Syntax Directed Editing - Master Document

# Part I: Core Concepts

Syntax-Directed Editing (SDE) is an approach to program and document editing that enforces syntactic correctness by coupling the editing process with the grammar of the underlying language. The foundation of SDE lies in attribute grammars and syntax trees, ensuring that any intermediate state of the document remains well-formed.

## UI and Editing Model

The user interface consists of dual views: a tree-based structural editor and a text-based view. Cursor movement follows a preorder traversal semantics, and editing actions are constrained to valid productions. Menus dynamically adapt to the grammar at the cursor position. Display inertia is maintained by minimizing redraws to only the spans affected by edits.

# Part II: Algorithms

At the heart of the SDE implementation are algorithms for maintaining correctness and incrementality:  
- Replace/Propagate/Evaluate/Visit contract for attribute grammar evaluation.  
- Threaded-Tree LR parsing with lazy nearest common ancestor (NCA) for grafting.  
- Dependency DAG scheduling for efficient attribute re-evaluation.

## Replace/Propagate Algorithm

Replace(r, U’): Given a node r and a replacement subtree U’, the Replace operation grafts U’ into the syntax tree. Propagation ensures that attribute values for all affected nodes are recomputed. The algorithm guarantees O(|Affected|) complexity.

## Threaded-Tree LR Parsing

The threaded-tree incremental LR parser maintains a parse stack encoded in the tree via threaded links. Upon edit, the stack is reset at the endpoint of the edited region, the replacement string y’ is parsed, and the resulting subtree is grafted optimally. Nodes are marked as Reused or Disposable to minimize reallocation.

# Part III: Grammar Tooling

BNF specifications are themselves editable in SDE. Grammars are parsed, analyzed for FIRST/FOLLOW sets, nullable symbols, and conflicts. A grammar lint system warns about ambiguities and left-recursion. Future work includes integrating GLR fallback for resolving unavoidable ambiguities.

# Part IV: Open Issues & Roadmap

Open issues identified:  
- Ambiguity handling: need GLR fallback or conflict resolution strategies.  
- Error tolerance: incremental error recovery mechanisms.  
- Persistent undo/redo with branching version control of AST.  
- Incremental pretty-printing with span-based repaint.  
  
Roadmap: Harden ambiguity resolution, implement error recovery, optimize repaint contracts, and integrate grammar QA tooling.

# Appendix

Detailed pseudo-code and complexity analysis for Replace/Propagate, Threaded-Tree LR parsing with lazy NCA, and Attribute DAG maintenance are provided in the Algorithms Appendix (separate document).