Executive synthesis

- **Problem framing & canon**: Clear articulation of syntax-directed editing, template-driven UI, and attribute-grammar semantics; anchored in Cornell Program Synthesizer, Reps/Demers, Degano, and Larcheveque (threaded-tree LR)

- **UI contract**: Dual-view (document tree + editor), menu of valid productions at cursor, preorder cursor semantics, reversible actions/undo, display inertia principles

- **Core algorithms**:
    - **Attribute-grammar flow** with **Replace / Propagate / Evaluate / Visit**; Replace is O(|Affected|) and formally defined (graft/prune then propagate)

    - **Incremental LR with threaded trees**: reset stack at end of x; parse y′; lazy NCA for optimal graft point; constraints on predecessor/successor/symbol match; node marking (Reused/Disposable)

- **BNF → BNF**: Self-hosting: parse BNF, transform to edited-language grammar; map stored as associative structures; Yacc/Bison compatible; ambig/left-recursion caveats called out

- **Data model**: Document as AST with terminal/nonterminal nodes; grammar stored/traversed to drive menus; editing via subtree graft/prune; pretty-print/unparse symmetry

Gaps & deltas

1. **Ambiguity handling**: You flag left-recursion/ambiguity risks but don’t specify the resolution strategy (GLR split, Earley fallback, or LR(k) conflict policy) beyond “design the parser to deal with it”

2. **Error-tolerant parsing**: UI mentions reversible actions and data-integrity but not concrete incremental error recovery (island grammars, fence tokens, or error productions)

3. **Semantic dependency maintenance**: Replace/Propagate is defined, but the **dependency graph maintenance** and **dirty-set computation** under concurrent edits need a precise algorithmic contract (callbacks vs. scheduler; topological slices)

4. **Performance envelope**: Threaded-tree optimality is stated; no **amortized bounds** for lazy-NCA with real-world edits (large y, repeated small edits, deep trees)

5. **Persistence & versioning**: Undo/redo is promised; the storage model for **snapshotting**, **branching**, and **merge** of AST/attributes isn't specified

6. **View consistency**: Dual view is described; no explicit **diff-aware layout** or **incremental pretty-printer** contract for minimal repaint

7. **Grammar tooling UX**: BNF editor exists; the **grammar QA loop** (FIRST/FOLLOW visualizer, conflict report, left-recursion elimination wizard) is not designed yet

Decision stack (recommended target state)

- **Parser core**: Keep **LALR(1) + threaded tree** as the happy path; add a **GLR fallback** module gated behind a feature flag to resolve unavoidable ambiguity with bounded cost.

- **Lazy NCA**: Retain your marking scheme; extend with **budgeted graft search** (cap per-edit NCA exploration depth; degrade gracefully to local reparses) for predictable latency.

- **Attributes runtime**: Implement a **worklist scheduler** over a maintained **dynamic dependency DAG**; store per-node version stamps; evaluate with **dirty-bit propagation** and **cycle detection** bailout to user hints.

- **UI**: Enforce **display inertia** with an incremental pretty-printer that operates on **layout spans** tied to AST nodes; repaint strictly by affected span set.

- **BNF pipeline**: Add **grammar lint**: FIRST/FOLLOW, nullable, conflict table, LR item set browser; one-click **left-recursion elimination** and **precedence/associativity** helpers.

MVP architecture (concrete, shippable)

- **Modules**
  - `grammar-core`: BNF parser, grammar graph, LR table builder, conflict lints
  - `ttlr` (Threaded-Tree LR): threaded nodes, stack threads, shift/reduce ops, **incremental reparsing** with lazy NCA and node-state bits (Reused/Disposable)
  - `attrib-engine`: attribute DAG, Replace/Propagate/Evaluate API (O(|Affected|) guarantee)
  - `unparse/pretty`: structure-aware unparser, span map, minimal repaint
  - `ui-shell`: dual view (tree + text), production menus, cursor preorder semantics, undo/redo log with structural ops
- **Data contracts**
  - `Node { id, sym, state, threadPrev, children[], attrs, marks:{reused,disposable} }` (threaded-tree node)
  - `Edit { xEndPtr, delta(y→y'), tokensAround, ts } → ReparsePlan { resetStackFrom(xEndPtr), parse(y'), graft(z) }`
  - `DepDAG { attr -> attr edges }` with **dirty set** and **version clocks**; `Replace(r, U')` contract as specified

KPI envelope

- **Edit latency** P95 < 20 ms for $y' \le 64$ tokens; **degrade** linearly with $|y'|$; **repaint** only affected spans (≤ 3% of doc)

- **Semantic catch-up**: O(|Affected|) attribute re-eval per Replace (prove via DAG worklist accounting)

- **Parser reuse**: ≥ 90% node reuse for single-token substitutions in stable regions (threaded-tree optimality target)

Risk register & mitigations

- **Ambiguity spikes** → Ship GLR fallback + editor warnings; allow per-nonterminal precedence rules (mitigates shift/reduce churn)

- **Attribute cycles** → Cycle detector with node-scoped diagnostics; suggest attribute rewrites at the grammar layer

- **Deep graft cost** → Budgeted NCA; if budget exceeded, **localized reparse** of enclosing production.

10-day sprint plan (sequenced, low-risk)

1. **Threaded-tree kernel**: node struct, shift/reduce, stack-reset from x, y′ exhaustive parse; golden tests for small edits

2. **Lazy NCA + marking**: implement Disposable/Reused flags, graft acceptance checks (pred/succ/symbol) with unit tests

3. **Replace/Propagate**: worklist engine honoring O(|Affected|) semantics; inject counters for attribution budget

4. **BNF ingest & lint**: load BNF, build LR tables, FIRST/FOLLOW visualizer; block ambiguous grammars behind GLR flag

5. **UI thin-slice**: tree view + text view, menu of productions at caret, preorder cursor policy, minimal repaint scaffolding

Recommendations

- **Precise graft algorithm**: you have the rules; add **pseudo-code** and a proof sketch tying lookahead==terminal successor and symbol/pred equivalence to well-formedness (you already hint at this)

- **Incremental pretty-printer**: define the **span map** from AST nodes to layout tokens and the invalidation protocol on Replace; this closes the UX loop

- **Grammar QA UX**: bake conflict diagnostics into the BNF editor flow with suggested rewrites (left-factoring wizard)