Addition Strategies: Adding Bases and Adding Ones

Compiled by: Theodore M. Savich

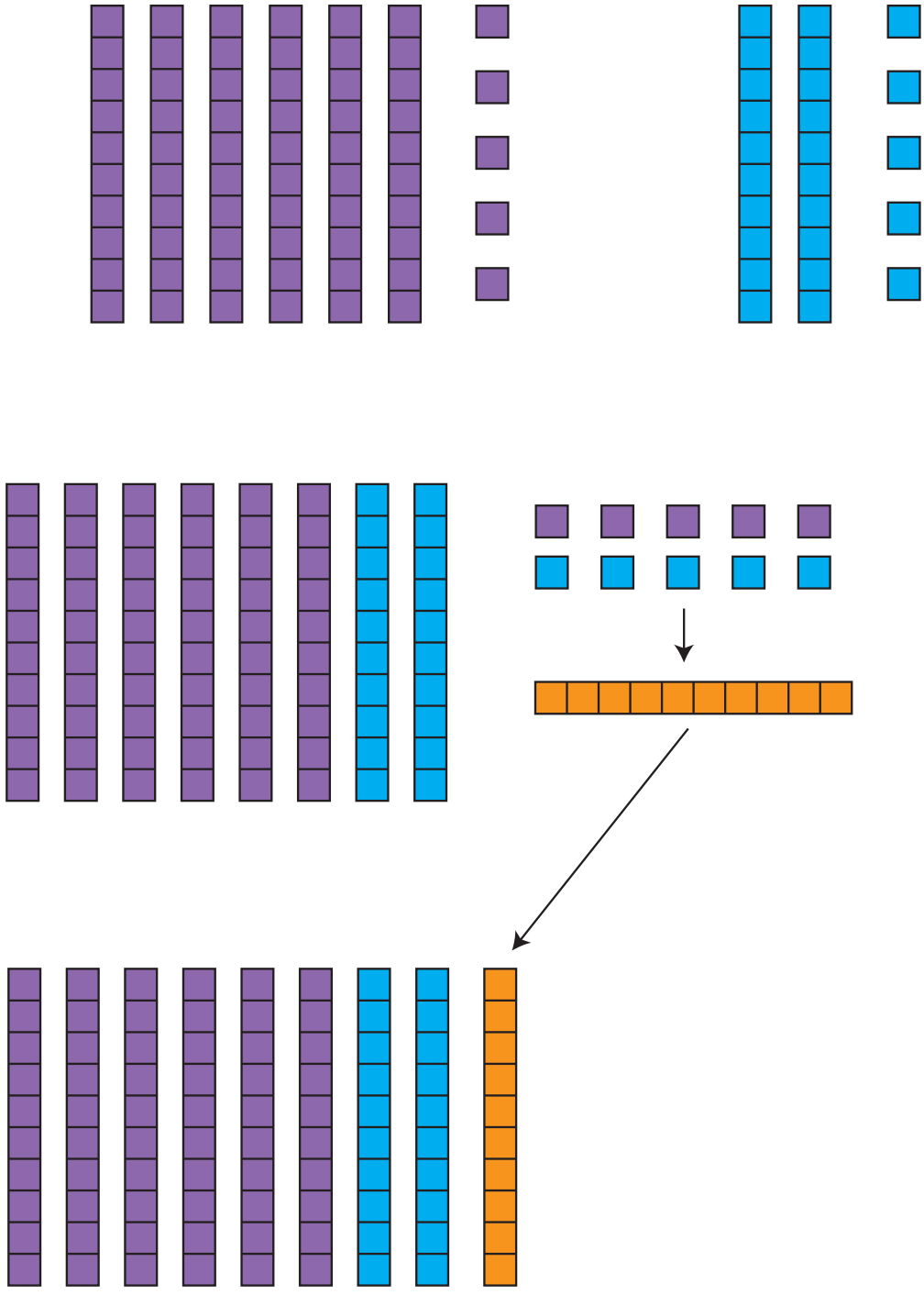
March 28, 2025

Transcript

Strategy descriptions and examples adapted from Hackenberg (2025). The video for this student's strategy was not a CGI video and has been removed from publicly accessible databases.

It involved a student named Sarah, who solved 65+25. She said the following:

Sarah's solution: "I used decomposing, I broke 65 into 60 and five. I broke 25 into 20 and five. I added the 60 and the 20 and I got 80. I added the 5 to 5 and I got ten. I connected the 5 to the 80 and I got 90."



Notation Representing Sarah's Solution:

$$\begin{aligned} 65 + 25 &= \square \\ 60 + 20 &= 80 \\ 5 + 5 &= 10 \\ 80 + 10 &= 90 \end{aligned}$$

Description of Strategy

- **Objective:** Split both addends into bases and ones, add bases together and ones together, then combine the partial sums.
- **Example:** $65 + 25$
  - Split: $65 = 60 + 5$, $25 = 20 + 5$.
  - Add bases: $60 + 20 = 80$.
  - Add ones: $5 + 5 = 10$.
  - Combine: $80 + 10 = 90$.

Automaton Type

Pushdown Automaton (PDA): Needed to handle composition-over when adding ones.

Formal Description of the Automaton

We define the PDA as the 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_{0/accept}, Z_0, F)$$

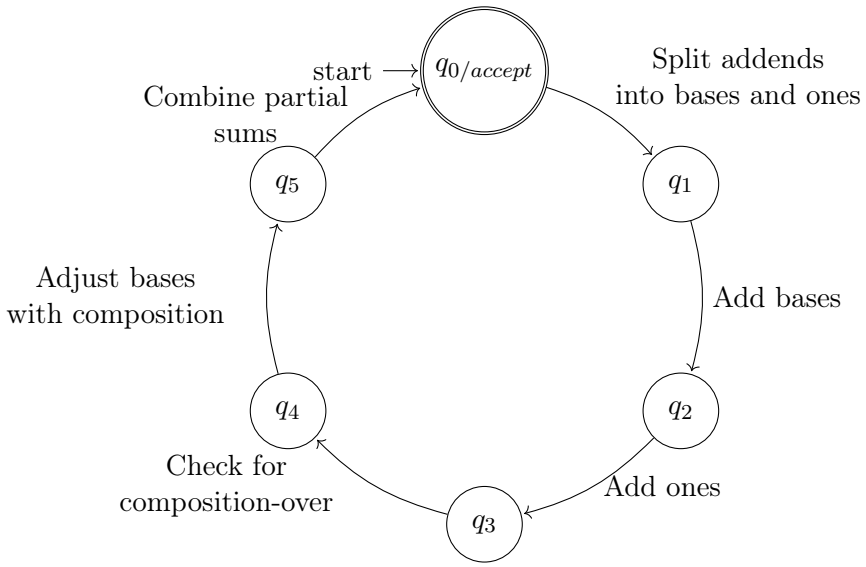
where:

- $Q = \{q_{0/accept}, q_1, q_2, q_3, q_4, q_5\}$ is the set of states.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +\}$ is the input alphabet.
- $\Gamma = \{Z_0\} \cup \{c \mid c \in \mathbb{N}\}$ is the stack alphabet, where Z_0 is the initial stack symbol and a symbol c represents a composition-over.
- $q_{0/accept}$ is the start state (which is also the accept state).
- Z_0 is the initial stack symbol.
- $F = \{q_{0/accept}\}$ is the set of accepting states.

The transition function δ is defined as follows:

1. $\delta(q_{0/accept}, \text{“}A, B\text{”}, Z_0) = \{(q_1, Z_0)\}$
   (Read A and B and split each into its base (tens, hundreds, ...) and ones components.)
2. $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$
   (Add the bases: compute $A_{\text{base}} + B_{\text{base}}$.)
3. $\delta(q_2, \varepsilon, Z_0) = \{(q_3, Z_0)\}$
   (Add the ones: compute $A_{\text{ones}} + B_{\text{ones}}$.)
4. $\delta(q_3, \varepsilon, Z_0) = \{(q_4, c\,Z_0)\}$
   (If the ones sum is greater than or equal to the base, push the composition c onto the stack.)
5. $\delta(q_4, \varepsilon, c) = \{(q_5, c)\}$
   (Adjust the bases sum by adding the composition-over c.)
6. $\delta(q_5, \varepsilon, Z_0) = \{(q_{0/accept}, Z_0)\}$
   (Combine the adjusted bases sum with the ones sum and output the final result.)

Automaton Diagram for ABAO



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Addition Strategies: Adding Bases and Adding Ones (ABAO)</title>
    <style>
        body { font-family: sans-serif; }
        #abaoDiagram { border: 1px solid #d3d3d3; }
        #outputContainer { margin-top: 20px; }
        .diagram-label { font-size: 14px; display: block; margin-bottom: 5px; }
        .calc-label { font-size: 12px; text-anchor: middle; }
        .group-rect { fill: none; stroke: black; stroke-dasharray: 5 5; stroke-width: 1; }
        .arrow { fill: none; stroke: black; stroke-width: 2; }
        .arrow-head { fill: black; stroke: black; }
    </style>
</head>
<body>

    <h1>Addition Strategies: Adding Bases and Adding Ones (ABAO)</h1>

    <div>
        <label for="abaoAddend1">Addend 1:</label>
        <input type="number" id="abaoAddend1" value="65"> <!-- Changed default back -->
    </div>
    <div>
        <label for="abaoAddend2">Addend 2:</label>
        <input type="number" id="abaoAddend2" value="25"> <!-- Changed default back -->
    </div>

    <button onclick="runABAOAutomaton()">Calculate and Visualize</button>

    <div id="outputContainer">
        <h2>Explanation:</h2>
        <div id="abaoOutput">
            <strong>Current Addends:</strong> 65+25<br> <!-- Initial content -->
        </div>
    </div>

    <h2>Diagram:</h2>
    <svg id="abaoDiagram" width="700" height="950"></svg> <!-- Increased height
        significantly -->

    <script>
    // --- Helper SVG Functions --- (Keep these the same as the previous version) ---
    function drawBlock(svg, x, y, width, height, fill) {
        const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
        rect.setAttribute('x', x);
        rect.setAttribute('y', y);
        rect.setAttribute('width', width);
        rect.setAttribute('height', height);
        rect.setAttribute('fill', fill);
        rect.setAttribute('stroke', 'black');
        rect.setAttribute('stroke-width', '0.5'); // Thinner lines for blocks
```

```
        svg.appendChild(rect);
    }

    function drawTenBlock(svg, x, y, width, height, fill, unitBlockSize) {
        const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
        const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", '
            rect');
        backgroundRect.setAttribute('x', x);
        backgroundRect.setAttribute('y', y);
        backgroundRect.setAttribute('width', width);
        backgroundRect.setAttribute('height', height);
        backgroundRect.setAttribute('fill', fill);
        backgroundRect.setAttribute('stroke', 'black');
        backgroundRect.setAttribute('stroke-width', '1');
        group.appendChild(backgroundRect);

        for (let i = 0; i < 10; i++) {
            const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", 'rect
                ');
            unitBlock.setAttribute('x', x);
            unitBlock.setAttribute('y', y + i * unitBlockSize);
            unitBlock.setAttribute('width', unitBlockSize);
            unitBlock.setAttribute('height', unitBlockSize);
            unitBlock.setAttribute('fill', fill);
            unitBlock.setAttribute('stroke', 'lightgrey');
            unitBlock.setAttribute('stroke-width', '0.5');
            group.appendChild(unitBlock);
        }
        svg.appendChild(group);
    }

    function drawHundredBlock(svg, x, y, size, fill, unitBlockSize) {
        const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
        const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", '
            rect');
        backgroundRect.setAttribute('x', x);
        backgroundRect.setAttribute('y', y);
        backgroundRect.setAttribute('width', size);
        backgroundRect.setAttribute('height', size);
        backgroundRect.setAttribute('fill', fill);
        backgroundRect.setAttribute('stroke', 'black');
        backgroundRect.setAttribute('stroke-width', '1');
        group.appendChild(backgroundRect);

        for (let row = 0; row < 10; row++) {
            for (let col = 0; col < 10; col++) {
                const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
                    rect');
                unitBlock.setAttribute('x', x + col * unitBlockSize);
                unitBlock.setAttribute('y', y + row * unitBlockSize);
                unitBlock.setAttribute('width', unitBlockSize);
                unitBlock.setAttribute('height', unitBlockSize);
                unitBlock.setAttribute('fill', fill);
                unitBlock.setAttribute('stroke', 'lightgrey');
```

```
                unitBlock.setAttribute('stroke-width', '0.5');
                group.appendChild(unitBlock);
            }
        }
        svg.appendChild(group);
    }


    function drawGroupRect(svg, x, y, width, height) {
        const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
        rect.setAttribute('x', x);
        rect.setAttribute('y', y);
        rect.setAttribute('width', width);
        rect.setAttribute('height', height);
        rect.setAttribute('class', 'group-rect');
        svg.appendChild(rect);
    }

    function createText(svg, x, y, textContent, className = 'diagram-label', anchor = '
        start') {
        const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
        text.setAttribute('x', x);
        text.setAttribute('y', y);
        text.setAttribute('class', className);
        text.setAttribute('text-anchor', anchor);
        // text.setAttribute('font-size', '14px'); // Use CSS
        text.textContent = textContent;
        svg.appendChild(text);
    }

    function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy, arrowClass='arrow', headClass
        ='arrow-head') {
        const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}`);
        path.setAttribute('class', arrowClass);
        svg.appendChild(path);

        const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        const arrowSize = 5;
         // Calculate angle at the end of the curve (approx)
        const dx = x2 - cx;
        const dy = y2 - cy;
        const angleRad = Math.atan2(dy, dx);
        const angleDeg = angleRad * (180 / Math.PI);
        arrowHead.setAttribute('d', `M 0 0 L ${arrowSize} ${arrowSize/2} L ${arrowSize} $
            {-arrowSize/2} Z`);
        arrowHead.setAttribute('class', headClass);
        arrowHead.setAttribute('transform', `translate(${x2}, ${y2}) rotate(${angleDeg +
            180})`);
        svg.appendChild(arrowHead);
    }
    // --- End Helper Functions ---
```

```
152 function drawABAODiagram(svgId, a1, a2, hunsA1, tensA1, onesA1, hunsA2, tensA2,
        onesA2,
153                         initialHunsSum, initialTensSum, initialOnesSum,
154                         onesCarry, tensCarry,
155                         finalHunsSum, finalTensSum, finalOnesSum, finalSum)
156 {
157     const svg = document.getElementById(svgId);
158     if (!svg) return;
159     svg.innerHTML = ''; // Clear SVG
160
161     const svgWidth = parseFloat(svg.getAttribute('width'));
162     const svgHeight = parseFloat(svg.getAttribute('height'));
163     const blockUnitSize = 10;
164     const tenBlockWidth = blockUnitSize;
165     const tenBlockHeight = blockUnitSize * 10;
166     const hundredBlockSize = blockUnitSize * 10;
167     const blockSpacing = 4;
168     const groupSpacingX = 30;
169     const sectionSpacingY = 140; // Increased spacing slightly
170     const startX = 30;
171     let currentY = 40;
172     const colorA1 = 'purple';
173     const colorA2 = 'cyan';
174     const colorOnesCarry = 'orange';
175     const colorTensCarry = 'lightgreen';
176     const maxBlockHeight = Math.max(tenBlockHeight, hundredBlockSize, blockUnitSize);
177     const calcLabelYOffset = 20; // Offset below blocks for calc labels
178     const textHeightApproximation = 10; // Approximate height of text for arrow start
            Y
179
180     // --- 1. Initial Split Visualization ---
181     createText(svg, startX, currentY, `Initial Split: ${a1} = ${hunsA1 > 0 ? hunsA1 +
            '+': ''}${tensA1}+${onesA1}, ${a2} = ${hunsA2 > 0 ? hunsA2 + '+' : ''}${tensA2
            }+${onesA2}`);
182     currentY += 30;
183
184     let currentX = startX;
185     let section1MaxY = currentY;
186
187     // A1 Blocks
188     for (let i = 0; i < hunsA1 / 100; i++) { drawHundredBlock(svg, currentX, currentY,
             hundredBlockSize, colorA1, blockUnitSize); currentX += hundredBlockSize +
            groupSpacingX; section1MaxY = Math.max(section1MaxY, currentY +
            hundredBlockSize); }
189     for (let i = 0; i < tensA1 / 10; i++) { drawTenBlock(svg, currentX, currentY,
            tenBlockWidth, tenBlockHeight, colorA1, blockUnitSize); currentX +=
            tenBlockWidth + blockSpacing; section1MaxY = Math.max(section1MaxY, currentY +
             tenBlockHeight); }
190     for (let i = 0; i < onesA1; i++) { drawBlock(svg, currentX, currentY +
            maxBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, colorA1);
            currentX += blockUnitSize + blockSpacing; section1MaxY = Math.max(section1MaxY
            , currentY + maxBlockHeight); }
191     const a1EndX = currentX;
192
```

```
193         // A2 Blocks
194         currentX = a1EndX + groupSpacingX * 2;
195         const a2StartX = currentX;
196         for (let i = 0; i < hunsA2 / 100; i++) { drawHundredBlock(svg, currentX, currentY,
                 hundredBlockSize, colorA2, blockUnitSize); currentX += hundredBlockSize +
                groupSpacingX; section1MaxY = Math.max(section1MaxY, currentY +
                hundredBlockSize); }
197         for (let i = 0; i < tensA2 / 10; i++) { drawTenBlock(svg, currentX, currentY,
                tenBlockWidth, tenBlockHeight, colorA2, blockUnitSize); currentX +=
                tenBlockWidth + blockSpacing; section1MaxY = Math.max(section1MaxY, currentY +
                 tenBlockHeight); }
198         for (let i = 0; i < onesA2; i++) { drawBlock(svg, currentX, currentY +
                maxBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, colorA2);
                currentX += blockUnitSize + blockSpacing; section1MaxY = Math.max(section1MaxY
                , currentY + maxBlockHeight); }
199         currentY = section1MaxY + sectionSpacingY;
200
201
202         // --- 2. Combine Like Units (Before Composition) ---
203         createText(svg, startX, currentY, `Combine Like Units`);
204         currentY += 30;
205
206         let section2MaxY = currentY;
207         let combinedHunsX = startX;
208         let combinedTensX = 0;
209         let combinedOnesX = 0;
210         let hunsEndX = startX;
211         let tensEndX = 0;
212         let onesEndX = 0;
213         let onesGroupEndX = 0; // For composition grouping rect later
214         let tensGroupStartX = 0; // For composition grouping rect later
215         let tensGroupEndX = 0; // For composition grouping rect later
216
217
218         // Draw Combined Hundreds
219         if(initialHunsSum > 0) {
220             for (let i = 0; i < initialHunsSum / 100; i++) { let color = (i < hunsA1 /
                    100) ? colorA1 : colorA2; drawHundredBlock(svg, combinedHunsX, currentY,
                    hundredBlockSize, color, blockUnitSize); combinedHunsX += hundredBlockSize
                     + blockSpacing; }
221             hunsEndX = combinedHunsX;
222             createText(svg, startX + (hunsEndX - startX - blockSpacing) / 2, currentY +
                    hundredBlockSize + calcLabelYOffset, `${hunsA1}+${hunsA2}=${initialHunsSum
                    }`, 'calc-label', 'middle');
223             section2MaxY = Math.max(section2MaxY, currentY + hundredBlockSize);
224             combinedTensX = hunsEndX + groupSpacingX;
225         } else {
226             combinedTensX = startX;
227         }
228
229         // Draw Combined Tens
230         tensGroupStartX = combinedTensX; // Mark start for potential grouping
231         currentX = combinedTensX;
232         for (let i = 0; i < initialTensSum / 10; i++) {
```

```
            let color = (i < tensA1 / 10) ? colorA1 : colorA2;
            drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, color,
                blockUnitSize);
             if (i < 10) tensGroupEndX = currentX + tenBlockWidth; // Track end of first
                 10 tens
            currentX += tenBlockWidth + blockSpacing;
        }
        tensEndX = currentX;
        const tensLabelX = combinedTensX + (tensEndX - combinedTensX - blockSpacing) / 2;
        const tensLabelY = currentY + tenBlockHeight + calcLabelYOffset;
        createText(svg, tensLabelX, tensLabelY, `${tensA1}+${tensA2}=${initialTensSum}`, '
            calc-label', 'middle');
        section2MaxY = Math.max(section2MaxY, currentY + tenBlockHeight);


        // Draw Combined Ones
        combinedOnesX = tensEndX + groupSpacingX;
        currentX = combinedOnesX;
        for (let i = 0; i < initialOnesSum; i++) {
            let color = (i < onesA1) ? colorA1 : colorA2;
            drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize,
                blockUnitSize, blockUnitSize, color);
            if (i < 10) onesGroupEndX = currentX + blockUnitSize; // Track end of first 10
                 ones
            currentX += blockUnitSize + blockSpacing;
        }
        onesEndX = currentX;
        const onesLabelX = combinedOnesX + (onesEndX - combinedOnesX - blockSpacing) / 2;
        const onesLabelY = currentY + maxBlockHeight + calcLabelYOffset;
        createText(svg, onesLabelX, onesLabelY, `${onesA1}+${onesA2}=${initialOnesSum}`, '
            calc-label', 'middle');
        section2MaxY = Math.max(section2MaxY, currentY + maxBlockHeight);


        // --- Store Coordinates for Arrows ---
        const onesArrowStartY = onesLabelY + textHeightApproximation; // Start below the
            ones calculation text
        const tensArrowStartY = tensLabelY + textHeightApproximation; // Start below the
            tens calculation text


        // --- 3. Skip separate composition step, move Y ---
        currentY = section2MaxY + sectionSpacingY;


        // --- 4. Final Sum Visualization ---
        createText(svg, startX, currentY, `Final Result (After Composition): ${finalSum}`)
            ;
        currentY += 30;

        let finalMaxY = currentY;
        currentX = startX;
        let finalHunsStartX = startX;
        let finalTensStartX = 0;
```

```
278         let finalOnesStartX = 0;
279
280
281         // Final Hundreds
282         let composedHundredX = 0, composedHundredY = 0;
283         for (let i = 0; i < finalHunsSum / 100; i++) {
284             let color;
285             if (i < hunsA1 / 100) color = colorA1;
286             else if (i < initialHunsSum / 100) color = colorA2;
287             else {
288                 color = colorTensCarry; // Color for hundred composed from tens
289                 composedHundredX = currentX + hundredBlockSize / 2; // Store center of
                        composed hundred
290                 composedHundredY = currentY + hundredBlockSize / 2;
291             }
292             drawHundredBlock(svg, currentX, currentY, hundredBlockSize, color,
                    blockUnitSize);
293             currentX += hundredBlockSize + blockSpacing;
294         }
295         let finalHunsEndX = currentX > startX ? currentX - blockSpacing : startX;
296
297         // Final Tens
298         currentX = finalHunsEndX + (finalHunsEndX > startX ? groupSpacingX : 0);
299         finalTensStartX = currentX; // Store start X for final tens
300         let composedTenX = 0, composedTenY = 0;
301         for (let i = 0; i < finalTensSum / 10; i++) {
302             let color = colorA1; // Default/placeholder color
303              // More precise coloring: Check if this ten block is the one created by
                    onesCarry
304             if (onesCarry > 0 && i === initialTensSum / 10) { // If it's the position
                    right after initial tens
305                 color = colorOnesCarry;
306                 composedTenX = currentX + tenBlockWidth / 2; // Store center of composed
                        ten
307                 composedTenY = currentY + tenBlockHeight / 2;
308             } else if (i < tensA1 / 10 && tensCarry == 0) { // Original A1 if no tens->
                    hundred carry
309                 color = colorA1;
310             } else if (i < initialTensSum / 10 && tensCarry == 0) { // Original A2 if no
                    tens->hundred carry
311                  color = colorA2;
312             }
313             // If tensCarry happened, coloring remaining tens accurately is complex,
                    using carry color as fallback
314             else if (tensCarry > 0) {
315                 color = colorOnesCarry; // Might be remaining original or from ones carry
316             }
317
318             drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, color,
                    blockUnitSize);
319             currentX += tenBlockWidth + blockSpacing;
320         }
321         let finalTensEndX = currentX > finalTensStartX ? currentX - blockSpacing :
                finalTensStartX;
```

```
322
323         // Final Ones Blocks
324         currentX = finalTensEndX + (finalTensEndX > finalTensStartX ? groupSpacingX : 0);
325         finalOnesStartX = currentX;
326         for (let i = 0; i < finalOnesSum; i++) {
327             let color = (i < onesA1 && onesCarry == 0) ? colorA1 : colorA2;
328             drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize,
                    blockUnitSize, blockUnitSize, color);
329             currentX += blockUnitSize + blockSpacing;
330         }
331         finalMaxY = Math.max(currentY + maxBlockHeight, currentY + hundredBlockSize);
332
333         // --- Draw Composition Arrows ---
334         // Arrow from ones sum text to composed ten block
335         if (onesCarry > 0 && composedTenX > 0) {
336             createCurvedArrow(svg,
337                 onesLabelX, onesArrowStartY, // Start below ones calculation text
338                 composedTenX, composedTenY - tenBlockHeight/2, // End at top-center of
                        composed ten block
339                 onesLabelX + 30, onesArrowStartY + sectionSpacingY / 2 // Control point
340             );
341         }
342         // Arrow from tens sum text to composed hundred block
343          if (tensCarry > 0 && composedHundredX > 0) {
344              createCurvedArrow(svg,
345                 tensLabelX, tensArrowStartY, // Start below tens calculation text
346                 composedHundredX, composedHundredY - hundredBlockSize/2, // End at top-
                        center of composed hundred block
347                 tensLabelX + 50, tensArrowStartY + sectionSpacingY / 2 // Control point
348             );
349          }
350     }
351
352
353     (function() { // IIFE
354         window.runABAOAutomaton = function() {
355             const outputDiv = document.getElementById('abaoOutput');
356             const a1 = parseInt(document.getElementById('abaoAddend1').value);
357             const a2 = parseInt(document.getElementById('abaoAddend2').value);
358
359             if (isNaN(a1) || isNaN(a2)) {
360                 outputDiv.textContent = "Please␣enter␣valid␣numbers␣for␣both␣addends";
361                 diagramABAOSVG.innerHTML = ''; // Clear diagram on error
362                 return;
363             }
364
365             let steps = '';
366
367             // Split both addends
368             const hunsA1 = Math.floor(a1 / 100) * 100;
369             const tensA1 = Math.floor((a1 % 100) / 10) * 10;
370             const onesA1 = a1 % 10;
371             const hunsA2 = Math.floor(a2 / 100) * 100;
372             const tensA2 = Math.floor((a2 % 100) / 10) * 10;
```

```
373             const onesA2 = a2 % 10;
374             steps += '<strong>Splitting␣Addends:</strong><br>';
375             steps += `${a1} = ${hunsA1 > 0 ? hunsA1 + '␣+␣' : ''}${tensA1} + ${onesA1}<br
                    >`;
376             steps += `${a2} = ${hunsA2 > 0 ? hunsA2 + '␣+␣' : ''}${tensA2} + ${onesA2}<br
                    >`;
377
378             // Add like units
379             const initialHunsSum = hunsA1 + hunsA2;
380             const initialTensSum = tensA1 + tensA2;
381             const initialOnesSum = onesA1 + onesA2;
382             steps += '<br><strong>Combine␣Like␣Units:</strong><br>';
383             if(initialHunsSum > 0) steps += `Hundreds: ${hunsA1} + ${hunsA2} = ${
                    initialHunsSum}<br>`;
384             steps += `Tens: ${tensA1} + ${tensA2} = ${initialTensSum}<br>`;
385             steps += `Ones: ${onesA1} + ${onesA2} = ${initialOnesSum}<br>`;
386
387             // Handle Compositions
388             steps += '<br><strong>Composition:</strong><br>';
389             let onesCarry = Math.floor(initialOnesSum / 10) * 10;
390             let finalOnesSum = initialOnesSum % 10;
391             if (onesCarry > 0) {
392                 steps += `- Compose ${onesCarry} from ones into ${onesCarry/10} ten(s).
                        Remaining ones: ${finalOnesSum}<br>`;
393             } else {
394                  steps += `- No composition needed for ones.<br>`;
395             }
396
397             let tensSumAfterOnesCarry = initialTensSum + onesCarry;
398             let tensCarry = Math.floor(tensSumAfterOnesCarry / 100) * 100;
399             let finalTensSum = tensSumAfterOnesCarry % 100;
400              if (tensCarry > 0) {
401                  steps += `- Compose ${tensCarry} from tens into ${tensCarry/100} hundred(s
                         ). Remaining tens: ${finalTensSum}<br>`;
402              } else {
403                   steps += `- No composition needed for tens.<br>`;
404              }
405
406              let finalHunsSum = initialHunsSum + tensCarry;
407
408             // Combine for final result
409             const finalSum = finalHunsSum + finalTensSum + finalOnesSum;
410             steps += '<br><strong>Final␣Result:</strong><br>';
411             steps += `${finalHunsSum > 0 ? finalHunsSum + '␣+␣': ''}${finalTensSum} + ${
                    finalOnesSum} = ${finalSum}`; // Hide 0 hundreds in final sum text
412
413
414             outputDiv.innerHTML = steps;
415             typesetMath();
416
417             // Draw Diagram
418              drawABAODiagram('abaoDiagram', a1, a2, hunsA1, tensA1, onesA1, hunsA2, tensA2
                     , onesA2,
419                              initialHunsSum, initialTensSum, initialOnesSum,
```

```
                            onesCarry, tensCarry,
                            finalHunsSum, finalTensSum, finalOnesSum, finalSum);
        };

        function typesetMath() { /* Placeholder */ }

        // Initialize on load
         const initialOutputDiv = document.getElementById('abaoOutput');
        if (initialOutputDiv) {
            // Run with default values on load
            runABAOAutomaton();
        }

    })(); // End of IIFE

</script>

</body>
</html>
```

References

Hackenberg, A. (2025). *Course notes* [Unpublished course notes].