Notes on Cited Works

Fischer, Huch, & Wilke (2010) — *A Play on Regular Expressions: Functional Pearl*
Gist. Introduces the *marked* or *pointed* approach: rather than growing expressions via derivatives, marks are propagated through a fixed abstract syntax tree to indicate where matching can continue. The paper is presented playfully as a "play," where regular expressions and strings interact as game participants.
Important details. Marks are placed *after* consuming a symbol (*mark-after-atom*). The algorithm is purely functional, keeps the expression size constant, and still exposes enough structure to reconstruct how a match proceeds.
Used in this report. Re-implemented as the starting point (Appendix **??**); serves as the conceptual foundation for later extensions with bitcodes, `\shift`/`\shifts`, and `NTIMES` handling.

Asperti, Sacerdoti Coen, & Tassi (2010) — *Regular Expressions, au point* **Gist.** Formalizes pointed regular expressions within a proof-assistant framework, adopting a *mark-before-atom* semantics (points denote the next atoms to be read).
Important details. Although described under McNaughton–Yamada, its operational behavior differs from Fischer's version: marks advance to future atoms rather than marking consumed ones. This distinction affects acceptance and automata correspondence.
Used in this report. Contrasted with Fischer's *mark-after-atom* version; cited in the motivation section to highlight the dual construction later proven by Nipkow and Traytel.

Brzozowski (1964) — *Derivatives of Regular Expressions* **Gist.** Defines the derivative $\partial_a r$ and establishes a recursive procedure for membership testing by iterating derivatives and checking nullability.
Important details. Provides clean algebraic rules for $+$, concatenation, and star, proving that $w \in L(r)$ iff $\varepsilon \in L(\partial_w r)$. The method is elegant but can syntactically expand expressions, motivating later simplifications.
Used in this report. Forms the theoretical baseline for derivative-based matching and the foundation for your comparison to marked and bitcoded approaches.

Antimirov (1996) — *Partial Derivatives of Regular Expressions and Finite Automaton Constructions* **Gist.** Extends Brzozowski's idea to compute *sets* of residuals, yielding NFAs whose states represent all possible continuations after reading a symbol.
Important details. Worst-case growth reaches $O(n^3)$ because (i) the number of distinct partial derivatives can be $O(n)$, (ii) each may contain $O(n)$ summands, and (iii) each summand can have size $O(n)$. Partial derivatives mitigate but do not remove blow-ups.
Used in this report. Cited to illustrate polynomial but unbounded growth, motivating your later focus on constant-shape marked matching and simplification strategies.

Owens, Reppy, & Turon (2009) — *Regular-expression derivatives re-examined* **Gist.** Revisits derivatives from an implementation perspective, emphasizing simplification and sharing to make derivative computation practical.
Important details. Introduces canonicalization for ALTs (ACI laws), neutral/absorbing simplifications for SEQs, and improved nullable propagation. Despite optimizations, nested STAR/SEQ constructs remain problematic.
Used in this report. Serves as evidence of modern interest in derivatives and the precursor to Tan and Urban's formally verified simplification pipeline.

Sulzmann & Lu (2014) — *POSIX Regular Expression Parsing with Derivatives* **Gist.** Extends derivatives to produce *POSIX* (leftmost-longest) parse values using bitcodes or injections

to encode branch and repetition choices.
Important details. Introduces bitcode annotations that track choices during derivative evaluation, allowing later reconstruction of the POSIX value; also notes simplification is essential to avoid growth.
Used in this report. Directly informs your Bit-Annotated Versions 1–2 and the design of `\mkfin`, `\mkeps`, and POSIX value extraction.

Might, Darais, & Spiewak (2011) — *Parsing with Derivatives: A Functional Pearl*
Gist. Generalizes the derivative concept from regular expressions to context-free grammars, treating parsing as iterative differentiation that constructs parse trees.
Important details. Demonstrates compositional parsing in a functional setting; highlights the need for memoization and ambiguity handling. Extends the reach of derivatives beyond regular languages.
Used in this report. Provides broader theoretical framing; supports the functional motivation for your Scala-based matcher implementations.

Tan & Urban (2023) — *POSIX Lexing with Bitcoded Derivatives* **Gist.** Presents a formally verified, bitcoded-derivative lexing algorithm that guarantees unique POSIX values and finite bounds on derivative growth.
Important details. The simplification pipeline—flattening, duplicate removal, and `bsimpSEQ`/`bsimpALTs` plus the language-subsumption (`LD`) rule—is key to provable boundedness. Their Isabelle/HOL proofs show correctness and uniqueness.
Used in this report. Central reference: informs your simplification reasoning, `\mkfin`/`\mkeps` structure, and comparison to derivative-based growth.

Nipkow & Traytel (2014) — *Unified Decision Procedures for Regular Expression Equivalence* **Gist.** Provides a unified Isabelle/HOL framework covering multiple decision procedures, both derivative-based and marked, enabling direct comparison and correctness proofs.
Important details. Shows that the two marked approaches (Fischer vs. Asperti) are dual (*mark-after* vs. *mark-before*) and proves quotient relations between their automata. Introduces the `\follow` and `\readF` operators.
Used in this report. Supports your duality claim and justifies your use of `\follow`/`\readF` when discussing the operational difference between marked variants.

Okui & Suzuki (2013) — *Disambiguation in Regular Expression Matching via Position Automata with Augmented Transitions* **Gist.** Proposes an augmented position-automaton construction that enforces the POSIX leftmost-longest rule via priority-encoded transitions.
Important details. Establishes a worst-case complexity of $O(m(n^2 + c))$ (regex size m, input length n, alphabet size c). Demonstrates that disambiguation can be embedded structurally, not handled as a later tie-break.
Used in this report. Provides an external automata-based reference for POSIX ordering and complexity, contextualizing your Version 2 “generate then order” approach.