Addition Strategies: Adding Bases and Adding Ones

Compiled by: Theodore M. Savich

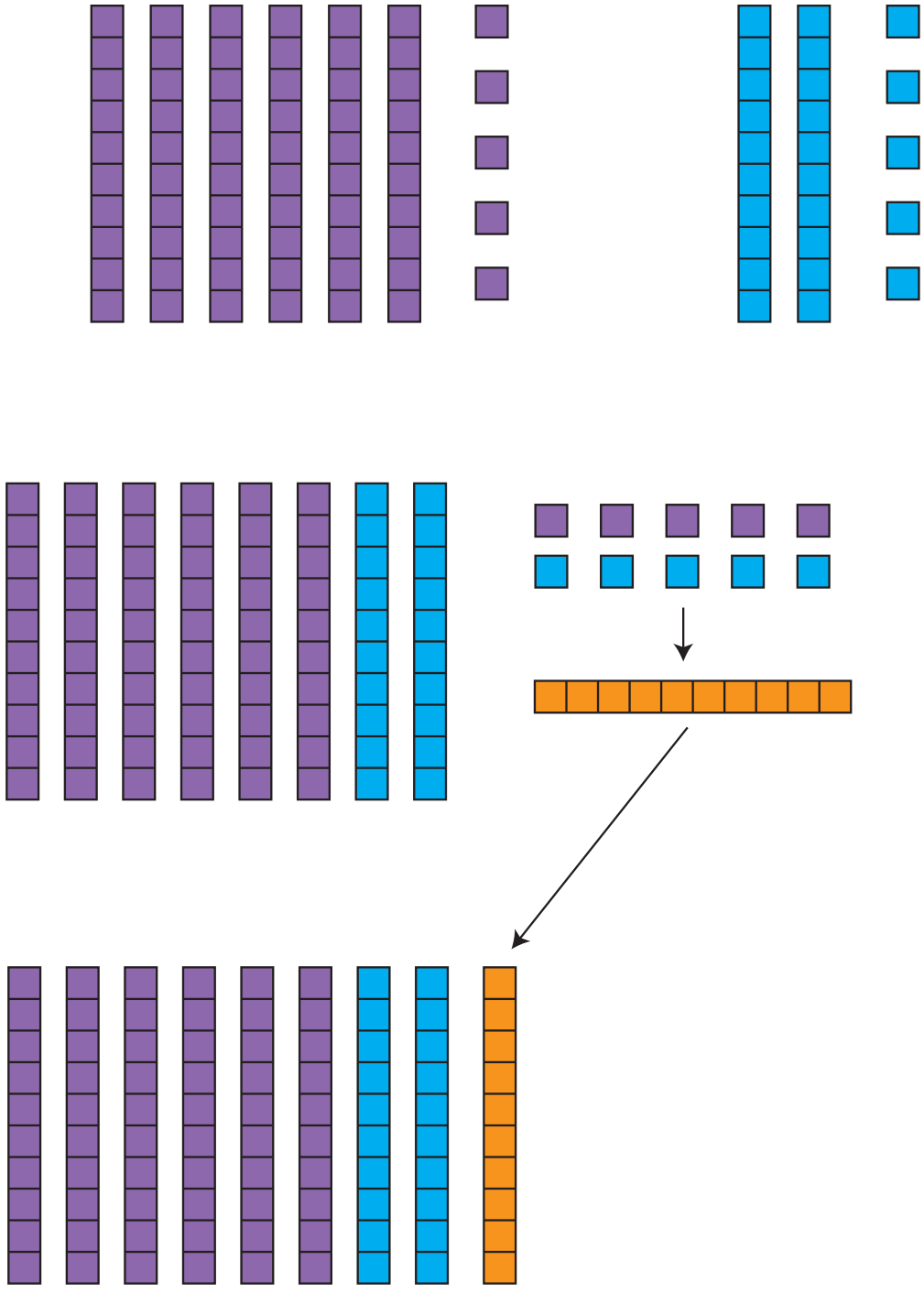
April 3, 2025

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

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

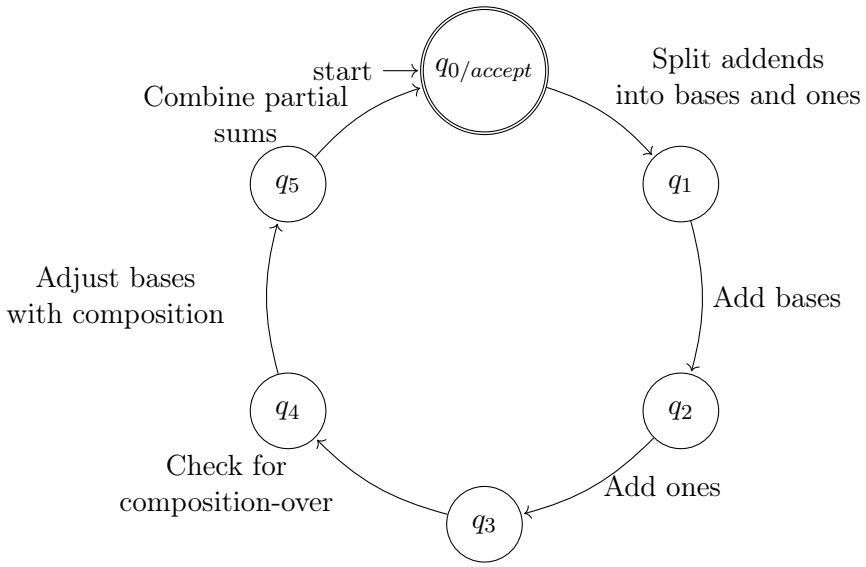
where:

- $Q = \{q_{0/accept}, q_1, q_2, q_3, q_4, q_5\}$ 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; }
        .stopping-point { fill: red; }
        .number-line-label { font-size: 12px; }
    </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);
    }
```

```
    function drawStoppingPoint(svg, x, y, labelText, labelOffsetBase = 20, index = 0) {
        const circle = document.createElementNS('http://www.w3.org/2000/svg', 'circle');
        circle.setAttribute('cx', x);
        circle.setAttribute('cy', y);
        circle.setAttribute('r', 4);
        circle.setAttribute('class', 'stopping-point');
        svg.appendChild(circle);

        // Use the provided y parameter instead of numberLineY
        if (labelText) {
            // Add staggering based on index to prevent overlap with large values
            const labelOffset = labelOffsetBase * (index % 2 === 0 ? 1.5 : -1.8);
            createText(svg, x, y + labelOffset, labelText, 'number-line-label');
        }
    }
// --- End Helper Functions ---


function drawABAODiagram(svgId, a1, a2, hunsA1, tensA1, onesA1, hunsA2, tensA2, onesA2,
                         initialHunsSum, initialTensSum, initialOnesSum,
                         onesCarry, tensCarry,
                         finalHunsSum, finalTensSum, finalOnesSum, finalSum)
{
    const svg = document.getElementById(svgId);
    if (!svg) return;
    svg.innerHTML = ''; // Clear SVG

    const svgWidth = parseFloat(svg.getAttribute('width'));
    const svgHeight = parseFloat(svg.getAttribute('height'));
    const blockUnitSize = 10;
    const tenBlockWidth = blockUnitSize;
    const tenBlockHeight = blockUnitSize * 10;
    const hundredBlockSize = blockUnitSize * 10;
    const blockSpacing = 4;
    const groupSpacingX = 30;
    const sectionSpacingY = 140; // Increased spacing slightly
    const startX = 30;
    let currentY = 40;
    const colorA1 = 'purple';
    const colorA2 = 'cyan';
    const colorOnesCarry = 'orange';
    const colorTensCarry = 'lightgreen';
    const maxBlockHeight = Math.max(tenBlockHeight, hundredBlockSize, blockUnitSize);
    const calcLabelYOffset = 20; // Offset below blocks for calc labels
    const textHeightApproximation = 10; // Approximate height of text for arrow start Y

    // --- 1. Initial Split Visualization ---
    createText(svg, startX, currentY, `Initial Split: ${a1} = ${hunsA1 > 0 ? hunsA1 + '+': ''}${tensA1}+${onesA1}, ${a2} = ${hunsA2 > 0 ? hunsA2 + '+' : ''}${tensA2}+${onesA2}`);
    currentY += 30;
```

```
201
202         let currentX = startX;
203         let section1MaxY = currentY;
204
205         // A1 Blocks
206         for (let i = 0; i < hunsA1 / 100; i++) { drawHundredBlock(svg, currentX, currentY,
                 hundredBlockSize, colorA1, blockUnitSize); currentX += hundredBlockSize +
                groupSpacingX; section1MaxY = Math.max(section1MaxY, currentY +
                hundredBlockSize); }
207         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); }
208         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); }
209         const a1EndX = currentX;
210
211         // A2 Blocks
212         currentX = a1EndX + groupSpacingX * 2;
213         const a2StartX = currentX;
214         for (let i = 0; i < hunsA2 / 100; i++) { drawHundredBlock(svg, currentX, currentY,
                 hundredBlockSize, colorA2, blockUnitSize); currentX += hundredBlockSize +
                groupSpacingX; section1MaxY = Math.max(section1MaxY, currentY +
                hundredBlockSize); }
215         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); }
216         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); }
217         currentY = section1MaxY + sectionSpacingY;
218
219
220         // --- 2. Combine Like Units (Before Composition) ---
221         createText(svg, startX, currentY, `Combine Like Units`);
222         currentY += 30;
223
224         let section2MaxY = currentY;
225         let combinedHunsX = startX;
226         let combinedTensX = 0;
227         let combinedOnesX = 0;
228         let hunsEndX = startX;
229         let tensEndX = 0;
230         let onesEndX = 0;
231         let onesGroupEndX = 0; // For composition grouping rect later
232         let tensGroupStartX = 0; // For composition grouping rect later
233         let tensGroupEndX = 0; // For composition grouping rect later
234
235
236         // Draw Combined Hundreds
```

```
237         if(initialHunsSum > 0) {
238             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; }
239             hunsEndX = combinedHunsX;
240             createText(svg, startX + (hunsEndX - startX - blockSpacing) / 2, currentY +
                    hundredBlockSize + calcLabelYOffset, `${hunsA1}+${hunsA2}=${initialHunsSum
                    }`, 'calc-label', 'middle');
241             section2MaxY = Math.max(section2MaxY, currentY + hundredBlockSize);
242             combinedTensX = hunsEndX + groupSpacingX;
243         } else {
244             combinedTensX = startX;
245         }
246
247         // Draw Combined Tens
248         tensGroupStartX = combinedTensX; // Mark start for potential grouping
249         currentX = combinedTensX;
250         for (let i = 0; i < initialTensSum / 10; i++) {
251             let color = (i < tensA1 / 10) ? colorA1 : colorA2;
252             drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, color,
                    blockUnitSize);
253              if (i < 10) tensGroupEndX = currentX + tenBlockWidth; // Track end of first
                     10 tens
254             currentX += tenBlockWidth + blockSpacing;
255         }
256         tensEndX = currentX;
257         const tensLabelX = combinedTensX + (tensEndX - combinedTensX - blockSpacing) / 2;
258         const tensLabelY = currentY + tenBlockHeight + calcLabelYOffset;
259         createText(svg, tensLabelX, tensLabelY, `${tensA1}+${tensA2}=${initialTensSum}`, '
                calc-label', 'middle');
260         section2MaxY = Math.max(section2MaxY, currentY + tenBlockHeight);
261
262
263         // Draw Combined Ones
264         combinedOnesX = tensEndX + groupSpacingX;
265         currentX = combinedOnesX;
266         for (let i = 0; i < initialOnesSum; i++) {
267             let color = (i < onesA1) ? colorA1 : colorA2;
268             drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize,
                    blockUnitSize, blockUnitSize, color);
269             if (i < 10) onesGroupEndX = currentX + blockUnitSize; // Track end of first 10
                     ones
270             currentX += blockUnitSize + blockSpacing;
271         }
272         onesEndX = currentX;
273         const onesLabelX = combinedOnesX + (onesEndX - combinedOnesX - blockSpacing) / 2;
274         const onesLabelY = currentY + maxBlockHeight + calcLabelYOffset;
275         createText(svg, onesLabelX, onesLabelY, `${onesA1}+${onesA2}=${initialOnesSum}`, '
                calc-label', 'middle');
276         section2MaxY = Math.max(section2MaxY, currentY + maxBlockHeight);
277
278
279         // --- 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;
        let finalOnesStartX = 0;


        // Final Hundreds
        let composedHundredX = 0, composedHundredY = 0;
        for (let i = 0; i < finalHunsSum / 100; i++) {
            let color;
            if (i < hunsA1 / 100) color = colorA1;
            else if (i < initialHunsSum / 100) color = colorA2;
            else {
                color = colorTensCarry; // Color for hundred composed from tens
                composedHundredX = currentX + hundredBlockSize / 2; // Store center of composed hundred
                composedHundredY = currentY + hundredBlockSize / 2;
            }
            drawHundredBlock(svg, currentX, currentY, hundredBlockSize, color, blockUnitSize);
            currentX += hundredBlockSize + blockSpacing;
        }
        let finalHunsEndX = currentX > startX ? currentX - blockSpacing : startX;

        // Final Tens
        currentX = finalHunsEndX + (finalHunsEndX > startX ? groupSpacingX : 0);
        finalTensStartX = currentX; // Store start X for final tens
        let composedTenX = 0, composedTenY = 0;
        for (let i = 0; i < finalTensSum / 10; i++) {
            let color = colorA1; // Default/placeholder color
             // More precise coloring: Check if this ten block is the one created by onesCarry
            if (onesCarry > 0 && i === initialTensSum / 10) { // If it's the position right after initial tens
                color = colorOnesCarry;
                composedTenX = currentX + tenBlockWidth / 2; // Store center of composed ten
                composedTenY = currentY + tenBlockHeight / 2;
```

```
            } else if (i < tensA1 / 10 && tensCarry == 0) { // Original A1 if no tens->hundred carry
                color = colorA1;
            } else if (i < initialTensSum / 10 && tensCarry == 0) { // Original A2 if no tens->hundred carry
                color = colorA2;
            }
            // If tensCarry happened, coloring remaining tens accurately is complex, using carry color as fallback
            else if (tensCarry > 0) {
                color = colorOnesCarry; // Might be remaining original or from ones carry
            }

            drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, color, blockUnitSize);
            currentX += tenBlockWidth + blockSpacing;
        }
        let finalTensEndX = currentX > finalTensStartX ? currentX - blockSpacing : finalTensStartX;

        // Final Ones Blocks
        currentX = finalTensEndX + (finalTensEndX > finalTensStartX ? groupSpacingX : 0);
        finalOnesStartX = currentX;
        for (let i = 0; i < finalOnesSum; i++) {
            let color = (i < onesA1 && onesCarry == 0) ? colorA1 : colorA2;
            drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, color);
            currentX += blockUnitSize + blockSpacing;
        }
        finalMaxY = Math.max(currentY + maxBlockHeight, currentY + hundredBlockSize);

        // --- Draw Composition Arrows ---
        // Arrow from ones sum text to composed ten block
        if (onesCarry > 0 && composedTenX > 0) {
            createCurvedArrow(svg,
                onesLabelX, onesArrowStartY, // Start below ones calculation text
                composedTenX, composedTenY - tenBlockHeight/2, // End at top-center of composed ten block
                onesLabelX + 30, onesArrowStartY + sectionSpacingY / 2 // Control point
            );
        }
        // Arrow from tens sum text to composed hundred block
         if (tensCarry > 0 && composedHundredX > 0) {
             createCurvedArrow(svg,
                tensLabelX, tensArrowStartY, // Start below tens calculation text
                composedHundredX, composedHundredY - hundredBlockSize/2, // End at top-center of composed hundred block
                tensLabelX + 50, tensArrowStartY + sectionSpacingY / 2 // Control point
            );
         }
    }


    (function() { // IIFE
```

```
372         window.runABAOAutomaton = function() {
373             const outputDiv = document.getElementById('abaoOutput');
374             const a1 = parseInt(document.getElementById('abaoAddend1').value);
375             const a2 = parseInt(document.getElementById('abaoAddend2').value);
376 
377             if (isNaN(a1) || isNaN(a2)) {
378                 outputDiv.textContent = "Please␣enter␣valid␣numbers␣for␣both␣addends";
379                 diagramABAOSVG.innerHTML = ''; // Clear diagram on error
380                 return;
381             }
382 
383             let steps = '';
384 
385             // Split both addends
386             const hunsA1 = Math.floor(a1 / 100) * 100;
387             const tensA1 = Math.floor((a1 % 100) / 10) * 10;
388             const onesA1 = a1 % 10;
389             const hunsA2 = Math.floor(a2 / 100) * 100;
390             const tensA2 = Math.floor((a2 % 100) / 10) * 10;
391             const onesA2 = a2 % 10;
392             steps += '<strong>Splitting␣Addends:</strong><br>';
393             steps += `${a1} = ${hunsA1 > 0 ? hunsA1 + '␣+␣' : ''}${tensA1} + ${onesA1}<br
                >`;
394             steps += `${a2} = ${hunsA2 > 0 ? hunsA2 + '␣+␣' : ''}${tensA2} + ${onesA2}<br
                >`;
395 
396             // Add like units
397             const initialHunsSum = hunsA1 + hunsA2;
398             const initialTensSum = tensA1 + tensA2;
399             const initialOnesSum = onesA1 + onesA2;
400             steps += '<br><strong>Combine␣Like␣Units:</strong><br>';
401             if(initialHunsSum > 0) steps += `Hundreds: ${hunsA1} + ${hunsA2} = ${
                    initialHunsSum}<br>`;
402             steps += `Tens: ${tensA1} + ${tensA2} = ${initialTensSum}<br>`;
403             steps += `Ones: ${onesA1} + ${onesA2} = ${initialOnesSum}<br>`;
404 
405             // Handle Compositions
406             steps += '<br><strong>Composition:</strong><br>';
407             let onesCarry = Math.floor(initialOnesSum / 10) * 10;
408             let finalOnesSum = initialOnesSum % 10;
409             if (onesCarry > 0) {
410                 steps += `- Compose ${onesCarry} from ones into ${onesCarry/10} ten(s).
                        Remaining ones: ${finalOnesSum}<br>`;
411             } else {
412                  steps += `- No composition needed for ones.<br>`;
413             }
414 
415             let tensSumAfterOnesCarry = initialTensSum + onesCarry;
416             let tensCarry = Math.floor(tensSumAfterOnesCarry / 100) * 100;
417             let finalTensSum = tensSumAfterOnesCarry % 100;
418              if (tensCarry > 0) {
419                  steps += `- Compose ${tensCarry} from tens into ${tensCarry/100} hundred(s
                        ). Remaining tens: ${finalTensSum}<br>`;
420              } else {
```

```
                steps += `- No composition needed for tens.<br>`;
            }

            let finalHunsSum = initialHunsSum + tensCarry;

            // Combine for final result
            const finalSum = finalHunsSum + finalTensSum + finalOnesSum;
            steps += '<br><strong>Final Result:</strong><br>';
            steps += `${finalHunsSum > 0 ? finalHunsSum + ' + ': ''}${finalTensSum} + ${finalOnesSum} = ${finalSum}`; // Hide 0 hundreds in final sum text


            outputDiv.innerHTML = steps;
            typesetMath();

            // Draw Diagram
            drawABAODiagram('abaoDiagram', a1, a2, hunsA1, tensA1, onesA1, hunsA2, tensA2, onesA2,
                            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>

<!-- New button for viewing PDF documentation -->
<button onclick="openPdfViewer()">Want to learn more about this strategy? Click here.</button>

<script>
    function openPdfViewer() {
        // Opens the PDF documentation for the strategy.
        window.open('../SAR_ADD_ABAO.pdf', '_blank');
    }
</script>

</body>
</html>
```

References

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