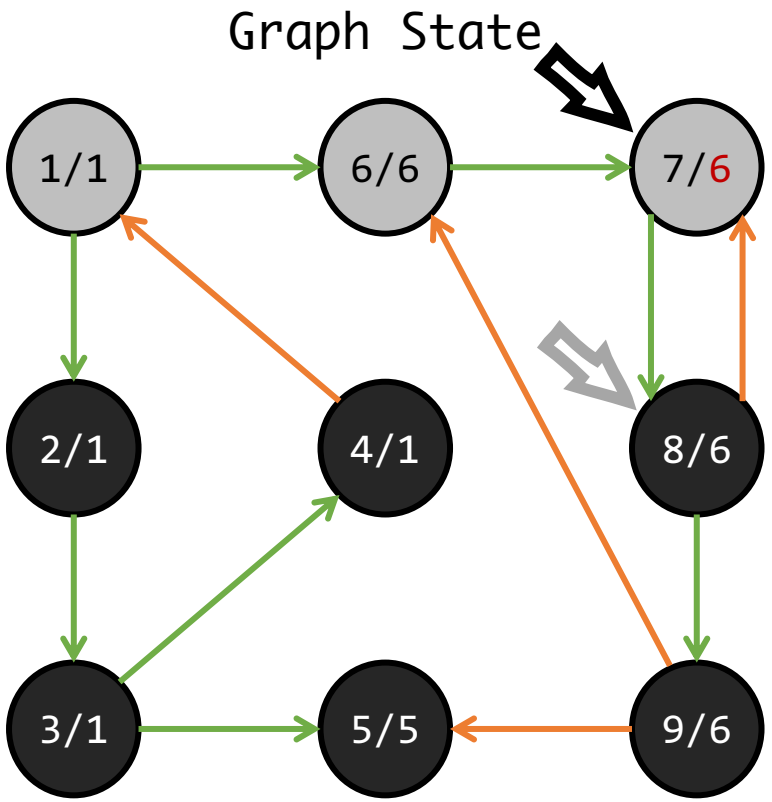
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

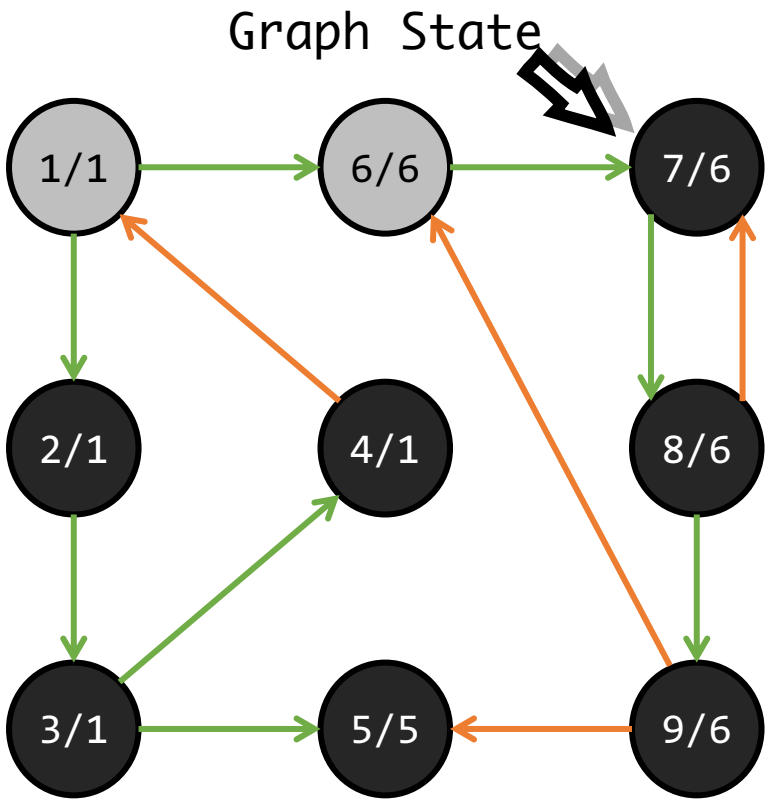


Function Stack	"SCC" Stack
1/1	1/1
6/6	2/1
7/6	3/1
	4/1
	6/6
	7/6
	8/6
	9/6

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

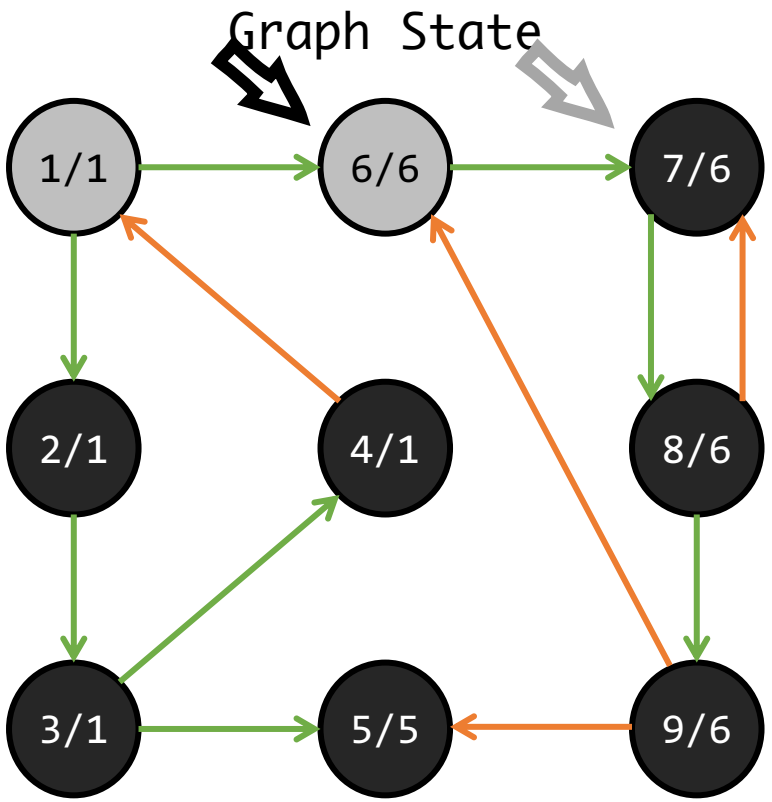


Function Stack	"SCC" Stack
1/1	1/1
6/6	2/1
7/6	3/1
	4/1
	6/6
	7/6
	8/6
	9/6

SCC Set
{5}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise **do nothing**
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

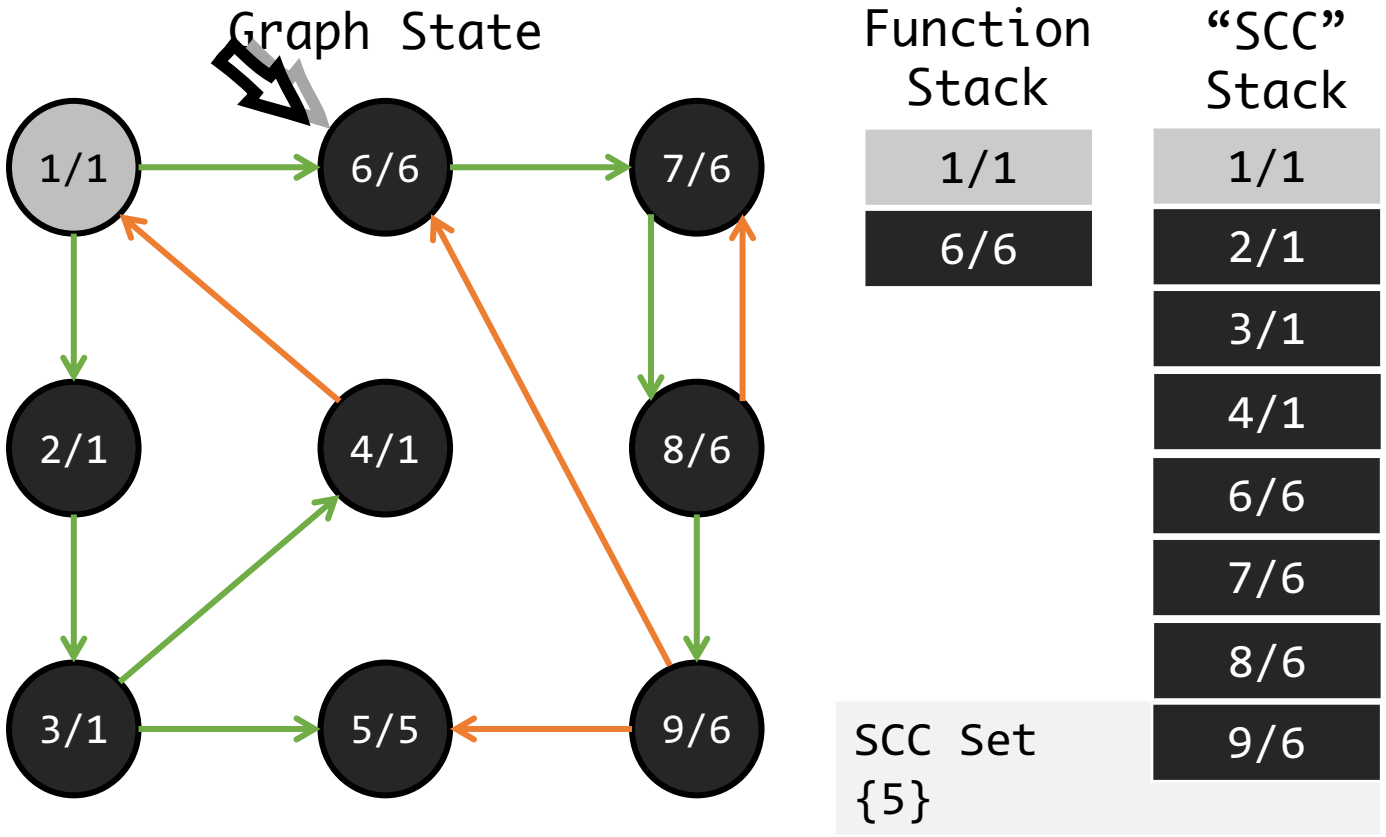


Function Stack	"SCC" Stack
1/1	1/1
6/6	2/1
	3/1
	4/1
	6/6
	7/6
	8/6
	9/6

SCC Set
{5}

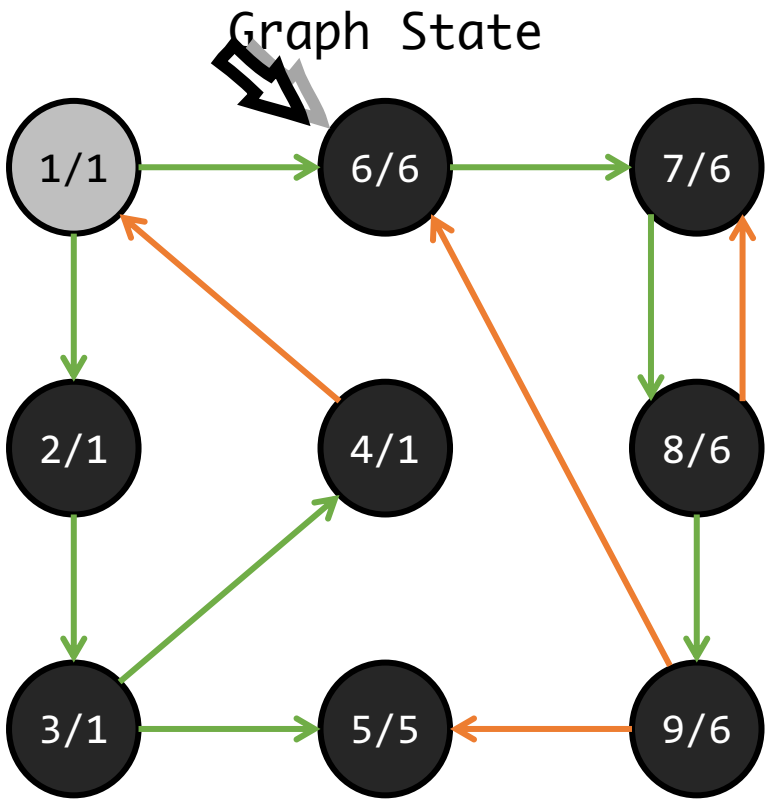
- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



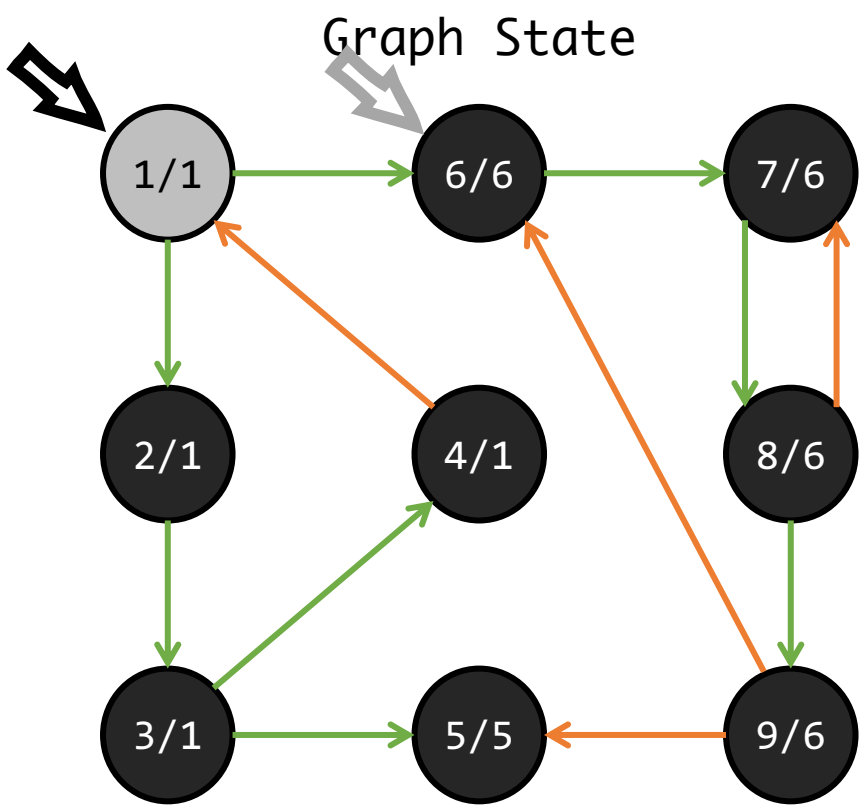
Function Stack
1/1
6/6

"SCC" Stack
1/1
2/1
3/1
4/1

SCC Set
{5},{6,7,8,9}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



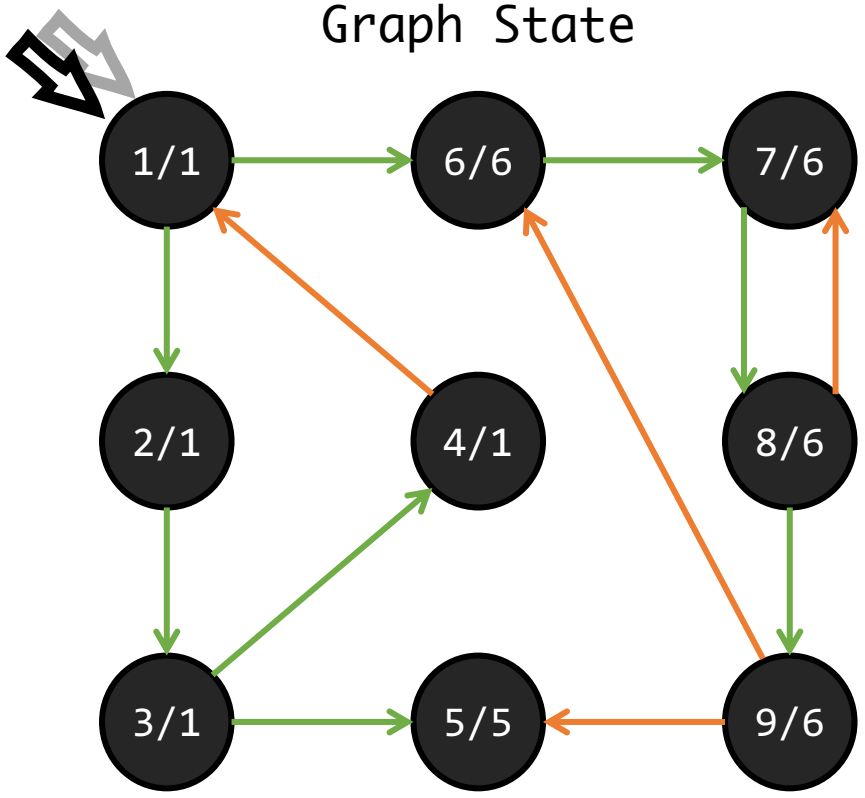
Function Stack
1/1

"SCC" Stack
1/1
2/1
3/1
4/1

SCC Set
{5},{6,7,8,9}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- DecidePop Step
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then pop, otherwise do nothing
- **Backward Step**
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



Function Stack

1/1

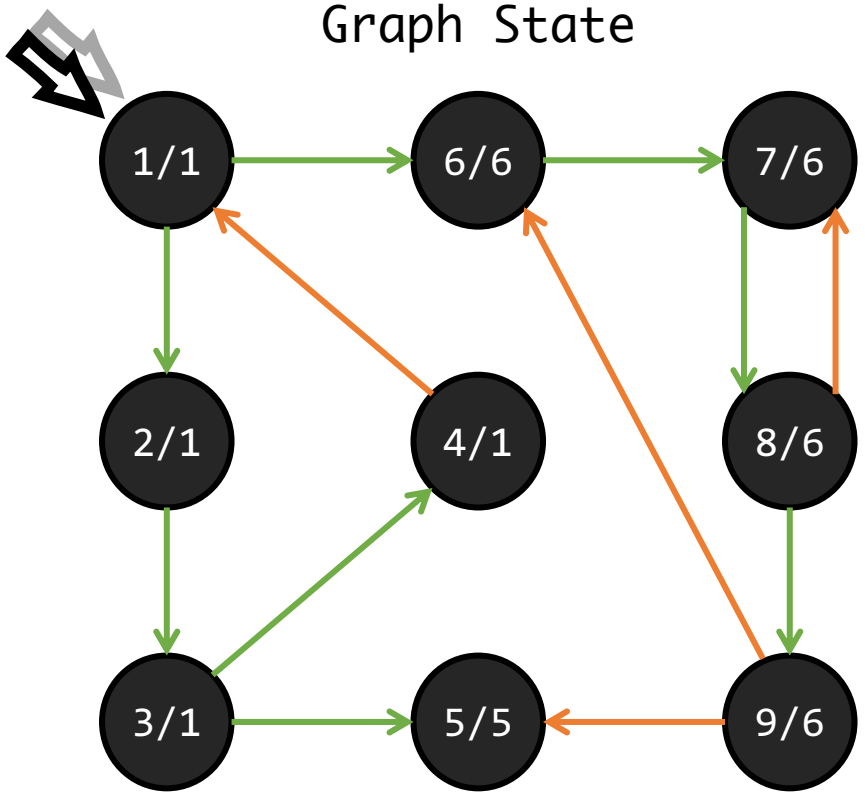
"SCC" Stack

1/1
2/1
3/1
4/1

SCC Set
{5},{6,7,8,9}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)



Function Stack

1/1

"SCC" Stack

SCC Set
{5},{6,7,8,9},
{1,2,3,4}

- Dfn visiting order of DFS
- Low the smallest vertex's DFN which is reachable by passing at most one back edge
- Scc_Stack vertices which have been reached during the search but have not yet been placed in a component

- Forward Step
  - *Cond: exists edge to unvisited vertex*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge
- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge
- **DecidePop Step**
  - *Cond: all out edges explored*
  - If current vertex has Dfn = Low then **pop**, otherwise do nothing
- Backward Step
  - *Cond: Pop Decided*
  - Pop the current vertex out of function stack
  - PrevNode.Low = min(Cur.Low, Fa.Low)

Challenges in Verifying SCC

- DFS is usually implemented as a recursive function in C
- How VST handles recursive call?

```
// fact_spec :=
// forall x, { arg1 -> x }
//           fact(arg1)
//           { ret_val -> x! }
unsigned int fact(unsigned int n) {
  if (n == 0)
    return 1;
  else
            // { arg1 -> (n - 1) }
    return n * fact(n - 1);
            // { ret_val -> (n - 1)! }
}
```

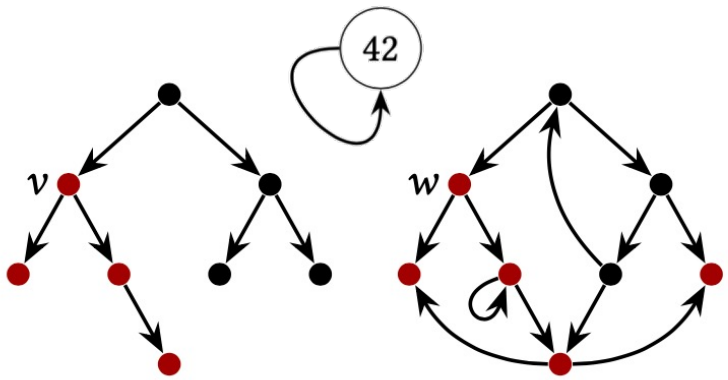
- The $\{P\}\ f\ \{Q\}$ style specification is not convenient to compose proof for complex algorithm
  - Function Specification needs delicate design

```
// PRE: ?
void DFS_visit(struct vertex * x)
{
  struct edge * pe;
  pe = x->vedge;
  x -> visited = 1;

  while (pe != NULL){
    if (pe -> etail ->visited == 0)
      DFS_visit(pe -> etail);
    pe = pe -> next;
  }
}
// POST: ?
```

Challenges in Verifying SCC

- The $\{P\}\ f\ \{Q\}$ style function specification is not convenient to compose proof for graph algorithm
- Separation Logic requires disjoint predicates on heap memory
  - Makes the design of PRE/POST assertion even harder for graph algorithm verification



Our Solution:
Split the Proof into Two Phases

```
// PRE: ?
void DFS_visit(struct vertex * x)
{
struct edge * pe;
pe = x->vedge;
x -> visited = 1;

while (pe != NULL)
// LOOP-INV: ?
{
if (pe -> etail ->visited == 0)
// PRE: ?
DFS_visit(pe -> etail);
// POST: ?

pe = pe -> next;
}
}
// POST: ?
```

Phase 1: Formalize the Algorithm in Small Step Description

- Define small step as relation between states:

  $tarjan\_step: state \rightarrow state \rightarrow Prop := \cdots$

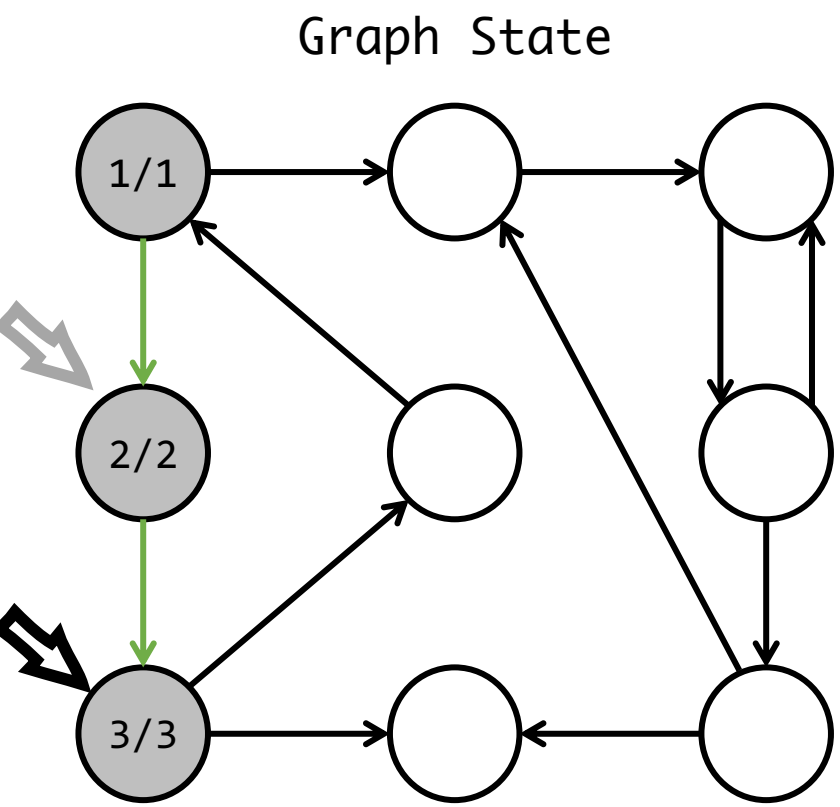
- The **reflexive transitive closure** of the small step can represent an execution of the algorithm (from one state to another)

```
Record state : Type := {
  Func_stack: list Vertex;
  Dfn: Vertex -> nat;
  Low: Vertex -> nat;
  Stack: list Vertex;
  Scc_list: list (list Vertex);
  Vis_edge: Edge -> nat;
  Idx: nat;
  Stage: stage_type;
}.
```

```
Definition Tarjan: relation state := clos_refl_trans state tarjan_step.
Definition Tarjan_n1: relation state := clos_refl_trans_n1 state tarjan_step.
Definition Tarjan_1n: relation state := clos_refl_trans_1n state tarjan_step.
```

```
Inductive clos_refl_trans (A : Type) (R : relation A) (x : A) : A -> Prop :=
    rt_step : forall y : A, R x y -> clos_refl_trans A R x y
  | rt_refl : clos_refl_trans A R x x
  | rt_trans : forall y z : A,
               clos_refl_trans A R x y ->
               clos_refl_trans A R y z -> clos_refl_trans A R x z
```

Example of Small Step Description: Forward



Function Stack

1/1
2/2
3/3

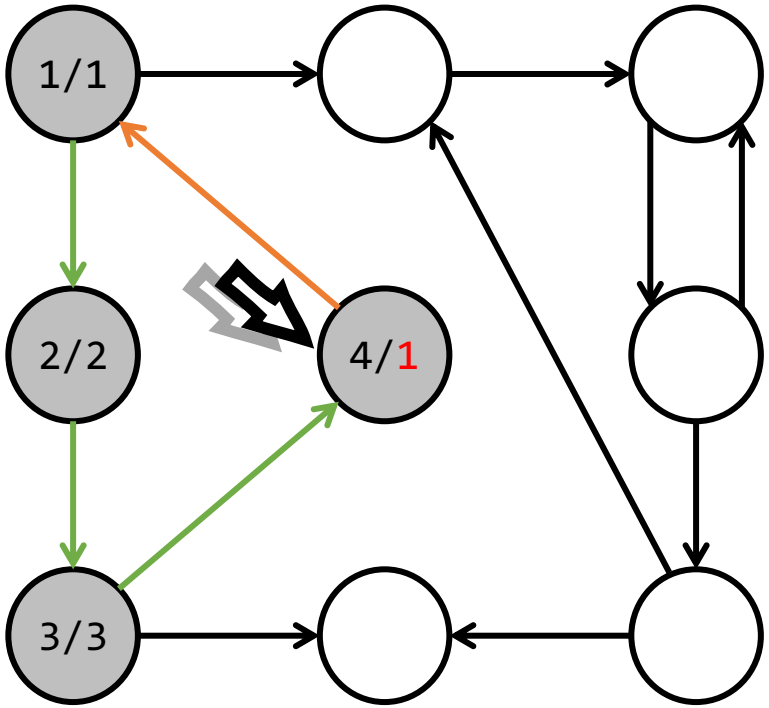
1/1
2/2
3/3

- Forward Step
  - *<u>Cond: exists edge to unvisited vertex</u>*
  - new vertex's Dfn and Low = Idx + 1
  - Push new vertex into Function stack
  - Push new vertex into SCC stack
  - Mark Edge as tree edge

```
tarjan_forwarding: forall e func_stack dfn low stack scc_list vis_edge idx,
  strong_evalid pg e -> vis_edge e = 0 -> dfn (dst pg e) = 0 ->
    tarjan_step {| Func_stack := (src pg e) :: func_stack; Dfn := dfn; Low := low; Stack := stack;
                   Scc_list := scc_list; Vis_edge := vis_edge; Idx := idx; Stage := stage1 |}
                {| Func_stack := (dst pg e) :: (src pg e) :: func_stack;
                   Dfn := ((dst pg e) !-> (S idx); dfn); Low := ((dst pg e) !-> (S idx); low);
                   Stack := (dst pg e) :: stack; Scc_list := scc_list;
                   Vis_edge := (e !--> 1; vis_edge); Idx := (S idx); Stage := stage1 |}
```

Example of Small Step Description: Forward-Visited

Graph State



Function Stack

1/1	1/1
2/2	2/2
3/3	3/3
4/1	4/1

- Forward-Visited Step
  - *Cond: exists edge to visited vertex*
  - If Dst in Stack, current vertex's Low = min(Dst.Dfn, Src.Low)
  - Mark Edge as back edge

```
tarjan_update_visited_1: forall e func_stack dfn low stack scc_list vis_edge idx,
  strong_evalid pg e -> vis_edge e = 0 -> dfn (dst pg e) <> 0 -> dfn (dst pg e) < low (src pg e) ->  In (dst pg e) stack ->
    tarjan_step {| Func_stack := (src pg e) :: func_stack; Dfn := dfn; Low := low;
                   Stack := stack; Scc_list := scc_list; Vis_edge := vis_edge; Idx := idx; Stage := stage1 |}
                {| Func_stack := (src pg e) :: func_stack; Dfn := dfn;
                   Low := ((src pg e) !-> dfn (dst pg e); low); Stack := stack;
                   Scc_list := scc_list; Vis_edge := (e !--> 2; vis_edge); Idx := idx; Stage := stage1 |}
```

Prove Useful Properties on Small Step

- By composing lemmas. $\forall s, StartState(v_0) \rightarrow^* s \Rightarrow P\ s$
- E.g. $\forall s, StartState(v_0) \rightarrow^* s \Rightarrow \forall\ v, (Dfn\ \ s\ \ v) \neq 0 \Rightarrow v_0 \rightarrow^* v$
- Proof. By n-1 induction.
  - Clearly, the initial start satisfy P (only v_0 has non-zero Dfn)
  - Induction Case: $StartState(v_0) \rightarrow^* s \rightarrow s'$,
  - Induction Hypothesis: $\forall\ v, (Dfn\ \ s\ \ v) \neq 0 \Rightarrow v_0 \rightarrow^* v$
  - Goal: $\forall\ v, (Dfn\ \ s'\ \ v) \neq 0 \Rightarrow v_0 \rightarrow^* v$
  - $s \rightarrow s'$ is a single step, the only single step that will update Dfn is forward step. We only need to prove $v_0 \rightarrow^* v_{fwd.dst}$, which can be obtained by $v_0 \rightarrow^* v_{fwd.src} \rightarrow v_{fwd.dst}$

Prove Useful Properties on Small Step

- By composing lemmas. $\forall s, StartState(v_0) \rightarrow^* s \Rightarrow P\ s$
- General. $\forall s, StartState(v_0) \rightarrow^* s \Rightarrow$ $P\ s$
- Proof. By 1-n induction.
  - Clearly, the initial state satisfy P (Why …)
  - Induction Case: $StartState(v_0) \rightarrow^* s \rightarrow s'$,
  - Induction Hypothesis: $P\ s$
  - Goal: $P\ s'$
  - $s \rightarrow s'$ is a single step,

Case analysis on type of single step,
use the step transition condition to finish the proof

Phase 1: Achievement

- We prove that at end state, the algorithm can produce the desired `Scc_list`
  - all_connect
  - maximal

```
Definition Tarjan_all_connect (scc: list (list Vertex)): Prop:=
  forall c, In c scc ->
  forall (v1 v2: Vertex), In v1 c -> In v2 c ->
    reachable pg v1 v2 /\ reachable pg v2 v1.

Definition Tarjan_maximal_scc (scc: list (list Vertex)): Prop:= forall c,
  In c scc ->
   (forall v,
     (forall v', In v' c -> reachable pg v' v /\ reachable pg v v')
   -> In v c).
```

```
Theorem Tarjan_end_all_connect : forall v0 es,
  vvalid pg v0 ->
  Tarjan(pg:=pg) (start_state v0) es ->
  end_state(pg:=pg) es ->
  Tarjan_all_connect (Scc_list es).
```

```
Theorem Tarjan_end_maximal_scc : forall v0 es,
  vvalid pg v0 ->
  Tarjan(pg:=pg) (start_state v0) es ->
  end_state(pg:=pg) es ->
  Tarjan_maximal_scc (Scc_list es).
```

Phase 2: Prove Simulation in VST

- VST handles field operations by symbolic execution
- We need to provide assertions at certain position
- Small Step Description fills the assertion

```
// PRE: StartState(v_0) →* s /\ all out edges of x not visited
void tarjan_visit(struct vertex * x){
  x -> dfn = x -> low = ++idx;
  stk_push(x);
  struct edge * pe = x->vedge;

  while (pe != NULL){.  // INV: EX s', StartState(v_0) →* s'
    struct vertex * y = pe -> etail; /\ some out edges of x visited
    if (y -> dfn == 0){
      // PRE: ?
      tarjan_visit(y);
      // POST: ?
      if (y -> low < x -> low)
        x -> low = y -> low;
    }
    else if (y -> instack == 1
            && y -> dfn < x -> low)
      x -> low = y -> dfn;
    pe = pe -> next;
  }
  if (x -> low == x -> dfn)
    stk_pop(x);
}
// POST: EX s', StartState(v_0) →* s' /\ all out edges of x visited
```

Detailed View of Simulation in VST: Forward

$$\frac{pg \Vdash u \overset{e}{\rightsquigarrow} v, VisEdge(e) = unvisited, Dfn(u) = 0}{\text{graph}(pg) \vdash \begin{bmatrix} u :: FuncStack, Dfn, \\ Low, SccList, Stack, \\ VisEdge, Idx, stage1 \end{bmatrix} \Rightarrow \begin{bmatrix} v :: u :: FuncStack, (v \mapsto Idx + 1; Dfn), \\ (v \mapsto Idx + 1; Low), SccList, v :: Stack, \\ (e \mapsto tree; VisEdge), Idx + 1, stage1 \end{bmatrix}}$$

- From Loop Invariant, we know
  - $EX\ s', StartState(v_0) \rightarrow^* s'$
  - $\exists$ e, x $\overset{e}{\rightarrow}$ y , e is unvisited
- Symbolic Execution gives
  - Dfn(y) = 0
  - Implies all out edges of y not visited
- We can construct a forward step
  - $StartState(v_0) \rightarrow^* s' \xrightarrow{fwd} s''$
- Satisfy the pre-condition of the call

```
// PRE: StartState(v0) →* s /\ all out edges of x not visited
void tarjan_visit(struct vertex * x){
  x -> dfn = x -> low = ++idx;
  stk_push(x);
  struct edge * pe = x->vedge;

  while (pe != NULL){.  // INV: EX s', StartState(v0) →* s'
    struct vertex * y = pe -> etail; /\ some out edges of x visited
    if (y -> dfn == 0){
      // PRE: ?
      tarjan_visit(y);
      // POST: ?
      if (y -> low < x -> low)
        x -> low = y -> low;
    }
    ...
```

Detailed View of Simulation in VST: FwdVisit

$$\frac{pg \Vdash u \overset{e}{\rightsquigarrow} v, VisEdge(e) = unvisited, Dfn(u) \neq 0, v \in Stack}{\text{graph}(pg) \vdash \left[\begin{array}{l} u :: FuncStack, Dfn, \\ Low, SccList, Stack, \\ VisEdge, Idx, stage1 \end{array}\right] \Rightarrow \left[\begin{array}{l} u :: FuncStack, Dfn, \\ (u \mapsto \min(Dfn(v), Low(u)); Low), \\ SccList, Stack, (e \mapsto other; VisEdge), \\ Idx, stage1 \end{array}\right]}$$

- From Loop Invariant, we know
  - $EX\ s', StartState(v_0) \rightarrow^* s'$
  - $\exists$ e, x $\overset{e}{\rightarrow}$ y , e is unvisited
- Symbolic Execution gives
  - $y \in$ Stack, Dfn y != 0
  - x.low = min (y.dfn, x.low)
- We can construct a forward step
  - $StartState(v_0) \rightarrow^* s' \xrightarrow{fwd\_vis} s''$
- Satisfy Loop Invariant

```
// PRE:
void tarjan_visit(struct vertex * x){
  x -> dfn = x -> low = ++idx;
  stk_push(x);
  struct edge * pe = x->vedge;

  while (pe != NULL){.  // INV:  EX s', StartState(v0) →* s'
    struct vertex * y = pe -> etail; /\ some out edges of x visited
    if (y -> dfn == 0){
       ...
    }
    else if (y -> instack == 1
            && y -> dfn < x -> low)
      x -> low = y -> dfn;
    pe = pe -> next;
  }  EX s', StartState(v0) →* s'  /\ some out edges of x visited
  ...
}
```

Phase 2: Achievement

- Correctness of a Funcspec about step simulation
  - { $StartState(v_0) \rightarrow^* s$ /\ all out edges of x not visited }
  - tarjan_visit(v)
  - { $EX\ s', StartState(v_0) \rightarrow^* s'$ /\ all out edges of x visited }
- What we actually want:
  - { $StartState(v_0)$}
  - tarjan_visit(v)
  - { The scc_list of endstate satisfies maximal and connected property }
- By setting $s = StartState(v_0)$, our desired spec is a specialization of the proved spec
  - Implemented in VST by subsumption

Contributions

- An intuitive description and proof of the Tarjan's Algorithm
- Adjacency Table based Graph Library in VST
  - Well defined predicates about vertex, edge list, and graph
  - (Half-)automated tactics that supports reasoning on field manipulation
- Separate the verification on model-level algorithm from concrete implementation, and combine them in VST to ensure functional correctness of the C program

Thank You