Subtraction Strategies: Decomposition

Compiled by: Theodore M. Savich

April 1, 2025

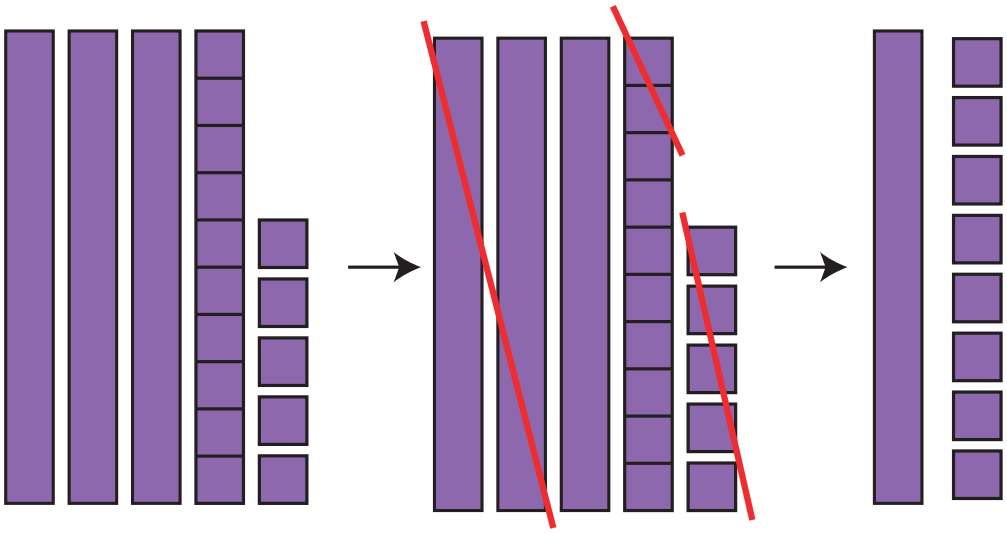
Transcript

Video from **Carpenter1999<empty citation>**. Strategy descriptions and examples adapted from **HackenbergCourseNotes<empty citation>**

- **Teacher:** Lucy ordered 45 cupcakes for her birthday. At the party, her guests ate 27 cupcakes, how many cupcakes did she have left? [BACKGROUND]
- **Joel:** This is 10, this is 10, this is 10, this is 10 and this is five.18.
- **Teacher:** Explain to us what you did there.
- **Joel:** I have, this is 10, this is 10, this is 10, and this is five. So I take away 20 and I take away five. I take away two more. So they enter and then I counted these and those, and so the answer was 18.
- **Teacher:** Nice work

Notation Representing Joel's Solution:

$$
\begin{aligned}
&47 - 27 \\
&45 - 20 = 25 \\
&\ 25 - 7 = ? \\
&\qquad 2 \text{ tens } + 5 \text{ ones } - 7 \text{ ones} \\
&1 \text{ ten } + 1 \text{ ten } + 5 \text{ ones } - 7 \text{ ones} \\
&\qquad \downarrow \text{DECOMPOSE} \\
&1 \text{ ten } + 10 \text{ ones } + 5 \text{ ones } - 7 \text{ ones} \\
&\qquad 1 \text{ ten } + 8 \text{ ones } + \underbrace{7 \text{ ones } - 7 \text{ ones}}_{=0} \\
&\qquad 1 \text{ ten } + 8 \text{ ones}
\end{aligned}
$$



Notation Representing Joel's Solution: Imagine representing both numbers by their base units and ones. Begin by subtracting the base components, then subtract the ones. If there aren't enough ones available in the larger number to subtract the ones from the smaller number (while keeping the result positive), break one base unit into its individual ones. Finally, remove only the exact number of ones required to complete the subtraction.

Decomposition

Description of Strategy

- **Objective:** Decompose a base unit from the minuend into ones to have enough ones to subtract the ones in the subtrahend.

Automaton Type

Pushdown Automaton (PDA): Needed to handle the decomposition process and keep track of base units.

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.
- $\Gamma = \{Z_0\} \cup \{b \mid b \in \mathbb{N}\}$ is the stack alphabet, where Z_0 is the initial stack symbol and b represents a base unit (e.g., 10 in base-ten).
- $q_{0/accept}$ is the start state, which is also the accept state.
- $F = \{q_{0/accept}\}$ is the set of accepting states.

The transition function δ is defined as:

1. $\delta(q_{0/accept}, \text{“}M, S\text{”}, Z_0) = \{(q_1, Z_0)\}$
   (Split the minuend M and subtrahend S into their base and ones components.)

2. $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$
   (Compare the ones in M and S.)

3. $\delta(q_2, \varepsilon, Z_0) = \{(q_3, b\,Z_0)\}$
   (If the ones in M are insufficient, decompose a base unit b into ones.)

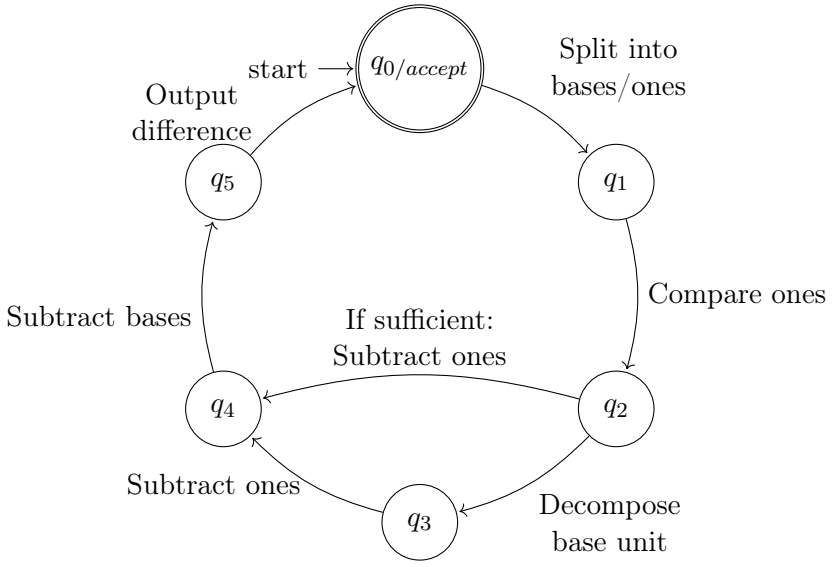
4. $\delta(q_2, \varepsilon, Z_0) = \{(q_4, Z_0)\}$
   (If the ones in M are sufficient, proceed to subtract ones.)

5. $\delta(q_3, \varepsilon, b) = \{(q_4, b)\}$
   (After decomposition, subtract the ones.)

6. $\delta(q_4, \varepsilon, Z_0) = \{(q_5, Z_0)\}$
   (Subtract the bases.)

7. $\delta(q_5, \varepsilon, Z_0) = \{(q_{0/accept}, Z_0)\}$
   (Output the final difference.)

Automaton Diagram for Decomposition



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Subtraction Strategies: Decomposition</title>
    <style>
        body { font-family: sans-serif; }
        #diagramDecompositionSVG { border: 1px solid #d3d3d3; min-height: 750px; /* Ensure ample space */ }
        #outputContainer { margin-top: 20px; }
        /* Notation Styles */
        .notation-line { padding-left: 1em; margin: 0.1em 0; font-family: monospace; /* Monospace for alignment */ }
        .notation-line.problem { font-weight: bold; padding-left: 0; margin-bottom: 0.5em;}
        .notation-line.indent-1 { padding-left: 1em; }
        .notation-line.indent-2 { padding-left: 2em; }
        .notation-line.indent-3 { padding-left: 3em; }
        .notation-line.decompose-arrow { padding-left: 3em; font-size: 1.2em; margin: 0.2em 0; }
        .notation-line.final-step { margin-top: 0.5em; }
        .cancel-group { text-decoration: line-through; color: #888; }

        /* Diagram Styles */
        .diagram-label { font-size: 14px; display: block; margin-bottom: 10px; font-weight: bold;}
        .calc-label { font-size: 12px; text-anchor: middle; }
        .block { stroke: black; stroke-width: 0.5; }
        .ten-block-bg { stroke: black; stroke-width: 1; }
        .hundred-block-bg { stroke: black; stroke-width: 1; }
        .unit-block-inner { stroke: lightgrey; stroke-width: 0.5; }
        .decomposed-block-visual { /* Style for the visually decomposed TEN block */
            fill: none; /* Make transparent */
            stroke: black;
            stroke-width: 1.5;
            stroke-dasharray: 4 4; /* Dashed line */
        }
        /* Removed decomposed-block-overlay */
        .cross-out { stroke: red; stroke-width: 2.5; stroke-opacity: 0.8; }
        .number-line-arrow { fill: black; stroke: black;} /* Reuse for general arrows if needed */

    </style>
</head>
<body>

<h1>Subtraction Strategies: Decomposition</h1>

<div>
    <label for="decompMinuend">Minuend:</label>
    <input type="number" id="decompMinuend" value="45"> <!-- Default to Joe's␣example␣-->
</div>
<div>
```

```
␣␣␣␣<label␣for="decompSubtrahend">Subtrahend:</label>
␣␣␣␣<input␣type="number"␣id="decompSubtrahend"␣value="27">␣<!--␣Default␣to␣Joe's example -->
</div>

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

<div id="outputContainer">
    <h2>Explanation (Notation):</h2>
    <div id="decompositionOutput">
        <!-- Text notation will be displayed here -->
    </div>
</div>

<h2>Diagram:</h2>
<svg id="diagramDecompositionSVG" width="700" height="800"></svg> <!-- Adjusted height -->

<!-- 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_SUB_Decomposition.pdf', '_blank');
    }
</script>

<script>
    // --- Helper SVG Functions --- (Keep functions from previous version: drawBlock, drawTenBlock, drawHundredBlock, createText, drawCrossOut) ---
     function drawBlock(svg, x, y, width, height, fill, className = 'block') {
        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('class', className);
        svg.appendChild(rect);
        return { x, y, width, height, type: 'o' }; // Return info including type
    }

    function drawTenBlock(svg, x, y, width, height, fill, unitBlockSize, isDecomposed = false) {
        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', isDecomposed ? 'none' : fill); // Transparent if visually decomposed
        backgroundRect.setAttribute('class', isDecomposed ? 'decomposed-block-visual' : 'ten-block-bg␣block');
```

```
92          group.appendChild(backgroundRect);
93
94          // Draw inner units only if NOT visually decomposed
95          if (!isDecomposed) {
96              for (let i = 0; i < 10; i++) {
97                  const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
                        rect');
98                  unitBlock.setAttribute('x', x); unitBlock.setAttribute('y', y + i *
                        unitBlockSize);
99                  unitBlock.setAttribute('width', unitBlockSize); unitBlock.setAttribute('
                        height', unitBlockSize);
100                 unitBlock.setAttribute('fill', fill);
101                 unitBlock.setAttribute('class', 'unit-block-inner');
102                 group.appendChild(unitBlock);
103             }
104         }
105         svg.appendChild(group);
106         return { x, y, width, height, type: 't', decomposed: isDecomposed }; // Return
                info
107     }
108
109      function drawHundredBlock(svg, x, y, size, fill, unitBlockSize, isDecomposed = false
            ) {
110         const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
111         const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", '
                rect');
112         backgroundRect.setAttribute('x', x); backgroundRect.setAttribute('y', y);
113         backgroundRect.setAttribute('width', size); backgroundRect.setAttribute('height',
                size);
114         backgroundRect.setAttribute('fill', isDecomposed ? 'none' : fill); // Transparent
                if visually decomposed
115          backgroundRect.setAttribute('class', isDecomposed ? 'decomposed-block-visual' : '
                 hundred-block-bg␣block');
116         group.appendChild(backgroundRect);
117
118         if (!isDecomposed) {
119             for (let row = 0; row < 10; row++) {
120                 for (let col = 0; col < 10; col++) {
121                     const unitBlock = document.createElementNS("http://www.w3.org/2000/svg"
                            , 'rect');
122                     unitBlock.setAttribute('x', x + col * unitBlockSize);
123                     unitBlock.setAttribute('y', y + row * unitBlockSize);
124                     unitBlock.setAttribute('width', unitBlockSize);
125                     unitBlock.setAttribute('height', unitBlockSize);
126                     unitBlock.setAttribute('fill', fill);
127                     unitBlock.setAttribute('class', 'unit-block-inner');
128                     group.appendChild(unitBlock);
129                 }
130             }
131         }
132         svg.appendChild(group);
133         return { x, y, width: size, height: size, type: 'h', decomposed: isDecomposed };
                // Return info
134     }
```

```
135
136     function createText(svg, x, y, textContent, className = 'diagram-label', anchor = '
            start') {
137         const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
138         text.setAttribute('x', x); text.setAttribute('y', y);
139         text.setAttribute('class', className);
140         text.setAttribute('text-anchor', anchor);
141         text.textContent = textContent;
142         svg.appendChild(text);
143     }
144
145     function drawCrossOut(svg, x, y, width, height) {
146          const line1 = document.createElementNS("http://www.w3.org/2000/svg", 'line');
147          line1.setAttribute('x1', x); line1.setAttribute('y1', y);
148          line1.setAttribute('x2', x + width); line1.setAttribute('y2', y + height);
149          line1.setAttribute('class', 'cross-out');
150          svg.appendChild(line1);
151           const line2 = document.createElementNS("http://www.w3.org/2000/svg", 'line');
152          line2.setAttribute('x1', x + width); line2.setAttribute('y1', y);
153          line2.setAttribute('x2', x); line2.setAttribute('y2', y + height);
154          line2.setAttribute('class', 'cross-out');
155          svg.appendChild(line2);
156     }
157     // --- End Helper Functions ---
158
159     // --- Refactored Diagram Function for Notation Alignment ---
160     function drawDecompositionDiagram(svgId, m, s,
161                                       mHunsOrig, mTensOrig, mOnesOrig,
162                                       sHuns, sTens, sOnes,
163                                       diffAfterTens, // Value after tens subtraction (for
                                              stage 2 start)
164                                       didDecomposeTen, // Flag
165                                       finalResult)
166     {
167         const svg = document.getElementById(svgId);
168         if (!svg) return;
169         svg.innerHTML = '';
170
171         const svgWidth = parseFloat(svg.getAttribute('width'));
172         const svgHeight = parseFloat(svg.getAttribute('height'));
173         const blockUnitSize = 10;
174         const tenBlockWidth = blockUnitSize;
175         const tenBlockHeight = blockUnitSize * 10;
176         const hundredBlockSize = blockUnitSize * 10;
177         const blockSpacing = 4;
178         const groupSpacingX = 20;
179         const sectionSpacingY = 150; // Increased vertical spacing
180         const startX = 30;
181         let currentY = 40;
182         const colorM = 'lightblue';
183         const colorResult = 'gold';
184         const maxBlockHeight = Math.max(tenBlockHeight, hundredBlockSize, blockUnitSize);
185         let blockDataStage2 = []; // Store blocks drawn in stage 2
186
```

```
        // --- 1. Initial Minuend Visualization ---
        createText(svg, startX, currentY, `Initial Minuend: ${m}`);
        currentY += 30;
        let currentX = startX;
        let section1MaxY = currentY;

        for (let i = 0; i < mHunsOrig / 100; i++) { drawHundredBlock(svg, currentX, currentY, hundredBlockSize, colorM, blockUnitSize); currentX += hundredBlockSize + groupSpacingX; section1MaxY = Math.max(section1MaxY, currentY + hundredBlockSize); }
        for (let i = 0; i < mTensOrig / 10; i++) { drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, colorM, blockUnitSize); currentX += tenBlockWidth + blockSpacing; section1MaxY = Math.max(section1MaxY, currentY + tenBlockHeight); }
        for (let i = 0; i < mOnesOrig; i++) { drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, colorM); currentX += blockUnitSize + blockSpacing; section1MaxY = Math.max(section1MaxY, currentY + maxBlockHeight); }

        currentY = section1MaxY + sectionSpacingY;


        // --- 2. Subtract Tens & Decompose/Subtract Ones ---
        createText(svg, startX, currentY, `Subtracting ${s} (${sTens} tens, ${sOnes} ones) ${didDecomposeTen ? ' - Decomposing 1 Ten' : ''}`);
        currentY += 30;
        currentX = startX;
        let section2MaxY = currentY;
        blockDataStage2 = []; // Reset for this stage

        // Draw the state *after* TENS subtraction, marking decomposition visually
        let hunsAfterTens = Math.floor(diffAfterTens / 100) * 100;
        let tensAfterTens = Math.floor((diffAfterTens % 100) / 10) * 10;
        let onesAfterTens = diffAfterTens % 10;

        // Draw hundreds remaining
        for (let i = 0; i < hunsAfterTens / 100; i++) {
            let info = drawHundredBlock(svg, currentX, currentY, hundredBlockSize, colorM, blockUnitSize);
            blockDataStage2.push({ status: 'keep', ...info });
            currentX += hundredBlockSize + groupSpacingX;
            section2MaxY = Math.max(section2MaxY, currentY + hundredBlockSize);
        }
        // Draw tens remaining, mark the one to be decomposed
        let decomposedTenIndex = -1; // Index relative to *drawn* tens in this stage
        for (let i = 0; i < tensAfterTens / 10; i++) {
            let isDecomposed = didDecomposeTen && i === (tensAfterTens / 10) - 1;
            let info = drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, colorM, blockUnitSize, isDecomposed);
            if (isDecomposed) {
                decomposedTenIndex = blockDataStage2.length; // Store index if decomposed
                blockDataStage2.push({ status: 'decomposed', ...info });
            } else {
                blockDataStage2.push({ status: 'keep', ...info });
```

```
            }
            currentX += tenBlockWidth + blockSpacing;
            section2MaxY = Math.max(section2MaxY, currentY + tenBlockHeight);
        }
         // Draw ones remaining
         let onesStartX = currentX + groupSpacingX;
         currentX = onesStartX;
        for (let i = 0; i < onesAfterTens; i++) {
            let info = drawBlock(svg, currentX, currentY + maxBlockHeight - blockUnitSize,
                 blockUnitSize, blockUnitSize, colorM);
            blockDataStage2.push({ status: 'keep', ...info }); // Mark as 'keep' initially
            currentX += blockUnitSize + blockSpacing;
            section2MaxY = Math.max(section2MaxY, currentY + maxBlockHeight);
        }

        // Perform Cross Out for sOnes
        let onesToCrossOut = sOnes;
        // Cross out original ones first
        blockDataStage2.filter(b => b.type === 'o' && b.status === 'keep').forEach(block
            => {
            if (onesToCrossOut > 0) {
                drawCrossOut(svg, block.x, block.y, block.width, block.height);
                block.status = 'crossed'; // Mark as crossed
                onesToCrossOut--;
            }
        });

        // If still need to cross out more, visualize the decomposed tens as individual
            units
        if (onesToCrossOut > 0 && decomposedTenIndex !== -1) {
            let decompBlock = blockDataStage2[decomposedTenIndex];

            // First, draw the 10 individual unit blocks from the decomposed ten
            let decompUnitBlocks = [];
            for(let i = 0; i < 10; i++) {
                // Calculate position of unit block inside the ten block
                let unitY = decompBlock.y + i * blockUnitSize;
                let info = drawBlock(svg, decompBlock.x, unitY, blockUnitSize,
                    blockUnitSize, colorM, 'unit-from-decomposed');
                decompUnitBlocks.push(info);
            }

            // Now cross out the required number of ones from the decomposed ten
            for(let i = 0; i < onesToCrossOut; i++) {
                if (i < decompUnitBlocks.length) {
                    let unitBlock = decompUnitBlocks[i];
                    drawCrossOut(svg, unitBlock.x, unitBlock.y, unitBlock.width, unitBlock.
                        height);
                }
            }

            // Add an annotation to explain the decomposition
            createText(svg, decompBlock.x + tenBlockWidth + 10, decompBlock.y +
                tenBlockHeight/2,
```

```
                    "1 ten decomposed into 10 ones", "calc-label", "start");
        }


        currentY = section2MaxY + sectionSpacingY;


        // --- 3. Final Result Visualization ---
        createText(svg, startX, currentY, `Final Result: ${finalResult}`);
        currentY += 30;
        currentX = startX;
        let section3MaxY = currentY;

        // Draw Result Blocks based on finalResult calculation
        let finalHuns = Math.floor(finalResult / 100) * 100;
        let finalTens = Math.floor((finalResult % 100) / 10) * 10;
        let finalOnes = finalResult % 10;

        for (let i = 0; i < finalHuns / 100; i++) { drawHundredBlock(svg, currentX,
            currentY, hundredBlockSize, colorResult, blockUnitSize); currentX +=
            hundredBlockSize + groupSpacingX; section3MaxY = Math.max(section3MaxY,
            currentY + hundredBlockSize); }
        for (let i = 0; i < finalTens / 10; i++) { drawTenBlock(svg, currentX, currentY,
            tenBlockWidth, tenBlockHeight, colorResult, blockUnitSize); currentX +=
            tenBlockWidth + blockSpacing; section3MaxY = Math.max(section3MaxY, currentY +
             tenBlockHeight); }
        for (let i = 0; i < finalOnes; i++) { drawBlock(svg, currentX, currentY +
            maxBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, colorResult);
            currentX += blockUnitSize + blockSpacing; section3MaxY = Math.max(section3MaxY
            , currentY + maxBlockHeight); }

    } // End drawDecompositionDiagram


    document.addEventListener('DOMContentLoaded', function() {
        const outputDiv = document.getElementById('decompositionOutput');
        const mInput = document.getElementById('decompMinuend');
        const sInput = document.getElementById('decompSubtrahend');
        const diagramSVG = document.getElementById('diagramDecompositionSVG');

        if (!outputDiv || !mInput || !sInput || !diagramSVG) {
            console.error("Required HTML elements not found!");
            return;
        }

        window.runDecompositionAutomaton = function() {
            try {
                const m = parseInt(mInput.value);
                const s = parseInt(sInput.value);

                if (isNaN(m) || isNaN(s)) {
                    outputDiv.innerHTML = "<p>Please enter valid numbers</p>";
                    diagramSVG.innerHTML = ''; return;
                }
```

```
322             if (s > m) {
323                 outputDiv.innerHTML = "<p>Subtrahend cannot be greater than Minuend.</p>";
324                 diagramSVG.innerHTML = ''; return;
325             }
326
327             let steps = '';
328
329             // --- Generate Notation Step-by-Step ---
330             steps += `<p class="notation-line problem">${m} - ${s} = ?</p>`;
331             const sTens = Math.floor(s / 10) * 10;
332             const sOnes = s % 10;
333             const diffAfterTens = m - sTens;
334             steps += `<p class="notation-line">${m} - ${sTens} = ${diffAfterTens}</p>`;
335             steps += `<p class="notation-line">${diffAfterTens} - ${sOnes} = ?</p>`;
336             const diffTensVal = Math.floor((diffAfterTens % 100) / 10);
337             const diffOnesVal = diffAfterTens % 10;
338             steps += `<p class="notation-line indent-1">${diffTensVal} tens + ${diffOnesVal} ones - ${sOnes} ones</p>`;
339
340             let finalTens_calc = diffTensVal * 10;
341             let finalOnes_calc = diffOnesVal;
342             let didDecomposeTen = false;
343
344             if (diffOnesVal < sOnes) {
345                 didDecomposeTen = true;
346                 let onesNeeded = sOnes - diffOnesVal;
347                 if (diffTensVal > 0) {
348                     finalTens_calc = (diffTensVal - 1) * 10;
349                     let currentOnesPool = 10 + diffOnesVal;
350                     steps += `<p class="notation-line indent-1">${diffTensVal - 1} ten + 1 ten + ${diffOnesVal} ones - ${sOnes} ones</p>`;
351                     steps += `<p class="notation-line decompose-arrow"> DECOMPOSE</p>`;
352                     steps += `<p class="notation-line indent-1">${diffTensVal - 1} ten + 10 ones + ${diffOnesVal} ones - ${sOnes} ones</p>`;
353                     let onesLeftAfterCancel = currentOnesPool - sOnes;
354                     finalOnes_calc = onesLeftAfterCancel;
355                      steps += `<p class="notation-line indent-1">${diffTensVal - 1} ten + ${onesLeftAfterCancel} ones + <span class="cancel-group">${sOnes} ones - ${sOnes} ones</span></p>`;
356                      steps += `<p class="notation-line final-step">${diffTensVal - 1} ten + ${onesLeftAfterCancel} ones</p>`;
357                 } else {
358                     // Error: Cannot decompose hundred yet
359                     steps += `<p>ERROR: Cannot decompose - Not enough tens!</p>`;
360                      finalOnes_calc = diffOnesVal - sOnes;
361                 }
362             } else {
363                  finalOnes_calc = diffOnesVal - sOnes;
364                  steps += `<p class="notation-line final-step">${diffTensVal} tens + ${finalOnes_calc} ones</p>`;
365             }
366
367             const finalResult = (m - s);
```

```
                const finalResultTens = Math.floor((finalResult % 100) / 10); // Only tens part of final result
                const finalResultOnes = finalResult % 10;
                 steps += `<p class="notation-line problem">Result: ${finalResultTens} tens + ${finalResultOnes} ones = ${finalResult}</p>`;


                outputDiv.innerHTML = steps;
                typesetMath();

                // --- Call Diagram Function ---
                const mHuns_orig = Math.floor(m / 100) * 100;
                const mTens_orig_for_draw = Math.floor((m % 100) / 10) * 10;
                const mOnes_orig_for_draw = m % 10;
                const sHuns = Math.floor(s / 100) * 100;

                drawDecompositionDiagram('diagramDecompositionSVG',
                                         m, s, // Originals
                                         mHuns_orig, mTens_orig_for_draw, mOnes_orig_for_draw, // Initial M parts
                                         sHuns, sTens, sOnes, // S parts
                                         diffAfterTens, // Value after tens subtraction
                                         didDecomposeTen, // Flag
                                         finalResult);

            } catch (error) {
                console.error("Error in runDecompositionAutomaton:", error);
                outputDiv.textContent = `Error: ${error.message}`;
            }

        }; // End of runDecompositionAutomaton

        function typesetMath() { /* Placeholder */ }

        // Initialize on page load
        runDecompositionAutomaton();

    }); // End of DOMContentLoaded
</script>

</body>
</html>
```