Q: Can you give more explanation about the solution to the dangling else problem?

A: There are several basic principles; these principles are similar to decomposing a problem into a recursive solution in programming.

First we need a non-terminal that represents productions that only include if-statements with a matching else, i.e., *M*.
Second we need a non-terminal that represents productions that allow if-statements that have no else, i.e., *U*.

Now we have to define the necessary rules. We know that the then-part cannot have an if without an else if there is another else that follows. So we have

M –> if E then M else ?

U –> if E then M else ??

Since M cannot have unmatched if-statements and U must have one, we have

M –> if E then M else M

U –> if E then M else U

Now we need a base case for each of these non-terminals otherwise there will be no terminating production (like reasoning about recursive procedures).

M –> if E then M else M
M –> ?

U –> if E then M else U
U –> ??

U needs to include an unmatched if-statement.

M –> if E then M else M
M –> ?

U –> if E then M else U
U –> if E then ??

We cannot replace ? with U because we would be right back where we started with the dangling else problem. Thus we need a non-terminal that represents productions that do not contain if-statements, i.e., *other*. That is, the ? would be replaced by *other* because we cannot allow the non-terminal M to recurse back to M since that would still be non-terminating.

M –> if E then M else M
M –> *other*

U –> if E then M else U
U –> if E then ??

We can't replace the ?? with U in the last production rule because that would also be non-terminating. One might consider replacing ?? with M, but this would eliminate some of the valid unmatched if-statements, i.e., "if E1 then if E2 then S2" would not be derivable if ?? were replaced by M. This is why we also need S.
S –> M
S –> U

M –> if E then M else M
M –> *other*

U –> if E then M else U
U –> if E then S

The reason S works in the last production (U –> if E then S) is because S –> M has a terminating rule, i.e., M –> *other*.

I hope this helps. Note that I wouldn't expect students to come up with something like this on an exam. When I took compilers, we had this as a homework problem. I spent several hours trying to figure it out, so it would be ridiculous to have such questions on an exam (but it was an interesting homework problem). I've made it easy for you because the solution is in the book.