Combinatorial Explosion with 12–15 Primitives

1) Ordered sequences, one primitive per step

- Without repetition (each primitive used at most once): Count of sequences of length k = P(n,k) = n! / (n-k)! - With repetition (can reuse a primitive): Count of sequences of length k = n^k
Concrete numbers: - For n=12, K=3: No repeat = [12, 132, 1,320] (total 1,464). With repeat = [12, 144, 1,728] (total 1,884). - For n=12, K=5: Totals ≈ 108k (no repeat) vs ≈ 271k (with repeat). - For n=15, K=3: No repeat = [15, 210, 2,730] (total 2,955). With repeat = [15, 225, 3,375] (total 3,615). - For n=15, K=5: Totals ≈ 396k (no repeat) vs ≈ 814k (with repeat).

2) Multiple primitives in the same step (concurrent sets)

If you allow a “step” to be any non-empty subset of primitives, the options per step are (2^n - 1). - For n=12: 4,095 options per step. Two steps → ~16.8 million. Three steps → ~68.7 billion. - For n=15: 32,767 options per step. Two steps → ~1.07 billion. Three steps → ~35.2 trillion.

3) Parameters multiply the space

If each primitive has r discrete parameter choices (e.g., amounts, slippage bands), sequences of length k get multiplied by r^k. Example: n=12, k=4, with repetition and r=10 → 12^4 × 10^4 ≈ 2.07 × 10^8. With continuous parameters, the raw space is effectively unbounded.

4) Why WDD + constraints are essential

- A static hash map of allowed combos can help for a few hard “never” rules, but won’t scale with these numbers. - WDD (Warp→Detect→Denoise) slashes the search by only keeping primitives with real trace evidence and suppressing phantoms. - A small declarative constraint layer (compatibility + ordering + stateful guards) prunes the rest. - Verifiers (AMM invariant, HF≥1, oracle freshness, etc.) are the final gate → abstain if anything fails.

TL;DR

With 12–15 primitives, even short sequences (4–5 steps) already reach 10^4–10^6 possibilities; allowing combos per step jumps to 10^9–10^13+. Parameter choices multiply these counts dramatically. This is why we don’t enumerate: we detect from signals (WDD), apply tight constraints, and verify.