The discussion concerning the validity of the two-pass strongly connected components algorithm (due to Sharir) differs from that presented in the text, and seems to be slightly simpler. It is reproduced here.

First the algorithm:

1. Execute a dfs traversal, inserting each node $x$ into a stack upon completion of the call to $dfs(x)$.
2. Reverse the edges of the graph (obtaining the transpose graph).
3. Execute a dfs traversal on the transpose graph, but in the dfs-Sweep wrapper apply the node sequence given by the stack obtained from step 1, top-down order.
4. For each call to $dfs()$ from the dfs-Sweep of the previous step, the nodes traversed in the call give the node-set of a strongly connected component in the original graph.

To establish validity, we make use of the following Assertions, described in class, also found in the text.

1. (White Path Principle) At the time of the call to $dfs(x)$, if there is a white path from the node $x$ to a node $v$, then $v$ will descend from $x$ in a dfs-tree, and the call to $dfs(x)$ gets completed after the call to $dfs(v)$ gets completed.

2. At the time of the call to $dfs(x)$, if there is a path from $x$ to a gray node $v$ (so that the call to $dfs(v)$ is in progress), then there is a path from from $v$ to $x$ in the graph, so that $x$ and $v$ are strongly connected. (Follows from the nesting property of [discover, finish] intervals, implying that $x$ must descend from $v$ in a dfs-tree.)

3. The transpose and original graphs have the same strongly connected components

4. For any node $x$, a dfs-tree containing $x$ must contain the entire strongly connected component to which $x$ belongs.

We proceed to demonstrate the next Assertion, which departs from the treatment provided in the text.

5. If there is an edge from a node $v$ belonging to a component $D$ to a node $w$ in a separate component $C$, then the call to $dfs(v)$ gets completed subsequent to completion of the call to $dfs(z)$ for any node $z$ in $C$. 
Proof: If all nodes in $C$ are black at the time of the call $\text{dfs}(v)$, then the assertion follows by definition. On the other hand, if all nodes in $C$ are white at the time of the call $\text{dfs}(v)$, then the above Assertion 1 implies that the call to $\text{dfs}(v)$ gets completed subsequent to the completions of the $\text{dfs}(z)$ calls for all of the nodes $z$ in $C$. Now we argue that these two cases exhaust the possibilities. Certainly, no node in $C$ can be gray at the time of the $\text{dfs}(v)$ call, since according to the above Assertion 2, this would violate the presumed distinctness of $D$ and $C$. But if there are a mix of both white and black nodes in $C$, then there must also appear a gray node: For consider the first node $x$ in $C$ on which $\text{dfs}()$ gets called. At this point all nodes in $C$ are white, except for $x$ which has just turned gray. The above Assertion 1 implies that all nodes in $C$ descend from $x$ in the $\text{dfs}$-tree containing $x$, and the nesting of [discover, finish] intervals implies that $x$ is the last node in $C$ to turn black. So $x$ is a gray presence in $C$ until all of its nodes have turned black.

In view of the above fourth Assertion, validity of the algorithm hinges upon showing:

Claim With respect to the $\text{dfs}$ traversal of the transpose graph, no $\text{dfs}$-tree contains two nodes that belong to separate components.

Proof: (See the diagram) Assume the contrary; that in the $\text{dfs}$-Sweep of the transpose graph in step 3 a call $\text{dfs}(x)$ generates a $\text{dfs}$-tree containing more than the component $C$ to which $x$ belongs. The root of this $\text{dfs}$-tree is $x$, so that all nodes in this tree descend from $x$. In order for there to be a node descending from $x$ belonging to some component other than $C$, there would have to be an edge in this $\text{dfs}$-tree directed outward from some node $w$ in $C$ to a node $v$ in some other component $D$. Considering such an edge, which belongs to the transpose graph, the corresponding edge in the original graph traversed in step 1 would have been directed from $v$ to $w$. But the above Assertion 5 implies that the call $\text{dfs}(v)$ (in
the depth first search traversal of step 1 would have completed after the dfs(x) call had completed. Therefore in step 3 by the time that the call to dfs(x) gets underway, v would have already been traversed (reflecting the traversal priority in the dfs-Sweep, which assigns higher priority to a node having later dfs completion time in step 1). Therefore the edge from w to v cannot belong to this tree, resulting in a contradiction. □