Algorithms Appendix - Syntax Directed Editing

# 1. Replace/Propagate

Pseudo-code:  
  
function Replace(r, U'):  
 detach subtree rooted at r  
 graft U' in place of r  
 mark affected nodes as dirty  
 enqueue dirty nodes into worklist  
 while worklist not empty:  
 n = dequeue(worklist)  
 for each attribute a of n:  
 if dependencies of a are satisfied:  
 recompute a  
 propagate changes to dependents  
  
Complexity: O(|Affected|) where |Affected| is number of nodes in dirty set.

# 2. Threaded-Tree LR Parsing with Lazy NCA

Pseudo-code:  
  
function IncrementalParse(x\_end, y'):  
 reset parse stack at x\_end  
 parse y' using LR driver  
 result\_subtree = built parse tree  
 z = LazyNCA(x\_end, result\_subtree)  
 graft result\_subtree at z  
 mark reused vs disposable nodes  
  
function LazyNCA(x, subtree):  
 start from x upward until predecessor, successor, and symbol constraints are satisfied  
 return first valid ancestor  
  
Complexity: amortized near O(|y'|), optimal node reuse ensured.

# 3. Attribute DAG Maintenance

Pseudo-code:  
  
function EvaluateAttributes(dirty\_set):  
 initialize worklist = dirty\_set  
 while worklist not empty:  
 node = dequeue(worklist)  
 for each attribute a of node:  
 if dependencies ready:  
 compute a  
 enqueue dependents into worklist if stale  
  
Cycle detection:  
 maintain visited set per evaluation round  
 if an attribute is revisited without new info, report cycle  
  
Complexity: O(|Affected|) with guarantee of termination unless cycles exist.

This appendix formalizes the algorithmic backbone of Syntax-Directed Editing: incremental syntax maintenance, efficient semantic propagation, and attribute scheduling.