

Compressed Tree and Code Motion Transformations

Extend your semantic Analyzer so that it converts the parse tree into a Compressed Tree, preserving the references to specific variable identities produced by SA. In a Compressed Tree, expressions have been simplified, eliminating nodes that will be useless during code generation (usually, these are nodes with only one non-terminal descendant). Further, array references, like $A[\langle \text{expr} \rangle]$, have been expanded to include explicit nodes for the operators in the equivalent C expression “ $*(\&A + \langle \text{expr} \rangle \ll 2)$ ”. Also, each WHILE statement should be analyzed (to determine the set of variables it references, but does not assign to), and re-written into the nested-IF form presented in class. As part of this transformation, the largest sub-expressions occurring in the WHILE that contain only constants, and locally constant variables, should be moved to the entry block of the transformed WHILE.

Moved expressions should be turned into a new kind of statement $\langle \text{ExpStat} \rangle$, and linked into a $\langle \text{statement list} \rangle$ which forms the entry block of the WHILE. The moved expressions should also be pointed to from their old positions within the body of the WHILE.

Note that the nodes of the $\langle \text{test} \rangle$ must be duplicated (I think you can simply call on `node()` to do this), so that each copy can participate independently in the later transformation into DAG's.

WHILE statements nest hierarchically, and this complicates maintaining the necessary sets of variables. For example, a variable may be assigned to in one WHILE, X, and not within a sibling WHILE, Y. In the containing WHILE, this variable IS assigned to, but it is not assigned to in WHILE Y.

Print the nodes of the completed Compressed Tree, showing their nodes numbers, and giving an interpretation of their rule numbers.