# Hermeneutic Calculator

I am building a unified, testable theory of how students *develop* arithmetic understanding—not just a catalog of strategy names, but a developmental map of how those strategies *emerge, elaborate, invert, and nest.* My target is to formalize roughly 25 student-invented strategies across addition, subtraction, multiplication, and division, showing how each one is algorithmically constructed from prior embodied practices.

## Choreography as Computation

I treat each strategy as **written choreography for embodied cognition**. A formal automaton (register machine, bounded DPDA, or related model) becomes a script for the temporal unfolding of thought: initialize, transform, check, recurse, terminate. The power of this framing is that it preserves *how* a student actually moves through a calculation—counting up, pausing at a boundary, decomposing a number—rather than replacing those moves with opaque symbolic shortcuts.

## Two Fundamental Movements

I analyze student action through a dialectic of temporal structure: 1. **Temporal Compression (Sublation / Recollection):** Unitizing many micro-acts into a larger cognitive unit (ten ones $\rightarrow$ one ten; 3 base jumps $\rightarrow$ a single composite stride). Compression accelerates flow. 2. **Temporal Decompression (Determinate Negation):** Strategically undoing or expanding a composite to restore fine control (borrowing a ten; splitting 5 into $2 + 3$ in RMB; decomposing a factor for distributive reasoning). Fluency grows as students coordinate these movements, learning *when* to expand and *when* to re-compress.

## Fractal Architecture: Iterative Core + Strategic Shell

Across strategies I repeatedly recover the same **fractal pattern**: * **Iterative Core:** A minimal loop (initialize $\rightarrow$ step ($+1$, $-1$, $+Base$, $+Chunk$) $\rightarrow$ condition check). Counting by ones, skip counting, and accumulation loops in division all instantiate this engine. * **Strategic Shell:** A supervisory layer that *prepares*, *optimizes*, or *transforms* the problem so that the core runs fewer or cognitively lighter iterations. RMB, Rounding & Adjusting, Chunking, Sliding, Distributive and Inverse Distributive Reasoning all wrap the core with analysis (e.g., “find gap $K$", "split factor", "slide both numbers"). Because the shell often *invokes* the core as a subroutine (e.g., CountUpToBase, CountBackK), the global structure becomes self-similar: strategies *contain* (and sometimes nest) earlier strategies. This produces a genuine computational fractal—not metaphorical flourish, but recurrence of the same control schema at different conceptual scales.

## Mechanisms of Elaboration

I observe three progressive forms of algorithmic elaboration: 1. **Compression of Action:** Replacing many +1 steps with $+Base$ or $+StructuredChunk$ (COBO, Chunking). 2. **Optimization of Iteration:** Dynamically computing the *size* of a future stride (RMB gap $K$; Chunking's bridging chunk; Sliding's constant difference) before acting—analyze then accelerate. 3. **Structural Transformation:** Rewriting the problem space (Rounding detour + compensation; Distributive split; Sliding invariance; Inverse Distributive decomposition of dividend). Advanced strategies chain these moves (e.g., Rounding = transformation $\rightarrow$ compressed addition $\rightarrow$ compensatory inverse steps).

## Inversion of Practice

Subtraction fluency emerges not by inventing alien procedures but by **inverting or repurposing** addition shells: Missing Addend reframes subtraction as forward accumulation; Counting Back mirrors Counting On; Sliding preserves difference across a translation; Borrowing reverses carry (decompression of a prior sublation). Division analogues (Dealing by Ones vs. Coordinating Two Counts; Inverse Distributive Reasoning) continue the same inversion logic.

## Collaboration and Iterative Refinement

This project is explicitly *collaborative* with an AI assistant. I bring student transcripts, pedagogical insight, and theoretical intent; the assistant supplies relentless formal scrutiny—flagging mis-specified state sets, hidden non-determinism, premature algebraic assumptions, or missing termination guarantees. A typical refinement cycle: 1. Draft informal description from transcript. 2. Specify automaton (states, registers, transitions) in a first-pass formalism. 3. Implement executable prototype (Python) to test determinism, termination, and behavioral alignment with the transcript (sequence reconstruction like "$46, 56, 66, \ldots$"). 4. Trace failures (e.g., an early Rounding model produced an infinite loop after overshoot; an initial Chunking diagram hid the cognitive search for $K$) and revise. 5. Re-abstract the corrected machine into concise LaTeX-friendly specification. This loop ensures every claimed cognitive choreography *runs*—a falsifiability and reproducibility standard often missing in purely diagrammatic accounts.

## Why Executable Formal Models Matter

An executable automaton does four things for me: * **Validity Check:** Catches hidden cycles or unreachable states. * **Phenomenological Fidelity:** Lets me align generated action traces with verbatim student utterances. * **Comparative Anatomy:** Normalizes different strategies into a shared tuple structure so I can map elaboration edges precisely. * **Pedagogical Insight:** Identifies which internal subroutines (e.g., "CountBackK") must be instructionally stabilized before a composite strategy will consolidate.

## Algorithmic Elaboration (Brandom Frame)

Following Robert Brandom, I treat these developments as **algorithmic elaborations**: later practices are *PP-sufficient* expansions of earlier ones—achieved by reorganizing, nesting, or inverting existing abilities rather than importing foreign primitives. Some strategies become **LX** relative to prior practice: they both derive from and make explicit what was implicit (RMB makes base boundaries explicit; Borrowing renders the reversibility of carry explicit; Distributive Reasoning makes latent additivity across factors explicit).

## Scope of This Document

Below I give each strategy a uniform template: description, formal specification, choreography (compression/decompression dynamics), and genealogical lineage. I remove historical critique and raw code to foreground the structural logic while preserving testability through the already verified prototypes. The introduction you are reading consolidates the nuance of origin (embodiment), evolution (fractal elaboration), collaboration (human + AI), and rigor (execution + revision) from the longer source manuscript without duplicating passages.

---

# Hermeneutic Calculator: Strategy Formalizations

This draft reorganizes the strategies as primary sections. Each section supplies:

1. Phenomenological description (student-facing practice).
2. Formal automaton / register-machine specification in LaTeX-friendly notation.
3. Core choreography (temporal compression/decompression dynamics).
4. Algorithmic elaboration lineage (what primitives it builds upon).

All implementation details (Python prototypes) and historical critiques have been removed. Mathematical symbols are formatted for Pandoc $\rightarrow$ LaTeX conversion.

Notation (uniform across strategies):

- $M = (Q, V, \delta, q_0, F)$: machine with states $Q$, registers (or variables) $V$, transition function $\delta$, start state $q_0$, accepting states $F$.
- When convenient, we use auxiliary internal variables; these are included in $V$ implicitly.
- Counting primitives: Count Up (+1), Count Back (−1) regarded as atomic embodied actions.
- Temporal Compression: synthesizing many unit actions into a higher-order unit (e.g., a "ten").
- Temporal Decompression: strategic expansion of a unit into constituent parts.

---

## Counting and Counting On

**Description.** Sequential unit counting within a bounded base-10 place-value structure $(0 - 999)$. Embodied iterations ("ticks") increment units, propagate carries (sublation) into tens and hundreds.

**Formal Model (Sketch).** Deterministic PDA (bounded) or 3-register counter. For LaTeX exposition we specify a DPDA tuple:

$$M_{count} = (Q, \Sigma, \Gamma, \delta, q_{start}, Z_0, F)$$

with place-value stack symbols $U_i, T_j, H_k$. The transition function $\delta$ is defined as follows:

| Current State | Input | Top of Stack | Next State | Action (Stack) | Interpretation |
|---|---|---|---|---|---|
| $q_{start}$ | $\varepsilon$ | $Z_0$ | $q_{idle}$ | Push$(U_0, T_0, H_0)$ | Initialize count to 0. |
| $q_{idle}$ | `tick` | $U_n$ $(n < 9)$ | $q_{idle}$ | Pop; Push$(U_{n+1})$ | Increment units. |
| $q_{idle}$ | `tick` | $U_9$ | $q_{inc\_tens}$ | Pop | Unit overflow, carry to tens. |
| $q_{inc\_tens}$ | $\varepsilon$ | $T_m$ $(m < 9)$ | $q_{idle}$ | Pop; Push$(T_{m+1}, U_0)$ | Increment tens, reset units. |
| $q_{inc\_tens}$ | $\varepsilon$ | $T_9$ | $q_{inc\_hundreds}$ | Pop | Ten overflow, carry to hundreds. |

| Current State | Input | Top of Stack | Next State | Action (Stack) | Interpretation |
|---|---|---|---|---|---|
| $q_{inc\_hundreds}$ | $\varepsilon$ | $H_k$ $(k < 9)$ | $q_{idle}$ | Pop; Push$(H_{k+1}, T_0, U_0)$ | Increment hundreds, reset lower places. |
| $q_{inc\_hundreds}$ | $\varepsilon$ | $H_9$ | $q_{halt}$ | Pop; Push$(H_0, T_0, U_0)$ | Counter overflow. |

**Choreography.** Carry = temporal compression: ten unit steps recollected as one higher unit. Borrow (in inverse counting) is temporal decompression.

**Elaboration Lineage.** Primitive for all subsequent additive, subtractive, multiplicative, and divisional strategies.

---

## Rearranging to Make Bases (RMB)

**Description.** For $A + B$, identify gap $K$ from $A$ to next base (e.g., 10, 100), decompose $B = K + R$, form $A' = A + K$ (a base), then compute $A' + R$.

**Machine.**

$$M_{RMB} = (Q, V, \delta, q_0, F)$$

with

$$Q = \{q_{start}, q_{calcK}, q_{decompose}, q_{recombine}, q_{accept}\}$$
$$V = \{A, B, K, A', R\}$$

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{start}$ | - | $q_{calcK}$ | $K \leftarrow 0$; $A_{temp} \leftarrow A$ | Initialize. |
| $q_{calcK}$ | $A_{temp} <$ NextBase$(A)$ | $q_{calcK}$ | $A_{temp} \leftarrow A_{temp} + 1$; $K \leftarrow K + 1$ | Count up to find gap $K$. |
| $q_{calcK}$ | $A_{temp} ==$ NextBase$(A)$ | $q_{decompose}$ | $A' \leftarrow A_{temp}$ | Gap found. Store new base $A'$. |

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{decompose}$ | $K > 0$ | $q_{decompose}$ | $B \leftarrow B - 1$; $K \leftarrow K - 1$ | Decompose $B$ by transferring $K$. |
| $q_{decompose}$ | $K == 0$ | $q_{recombine}$ | $R \leftarrow B$ | Remainder $R$ is what's left of $B$. |
| $q_{recombine}$ | - | $q_{accept}$ | Output $A' + R$ | Combine new base and remainder. |

**Choreography.** Decompression (splitting $B$) enables immediate compression (forming base $A'$).

**Lineage.** Elaborates Counting Up + Counting Down primitives; anticipates strategic boundary manipulation used later in Rounding, Chunking, Sliding.

---

## COBO (Counting On by Bases then Ones)

**Description.** For $A + B$, decompose $B = b \cdot Base + r$; iterate base jumps ($+Base$) then unit steps ($+1$).

**Machine.** $M_{COBO}$ with states $\{q_{start}, q_{bases}, q_{ones}, q_{accept}\}$ and registers $\{Sum, BaseCounter, OneCounter\}$.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{start}$ | - | $q_{initialize}$ | Read $A, B$ | Start. |
| $q_{initialize}$ | - | $q_{add\_bases}$ | $Sum \leftarrow A$; $BaseCounter \leftarrow B//Base$; $OneCounter \leftarrow B \pmod{Base}$ | Initialize Sum. Decompose $B$. |
| $q_{add\_bases}$ | $BaseCounter > 0$ | $q_{add\_bases}$ | $Sum \leftarrow Sum + Base$; $BaseCounter \leftarrow BaseCounter - 1$ | Add one Base unit (Loop). |

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{add\_bases}$ | $BaseCounter == 0$ | $q_{add\_ones}$ | - | All bases added. Transition. |
| $q_{add\_ones}$ | $OneCounter > 0$ | $q_{add\_ones}$ | $Sum \leftarrow Sum + 1$; $OneCounter \leftarrow OneCounter - 1$ | Add one unit (Loop). |
| $q_{add\_ones}$ | $OneCounter == 0$ | $q_{accept}$ | Output $Sum$ | All ones added. Accept. |

**Choreography.** Two-phase rhythm: compressed temporal blocks (bases) followed by decompressed fine resolution (ones).

**Lineage.** Builds on counting; prepares for Chunking and Rounding by habitualizing base jumps.

---

## Rounding and Adjusting (Addition)

**Description.** Select addend closer to next base: round up $A \to A' = A + K$, compute $A' + B$, then adjust back: $(A' + B) - K$.

**Machine.** States $\{q_{start}, q_{calcK}, q_{add}, q_{adjust}, q_{accept}\}$; registers $\{A, B, K, A', Temp, Result\}$.

**Transition Function ($\delta$):**

| Current State | Subroutine / Action | Next State | Interpretation |
|---|---|---|---|
| $q_{start}$ | Read $A, B$; Heuristic select $Target$ | $q_{calcK}$ | Start. Select number closer to the next base. |
| $q_{calcK}$ | **Count Up To Base(**$Target$**)** $\to K, A_{rounded}$ | $q_{add}$ | Determine $K$ by counting up from $Target$. |
| $q_{add}$ | **COBO(**$A_{rounded}$**, Other)** $\to TempSum$ | $q_{adjust}$ | Add Other to the rounded $A$. |
| $q_{adjust}$ | **Count Back(**$TempSum, K$**)** $\to Result$ | $q_{accept}$ | Adjust by counting back $K$. |

**Choreography.** Strategic temporal detour: initial decompression (deriving $K$) enables major compression (base addition), followed by inverse correction.

**Lineage.** Elaborates RMB (boundary anticipation) and COBO (base efficiency); introduces explicit compensation schema.

---

## Chunking (Addition)

**Description.** Decompose $B$ into large base chunk + strategic residual chunks to force successive bases: $B = B_{base} + K + R$ where $K$ bridges current sum to next base.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{init}$ | - | $q_{addBase}$ | $Sum \leftarrow A$; Decompose $B$ into $B_{base}, B_{ones}$ | Initialize Sum. Decompose $B$. |
| $q_{addBase}$ | - | $q_{calcK}$ | $Sum \leftarrow Sum + B_{base}$ | Add the entire base chunk at once. |
| $q_{calcK}$ | $Sum <$ NextBase($Sum$) | $q_{calcK}$ | $Sum \leftarrow Sum + 1$; $K \leftarrow K + 1$ | Iteratively find gap $K$ to next base. |
| $q_{calcK}$ | $Sum ==$ NextBase($Sum$) | $q_{applyK}$ | - | Gap found. |
| $q_{applyK}$ | $B_{ones} \geq K$ | $q_{calcK}$ | $Sum \leftarrow Sum + K$; $B_{ones} \leftarrow B_{ones} - K$ | Add strategic chunk $K$. Loop back. |
| $q_{applyK}$ | $B_{ones} < K$ | $q_{finishR}$ | - | Not enough ones for full chunk. |
| $q_{finishR}$ | - | $q_{accept}$ | $Sum \leftarrow Sum + B_{ones}$ | Add remaining residue. |

**Choreography.** Iterative cycle: (1) large compression via aggregated base, (2) micro decompression to find $K$, (3) re-compression to new base, (4) terminal residue.

**Lineage.** Synthesizes COBO (bulk bases) + RMB (strategic gap finding).

## Subtraction Chunking (Three Orientations)

Given $M - S = D$.

**A. Backwards by Part (Take-Away).** Sequentially subtract decomposed parts of $S$ (place value or strategic chunks) from $M$.

**B. Forwards from Part (Missing Addend).** Treat as $S + D = M$; Count Up (RMB logic) accumulating $D$.

**C. Backwards to Part (Distance Down To).** Count Back from $M$ toward $S$ using strategic base landings; accumulate distance.

Each orientation is a register machine. Below are the key transition schemas.

**A. Backwards by Part (Take-Away):** $V = \{CurrentValue, S_{rem}\}$

| State | Condition | Action |
|---|---|---|
| $q_{init}$ | - | $CurrentValue \leftarrow M$; $S_{rem} \leftarrow S$ |
| $q_{chunk}$ | $S_{rem} > 0$ | $Chunk \leftarrow$ Decompose($S_{rem}$); $CurrentValue \leftarrow CurrentValue - Chunk$; $S_{rem} \leftarrow S_{rem} - Chunk$ |
| $q_{chunk}$ | $S_{rem} == 0$ | Accept $CurrentValue$ |

**B. Forwards from Part (Missing Addend):** $V = \{CurrentValue, Distance\}$

| State | Condition | Action |
|---|---|---|
| $q_{init}$ | - | $CurrentValue \leftarrow S$; $Distance \leftarrow 0$ |
| $q_{chunk}$ | $CurrentValue < M$ | $Chunk \leftarrow$ CalcStrategicChunk($CurrentValue, M$); $CurrentValue \leftarrow CurrentValue + Chunk$; $Distance \leftarrow Distance + Chunk$ |
| $q_{chunk}$ | $CurrentValue == M$ | Accept $Distance$ |

**C. Backwards to Part (Distance Down To):** $V = \{CurrentValue, Distance\}$

| State | Condition | Action |
|---|---|---|
| $q_{init}$ | - | $CurrentValue \leftarrow M$; $Distance \leftarrow 0$ |
| $q_{chunk}$ | $CurrentValue > S$ | $Chunk \leftarrow$ CalcStrategicChunk$(CurrentValue, S)$; $CurrentValue \leftarrow CurrentValue - Chunk$; $Distance \leftarrow Distance + Chunk$ |
| $q_{chunk}$ | $CurrentValue == S$ | Accept $Distance$ |

**Choreography.** Orientation selects temporal direction; strategies B and C exploit boundary compression via RMB subroutines.

## Subtraction COBO / CBBO

**COBO (Missing Addend).** Start at $S$, perform base jumps toward $M$ (without overshoot), then ones; distance accumulated is $D$.

**CBBO (Counting Back).** Start at $M$, subtract base units (from decomposed $S$) then ones; final position is $D$.

**Machines.** Two dual register machines are defined.

**COBO (Missing Addend):** $V = \{CurrentValue, Distance, Target\}$

| State | Condition | Action |
|---|---|---|
| $q_{init}$ | - | $CurrentValue \leftarrow S$; $Distance \leftarrow 0$; $Target \leftarrow M$ |
| $q_{add\_bases}$ | $CurrentValue + Base \leq Target$ | $CurrentValue \leftarrow CurrentValue + Base$; $Distance \leftarrow Distance + Base$ |
| $q_{add\_bases}$ | $CurrentValue + Base > Target$ | transition to $q_{add\_ones}$ |
| $q_{add\_ones}$ | $CurrentValue < Target$ | $CurrentValue \leftarrow CurrentValue + 1$; $Distance \leftarrow Distance + 1$ |
| $q_{add\_ones}$ | $CurrentValue == Target$ | Accept $Distance$ |

**CBBO (Counting Back):** $V = \{CurrentValue, BaseCounter, OneCounter\}$

| State | Condition | Action |
|---|---|---|
| $q_{init}$ | - | $CurrentValue \leftarrow M$; Decompose $S$ into $BaseCounter, OneCounter$ |
| $q_{sub\_bases}$ | $BaseCounter > 0$ | $CurrentValue \leftarrow CurrentValue - Base$; $BaseCounter \leftarrow BaseCounter - 1$ |
| $q_{sub\_bases}$ | $BaseCounter == 0$ | transition to $q_{sub\_ones}$ |
| $q_{sub\_ones}$ | $OneCounter > 0$ | $CurrentValue \leftarrow CurrentValue - 1$; $OneCounter \leftarrow OneCounter - 1$ |
| $q_{sub\_ones}$ | $OneCounter == 0$ | Accept $CurrentValue$ |

**Choreography.** Directional inversion of the same two-phase rhythm (bases $\rightarrow$ ones). Overshoot detection acts as control boundary in COBO.

---

## Subtraction Decomposition (Borrowing)

**Description.** Left-to-right: subtract higher place (tens), detect insufficiency in lower place, decompose (borrow) one higher unit into base smaller units, then subtract ones.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{init}$ | - | $q_{sub\_bases}$ | Decompose $M, S$ into place values $R_T, R_O, S_T, S_O$. | Initialize registers. |
| $q_{sub\_bases}$ | - | $q_{check\_ones}$ | $R_T \leftarrow R_T - S_T$ | Subtract the bases (Tens). |
| $q_{check\_ones}$ | $R_O \geq S_O$ | $q_{sub\_ones}$ | - | Sufficient ones. No borrow needed. |
| $q_{check\_ones}$ | $R_O < S_O$ | $q_{decompose}$ | - | Insufficient ones. Borrow. |

| Current State | Condition | Next State | Action | Interpretation |
|---|---|---|---|---|
| $q_{decompose}$ | $R_T > 0$ | $q_{sub\_ones}$ | $R_T \leftarrow R_T - 1$; $R_O \leftarrow R_O + Base$ | Decompose (borrow) one ten. |
| $q_{sub\_ones}$ | - | $q_{accept}$ | $R_O \leftarrow R_O - S_O$; Result $\leftarrow R_T \cdot Base + R_O$ | Subtract ones and combine result. |

**Choreography.** Inversion of sublation: temporal decompression of a ten into ten ones to restore operability.

**Lineage.** Builds on internalized carry (from counting) now executed in reverse.

---

## Subtraction Rounding and Adjusting

**Description.** Dual rounding (e.g., $M \to M'$ down, $S \to S'$ down) yields simplified $M' - S'$, then contrasting compensations: add $K_M$, subtract $K_S$.

**Transition Function ($\delta$):**

| Current State | Action | Next State | Interpretation |
|---|---|---|---|
| $q_{start}$ | Read $M, S$ | $q_{roundM}$ | Start. |
| $q_{roundM}$ | $M' \leftarrow$ RoundDown($M$); $K_M \leftarrow M - M'$ | $q_{roundS}$ | Round $M$ down. Store adjustment $K_M$. |
| $q_{roundS}$ | $S' \leftarrow$ RoundDown($S$); $K_S \leftarrow S - S'$ | $q_{subtract}$ | Round $S$ down. Store adjustment $K_S$. |
| $q_{subtract}$ | $Temp \leftarrow M' - S'$ | $q_{adjustM}$ | Calculate intermediate result. |
| $q_{adjustM}$ | $Temp \leftarrow Temp + K_M$ | $q_{adjustS}$ | Compensate for $M$ (Add back). |
| $q_{adjustS}$ | $Result \leftarrow Temp - K_S$ (via chunking) | $q_{accept}$ | Compensate for $S$ (Subtract). |

**Choreography.** Opposed adjustments highlight subtraction asymmetry: modification of minuend vs. subtrahend impacts result in inverse directions.

**Lineage.** Integrates rounding (addition strategy) and inverse compensation sequencing.

---

## Subtraction Sliding (Constant Difference)

**Description.** Find $K$ so that $S + K$ is a base (or friendly) number; compute $(M + K) - (S + K)$ exploiting invariance: $M - S = (M + K) - (S + K)$.

**Transition Function ($\delta$):**

| Current State | Action | Next State | Interpretation |
|---|---|---|---|
| $q_{start}$ | Read $M, S$ | $q_{calcK}$ | Start. Target $S$ for adjustment. |
| $q_{calcK}$ | $K \leftarrow$ CountUpToBase($S$) | $q_{slide}$ | Iteratively find the gap $K$. |
| $q_{slide}$ | $M' \leftarrow M + K$; $S' \leftarrow S + K$ | $q_{subtract}$ | Apply the slide $K$ to both $M$ and $S$. |
| $q_{subtract}$ | $Result \leftarrow M' - S'$ | $q_{accept}$ | Perform the simplified subtraction. |

**Choreography.** Up-front decompression (deriving $K$) enables single compressed subtraction against a base-aligned subtrahend.

**Lineage.** Extends RMB gap-finding; anticipates relational "distance" framing central to subtraction fluency.

---

## Commutative Reasoning (Multiplication Optimization)

**Description.** For $A \times B$, evaluate heuristic difficulty of $(A, B)$ vs $(B, A)$; select orientation minimizing cognitive load (iteration count & skip difficulty), then perform iterative addition (skip counting).

**Transition Function ($\delta$):**

| Current State | Condition / Heuristic | Next State | Action |
|---|---|---|---|
| $q_{evaluate}$ | $H(B, A) < H(A, B)$ | $q_{repackage}$ | - |

| Current State | Condition / Heuristic | Next State | Action |
|---|---|---|---|
| $q_{evaluate}$ | (Otherwise) | $q_{calc}$ | $Groups \leftarrow A$; $Items \leftarrow B$ |
| $q_{repackage}$ | - | $q_{calc}$ | $Groups \leftarrow B$; $Items \leftarrow A$ |
| $q_{calc}$ | - | $q_{accept}$ | $Total \leftarrow$ IterativeAdd$(Groups, Items)$ |

**Choreography.** Meta-level selection precedes execution; commutative symmetry exploited for temporal compression.

**Lineage.** Builds on C2C / Skip Counting; introduces optimization layer.

## Coordinating Two Counts (C2C)

**Description.** Foundational multiplication: nested counting—items within group, groups within total; total $T = N \cdot S$ emerges from exhaustive unit enumeration.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action |
|---|---|---|---|
| $q_{init}$ | - | $q_{checkG}$ | $G \leftarrow 0, I \leftarrow 0, T \leftarrow 0$ |
| $q_{checkG}$ | $G < N$ | $q_{countItems}$ | - |
| $q_{checkG}$ | $G == N$ | $q_{accept}$ | Output $T$ |
| $q_{countItems}$ | $I < S$ | $q_{countItems}$ | $I \leftarrow I + 1, T \leftarrow T + 1$ |
| $q_{countItems}$ | $I == S$ | $q_{nextGroup}$ | - |
| $q_{nextGroup}$ | - | $q_{checkG}$ | $G \leftarrow G + 1, I \leftarrow 0$ |

**Choreography.** Maximal temporal decompression (no compression yet); establishes structural scaffold for later compression (skip counting, distributive reasoning).

**Lineage.** Direct elaboration of counting primitives into nested loops.

## Conversion to Bases and Ones (CBO Multiplication)

**Description.** Redistribute units among groups so that many groups become exact base multiples, leaving a compact residual: $(k \cdot Base) + r$.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action |
|---|---|---|---|
| $q_{init}$ | - | $q_{select\_source}$ | Initialize $Groups$ array with value $S$. |
| $q_{select\_source}$ | $N > 0$ | $q_{transfer}$ | Select a $SourceIdx$. |
| $q_{transfer}$ | $Groups[Source] > 0$ AND not all targets full | $q_{transfer}$ | Transfer 1 unit from $Source$ to next available $Target$. |
| $q_{transfer}$ | (Source empty OR all targets full) | $q_{finalize}$ | - |
| $q_{finalize}$ | - | $q_{accept}$ | Total $\leftarrow \sum Groups$. |

**Choreography.** Proactive sublation: simultaneous decompression (source group) and compression (targets) to manufacture base units early.

**Lineage.** Multiplicative analogue of RMB and addition Chunking with explicit inter-group transfers.

---

## Distributive Reasoning (Multiplication)

**Description.** Decompose $S = S_1 + S_2$ (heuristically "easy" numbers), compute $NS_1$ and $NS_2$ (skip counting or compressed methods), then sum.

**Transition Function ($\delta$):**

| Current State | Action | Next State |
|---|---|---|
| $q_{split}$ | $S_1, S_2 \leftarrow \text{HeuristicSplit}(S)$ | $q_{P1}$ |
| $q_{P1}$ | $P_1 \leftarrow \text{IterativeAdd}(N, S_1)$ | $q_{P2}$ |
| $q_{P2}$ | $P_2 \leftarrow \text{IterativeAdd}(N, S_2)$ | $q_{sum}$ |
| $q_{sum}$ | $Total \leftarrow P_1 + P_2$ | $q_{accept}$ |

**Choreography.** Temporal decompression (factor split) followed by parallelizable compressed sub-calculations and final recombination.

**Lineage.** Extends skip counting with heuristic structural decomposition; precursor to algebraic distributivity recognition.

---

## Dealing by Ones (Division – Sharing)

**Description.** Partitive division: distribute single units round-robin into $N$ groups until total $T$ exhausted; per-group size $S$ emerges.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action |
|---|---|---|---|
| $q_{init}$ | - | $q_{deal}$ | $Remaining \leftarrow T$; Initialize $Groups$ array to 0s. |
| $q_{deal}$ | $Remaining > 0$ | $q_{deal}$ | $Groups[idx] \leftarrow Groups[idx] + 1$; $Remaining \leftarrow Remaining - 1$; $idx \leftarrow (idx + 1) \pmod N$ |
| $q_{deal}$ | $Remaining == 0$ | $q_{accept}$ | Output $Groups[0]$ |

**Choreography.** Maximal temporal decompression; rhythmic rounds establish invariant increase pattern (foundation for later compression insights).

**Lineage.** Inversion of C2C perspective (constructing equal groups from total rather than composing total from groups).

---

## Inverse Distributive Reasoning (Division)

**Description.** Measurement division $T/S$: decompose $T$ into known multiples of $S$: $T = \sum_i (m_i S)$; quotient $= \sum_i m_i$.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action |
|---|---|---|---|
| $q_{init}$ | - | $q_{search}$ | $Remaining \leftarrow T$; $TotalQ \leftarrow 0$; Load KB for $S$. |
| $q_{search}$ | Found $(P_T, P_Q)$ in KB where $P_T \leq Remaining$ | $q_{apply}$ | Select largest such $(P_T, P_Q)$. |
| $q_{search}$ | No suitable fact found | $q_{accept}$ | Output $TotalQ$. |
| $q_{apply}$ | - | $q_{search}$ | $Remaining \leftarrow Remaining - P_T$; $TotalQ \leftarrow TotalQ + P_Q$. |

**Choreography.** Temporal compression via retrieval of pre-compressed multiplication facts; loop greedily subtracts largest available chunk.

**Lineage.** Inversion of Distributive Reasoning in multiplication (switch from constructing product to decomposing dividend).

---

## Using Commutative Reasoning (Division via Iterated Accumulation)

**Description.** For $E/G$ (sharing reframed as measurement): iteratively accumulate $G$ until total $E$ reached; iteration count is quotient.

**Transition Function ($\delta$):**

| Current State | Condition | Next State | Action |
|---|---|---|---|
| $q_{init}$ | - | $q_{iterate}$ | $Acc \leftarrow 0$; $Q \leftarrow 0$. |
| $q_{iterate}$ | - | $q_{check}$ | $Acc \leftarrow Acc + G$; $Q \leftarrow Q + 1$. |
| $q_{check}$ | $Acc < E$ | $q_{iterate}$ | - |
| $q_{check}$ | $Acc == E$ | $q_{accept}$ | Output $Q$. |

**Choreography.** Symmetric inversion of repeated addition (multiplication) focusing on completion criterion instead of fixed loop count.

**Lineage.** Bridges between Dealing by Ones and chunk-based division (fact retrieval).

---

## Conversion to Groups Other than Bases (CGOB Division)

**Description.** Leverage base decomposition of dividend $T$ (e.g., tens & ones) plus analysis of base/divisor relation: $Base = q_1 S + r_1$; process all base units in bulk, aggregate remainders, finalize.

**Transition Function ($\delta$):**

| Current State | Action | Next State |
|---|---|---|
| $q_{init}$ | Decompose $T$ into $T_B, T_O$; $Q \leftarrow 0, R \leftarrow 0$. | $q_{analyze}$ |
| $q_{analyze}$ | $S_{inB} \leftarrow B//S$; $R_{inB} \leftarrow B \pmod S$. | $q_{processBases}$ |
| $q_{processBases}$ | $Q \leftarrow Q + T_B \cdot S_{inB}$; $R \leftarrow R + T_B \cdot R_{inB}$. | $q_{combineR}$ |
| $q_{combineR}$ | $R \leftarrow R + T_O$. | $q_{processR}$ |
| $q_{processR}$ | $Q \leftarrow Q + R//S$; $R \leftarrow R \pmod S$. | $q_{accept}$ |

**Choreography.** Dual decompression (dividend by base, base by divisor) → large compression (bulk quotient) → residual resolution.

**Lineage.** Division analogue of CBO (multiplication) and Distributive Reasoning; integrates multi-level structural analysis.

---

## Conceptual Dependency Graph (Narrative)

Counting → (RMB, COBO) → (Chunking, Rounding & Adjusting, Sliding) → (Subtraction Inversions: COBO/CBBO, Chunking orientations, Decomposition, Rounding, Sliding) → (C2C) → (Skip Counting / implicit in COBO Multiplication) → (Commutative & Distributive Reasoning, CBO Multiplication) → (Division primitives: Dealing by Ones, Iterated Accumulation) → (Inverse Distributive Reasoning, Fact-Based Decomposition) → (CGOB Division).

Each arrow denotes an algorithmic elaboration where prior compressed units or reversible decompositions become callable subroutines.

---

## Temporal Dynamics Summary

- **Primitive Decompression:** Counting by ones; Dealing by Ones; C2C (inner loop).

- **First Compression Layer:** COBO (bases as units); subtraction COBO/CBBO; iterative accumulation for division.
- **Strategic Boundary Forcing:** RMB, Chunking, Sliding, Rounding (anticipatory manipulation of base thresholds).
- **Structural Decomposition / Synthesis:** Distributive Reasoning, Inverse Distributive Reasoning, CBO (Multiplication & Division), CGOB.

---

## Glossary of Symbols

- $Base$: Typically 10 (extendable to other positional bases).
- $K$: Gap to next base (RMB, Rounding, Chunking, Sliding).
- $R$: Remainder after decomposition or partial processing.
- $S_1, S_2$: Split components of a factor (Distributive Reasoning).
- $Groups, Items$: Multiplicative roles after commutative optimization.
- $Remaining$: Unprocessed portion of a dividend in division strategies.
- $Q$: Quotient / accumulated result in division; also generic state set symbol context-dependent.
- $Acc$: Accumulated total during iterative division.

---

End of Clean Draft.