Strategic Multiplicative Reasoning: Conversion to Bases and Ones (CBO)

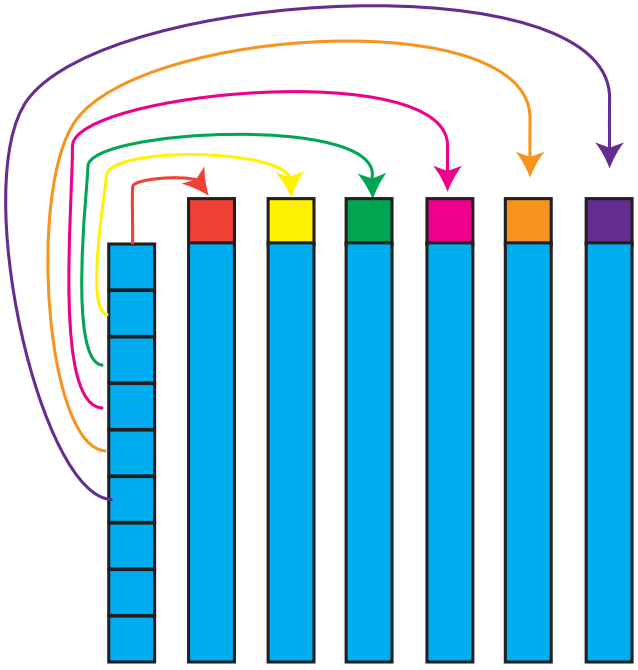
Compiled by: Theodore M. Savich

April 1, 2025

Transcript

Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** You have 7 mini cans of soda. Each can has 9 ounces of soda in it. How many ounces of soda do you have total?
- **George:** Well, you could take one of the 9 ounces and put an extra ounce into all other cans. That would give you 6 tens with 3 ounces leftover. So, 63.
- **Teacher:** Great!



$$\begin{aligned}\text{Seven} \times 9 &= \text{Six} \times 9 + 9 \\ &= \text{Six} \times 9 + 6 + 3 \\ &= \text{Six} \times (9 + 1) + 3 \\ &= \text{Six} \times 10 + 3 \\ &= 63\end{aligned}$$

Begin with groups of a known size. The objective is to form groups that equal the base size. To achieve this, break one group apart and redistribute its individual units to other groups until they form complete bases; repeat with additional groups if necessary. Typically, some units will remain ungrouped. The total count is then the sum of the complete bases and any leftover units.

Conversion to Bases and Ones (CBO)

Description of Strategy:

- **Objective:** Rearrange the items from groups to make complete base units by combining ones from different groups.
- **Method:** Break apart groups and redistribute ones to form full base units (e.g., tens).

Automaton Type:

Pushdown Automaton (PDA): The stack is used to represent the redistribution of ones in order to form complete base units.

Formal Description of the Automaton

We define the PDA as the 7-tuple

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

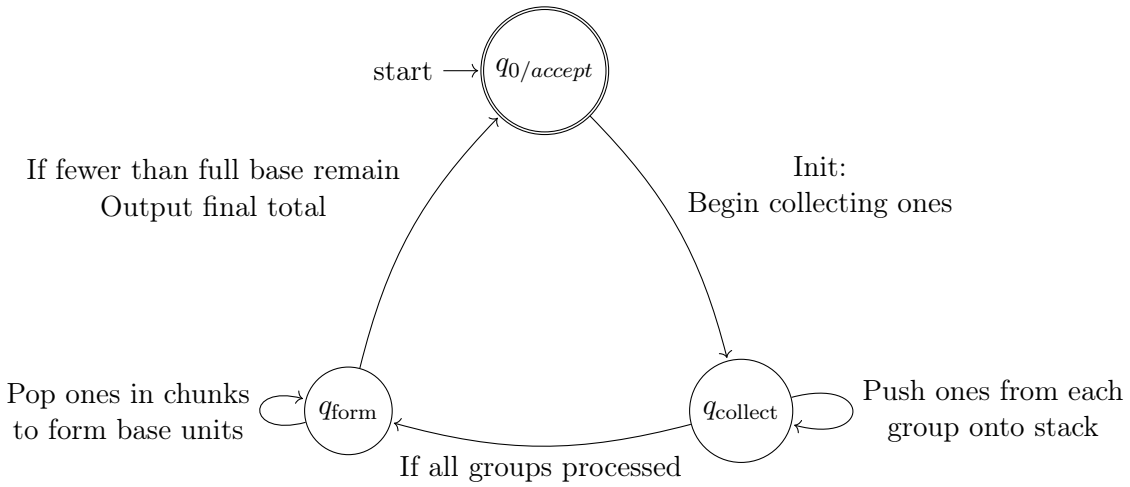
where:

- $Q = \{q_{0/accept}, q_{\text{collect}}, q_{\text{form}}\}$ is the set of states. Here, $q_{0/accept}$ serves as both the start and accept state.
- Σ is the input alphabet (encoding the group information, e.g., number of groups and ones per group).
- $\Gamma = \{Z_0\} \cup \{1\}$ is the stack alphabet, where Z_0 is the initial stack symbol and the symbol 1 represents a single one.
- $q_{0/accept}$ 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 by:

1. $\delta(q_{0/accept}, \text{“init”}, Z_0) = \{(q_{\text{collect}}, Z_0)\}$
   (Initialize the process to collect ones from the groups.)
2. In state q_{collect}: $\delta(q_{\text{collect}}, \varepsilon, x) = \{(q_{\text{collect}}, 1x)\}$ for any $x \in \Gamma$
   (For each group, push the ones (e.g., S ones) onto the stack.)
   Additionally, when all groups have been processed (i.e. a designated input symbol signals that the count of groups equals N), we have: $\delta(q_{\text{collect}}, \varepsilon, Z_0) = \{(q_{\text{form}}, Z_0)\}$.
3. In state q_{form}: $\delta(q_{\text{form}}, \varepsilon, 1) = \{(q_{\text{form}}, \varepsilon)\}$ (simulate popping a one) repeated until fewer than $BSize$ symbols remain on the stack. When fewer than $BSize$ ones remain (i.e., a full base unit cannot be formed), $\delta(q_{\text{form}}, \varepsilon, Z_0) = \{(q_{0/accept}, Z_0)\}$
   (Output the final result, which is implicitly represented by the distribution of ones on the stack.)

Automaton Diagram for Conversion to Bases and Ones



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Multiplication: Conversion to Bases and Ones (CBO - Redistribution)</title>
    <style>
        body {
            font-family: sans-serif;
            margin: 0;
            padding: 20px;
        }

        /* Layout improvements */
        .page-container {
            display: flex;
            flex-direction: column;
            gap: 15px;
        }

        .controls-container {
            position: sticky;
            top: 0;
            background-color: white;
            padding: 10px 0;
            border-bottom: 1px solid #ccc;
            z-index: 100;
            display: flex;
            flex-wrap: wrap;
            gap: 15px;
            align-items: center;
        }

        .input-group {
            display: flex;
            gap: 10px;
            align-items: center;
        }

        .visualization-container {
            display: flex;
            flex-direction: column;
            gap: 10px;
        }

        /* Step controls improvements */
        .step-controls {
            display: flex;
            align-items: center;
            gap: 10px;
            margin-left: auto;
        }

        /* Make the diagram more compact */
```

```
        #cboDiagram {
            border: 1px solid #d3d3d3;
            min-height: 500px; /* Increased from 400px */
            width: 100%;
        }

        /* More spacing between visualization sections */
        .section-spacer {
            margin-top: 20px;
            margin-bottom: 20px;
        }

        /* Existing styles */
        .diagram-label { font-size: 14px; display: block; margin-bottom: 10px; font-weight
            : bold;}
        .notation-line { margin: 0.2em 0; margin-left: 1em; font-family: monospace;}
        .notation-line.problem { font-weight: bold; margin-left: 0;}
        /* Block Styles */
        .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; }
        .initial-group-item { fill: teal; } /* Color for items in initial groups */
        .final-ten { fill: lightgreen; } /* Color for final ten blocks */
        .final-one { fill: gold; } /* Color for final one blocks */
        .redistribute-arrow {
            fill: none !important;
            stroke: orange;
            stroke-width: 1.5;
        }
        .redistribute-arrow-head {
            fill: orange;
            stroke: orange;
        }

        /* Animation controls */
        .step-button {
            padding: 8px 15px;
            background-color: #4a4a4a;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-family: sans-serif;
        }

        .step-button:hover {
            background-color: #666;
        }

        .step-button:disabled {
            background-color: #999;
            cursor: not-allowed;
        }
```

```
106
107         .step-indicator {
108             margin: 0 10px;
109             font-family: sans-serif;
110         }
111
112         .step-explanation {
113             background-color: #f9f9f9;
114             padding: 15px;
115             border-radius: 5px;
116             border: 1px solid #ddd;
117             margin: 10px 0;
118             font-family: sans-serif;
119             text-align: left;
120         }
121
122         /* Highlighting for current step */
123         .highlight-source {
124             stroke: red;
125             stroke-width: 2px;
126             animation: pulse 1s infinite alternate;
127         }
128
129         .highlight-target {
130             stroke: blue;
131             stroke-width: 2px;
132             animation: pulse 1s infinite alternate;
133         }
134
135         @keyframes pulse {
136             from { stroke-opacity: 0.5; }
137             to { stroke-opacity: 1; }
138         }
139
140         /* For arrows in the current step */
141         .current-step-arrow {
142             stroke: orange;
143             stroke-width: 2.5;
144             fill: none !important;
145         }
146
147         .current-step-arrow-head {
148             fill: orange;
149             stroke: orange;
150         }
151     </style>
152 </head>
153 <body>
154
155 <div class="page-container">
156     <h1>Strategic Multiplicative Reasoning: Conversion to Bases and Ones (CBO)</h1>
157
158     <!-- Sticky control panel -->
159     <div class="controls-container">
```

```
        <div class="input-group">
            <label for="cboGroups">Groups (N):</label>
            <input type="number" id="cboGroups" value="7" min="1">

            <label for="cboItems">Items per Group (S):</label>
            <input type="number" id="cboItems" value="9" min="1">

            <button class="action-button" onclick="runCBOAutomaton()">Calculate</button>
        </div>

        <!-- Step navigation controls -->
        <div class="step-controls">
            <button id="prevStepBtn" class="step-button" disabled> Previous</button>
            <span id="stepIndicator" class="step-indicator">Step 0/0</span>
            <button id="nextStepBtn" class="step-button">Next </button>
        </div>
    </div>

    <!-- Step explanation appears directly below controls -->
    <div id="stepExplanation" class="step-explanation">
        Click "Calculate" to begin.
    </div>

    <!-- Main visualization section -->
    <div class="visualization-container">
        <!-- Diagram is now above the notation so it's visible immediately -->
␣␣␣␣␣␣␣␣<h2>Diagram:</h2>
␣␣␣␣␣␣␣␣<svg id="cboDiagram" width="100%" height="600"></svg> <!-- Increased from 400 to 600 -->

␣␣␣␣␣␣␣␣<div id="outputContainer">
␣␣␣␣␣␣␣␣␣␣␣␣<h2>Notation:</h2>
␣␣␣␣␣␣␣␣␣␣␣␣<div id="cboOutput">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<!-- Text output will be displayed here -->
␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣</div>
␣␣␣␣</div>
</div>

<script>
␣␣␣␣// --- Helper SVG Functions ---
␣␣␣␣function drawBlock(svg, x, y, size, 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', size); rect.setAttribute('height', size);
␣␣␣␣␣␣␣␣rect.setAttribute('fill', fill);
␣␣␣␣␣␣␣␣rect.setAttribute('class', className);
␣␣␣␣␣␣␣␣svg.appendChild(rect);
␣␣␣␣␣␣␣␣return { x, y, width: size, height: size, type: 'o', cx: x + size/2, cy: y + size/2 }; // Add center point
␣␣␣␣}

␣␣␣␣function drawTenBlock(svg, x, y, width, height, fill, unitBlockSize) { // Keep vertical ten block
```

```
        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('class', 'ten-block-bg block');
        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('class', 'unit-block-inner');
            group.appendChild(unitBlock);
        }
        svg.appendChild(group);
        return {x, y, width, height, type: 't', cx: x + width/2, cy: y + height/2};
    }

    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.textContent = textContent;
        svg.appendChild(text);
    }

    function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy, arrowClass='redistribute-
        arrow', headClass='redistribute-arrow-head', arrowSize=4) {
        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);
        path.setAttribute('fill', 'none'); // Explicitly set fill to none
        svg.appendChild(path);

        const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        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 ---

    // --- Main CBO Automaton Function ---
    // Animation state variables
    let currentStep = 0;
    let totalSteps = 0;
    let animationSteps = [];

    let numGroups = 7;
    let itemsPerGroup = 9;
    let totalItems = numGroups * itemsPerGroup;
    let finalTensCount = Math.floor(totalItems / 10);
    let finalOnesCount = totalItems % 10;

    document.addEventListener('DOMContentLoaded', function() {
        const outputElement = document.getElementById('cboOutput');
        const groupsInput = document.getElementById('cboGroups');
        const itemsInput = document.getElementById('cboItems');
        const diagramSVG = document.getElementById('cboDiagram');

        const prevStepBtn = document.getElementById('prevStepBtn');
        const nextStepBtn = document.getElementById('nextStepBtn');
        const stepIndicator = document.getElementById('stepIndicator');
        const stepExplanation = document.getElementById('stepExplanation');

        if (!outputElement || !groupsInput || !itemsInput || !diagramSVG) {
            console.error("Required HTML elements not found!");
            return;
        }

        function numberToWord(num) {
            const words = ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven",
                "Eight", "Nine", "Ten", "Eleven", "Twelve"];
            if (num >= 0 && num < words.length) {
                return words[num];
            }
            return num.toString();
        }

        prevStepBtn.addEventListener('click', function() {
            if (currentStep > 0) {
                currentStep--;
                updateVisualization();
            }
        });

        nextStepBtn.addEventListener('click', function() {
            if (currentStep < totalSteps) {
                currentStep++;
                updateVisualization();
            }
        });

        function updateVisualization() {
```

```
            prevStepBtn.disabled = currentStep === 0;
            nextStepBtn.disabled = currentStep === totalSteps;
            stepIndicator.textContent = `Step ${currentStep} of ${totalSteps}`;

            if (currentStep === 0) {
                stepExplanation.innerHTML = `<p><strong>Initial State:</strong> We have ${numGroups} groups with ${itemsPerGroup} items each.</p>`;
            } else if (currentStep === totalSteps) {
                stepExplanation.innerHTML = `<p><strong>Final Result:</strong> After redistribution, we have ${finalTensCount} complete base-10 groups and ${finalOnesCount} individual items.</p>
                                             <p>This demonstrates that ${numGroups} × ${itemsPerGroup} = ${finalTensCount} * 10 + ${finalOnesCount}</p>`;
            } else {
                const step = animationSteps[currentStep - 1];
                stepExplanation.innerHTML = `<p><strong>Step ${currentStep}:</strong> Moving ${step.itemsToMove} item(s) from Group ${step.fromGroup + 1} to Group ${step.toGroup + 1}.</p>
                                             <p>This helps Group ${step.toGroup + 1} reach exactly 10 items, forming a complete base-10 unit.</p>`;
            }

            drawCBODiagram('cboDiagram', numGroups, itemsPerGroup, finalTensCount, finalOnesCount);

            document.getElementById('cboDiagram').scrollIntoView({ behavior: 'smooth', block: 'nearest' });
        }

        window.runCBOAutomaton = function() {
            try {
                numGroups = parseInt(groupsInput.value);
                itemsPerGroup = parseInt(itemsInput.value);

                if (isNaN(numGroups) || isNaN(itemsPerGroup) || numGroups <= 0 || itemsPerGroup <= 0) {
                    outputElement.textContent = "Please enter valid positive numbers";
                    diagramSVG.innerHTML = ''; return;
                }

                totalItems = numGroups * itemsPerGroup;
                finalTensCount = Math.floor(totalItems / 10);
                finalOnesCount = totalItems % 10;

                const numGroupsWord = numberToWord(numGroups);

                let output = `<h2>Conversion to Bases and Ones (CBO) - Notation</h2>\n\n`;
                output += `<p class="notation-line problem">${numGroupsWord} × ${itemsPerGroup} = ?</p>\n`;

                if (itemsPerGroup < 10 && numGroups > 1) {
                    const neededPerGroup = 10 - itemsPerGroup;
                    const groupsToComplete = numGroups - 1;
```

```
                    const totalNeeded = groupsToComplete * neededPerGroup;
                    const onesLeftInLastGroup = itemsPerGroup - totalNeeded;

                    if (onesLeftInLastGroup >= 0) {
                        output += `<p class="notation-line">= ${numberToWord(groupsToComplete)} ${itemsPerGroup} + ${itemsPerGroup}</p>\n`;
                        output += `<p class="notation-line">= ${numberToWord(groupsToComplete)} ${itemsPerGroup} + ${totalNeeded} + ${onesLeftInLastGroup}</p>\n`;
                        output += `<p class="notation-line">= ${numberToWord(groupsToComplete)} (${itemsPerGroup} + ${neededPerGroup}) + ${onesLeftInLastGroup}</p>\n`;
                        output += `<p class="notation-line">= ${numberToWord(groupsToComplete)} 10 + ${onesLeftInLastGroup}</p>\n`;
                        output += `<p class="notation-line">= ${groupsToComplete * 10} + ${onesLeftInLastGroup}</p>\n`;
                        output += `<p class="notation-line">= ${totalItems}</p>\n`;
                    } else {
                        output += `<p class="notation-line">= ${totalItems} (Direct Calculation)</p>\n`;
                    }
                } else {
                    output += `<p class="notation-line">= ${totalItems} (Direct Calculation)</p>\n`;
                }

                outputElement.innerHTML = output;

                drawCBODiagram('cboDiagram', numGroups, itemsPerGroup, finalTensCount, finalOnesCount);

                animationSteps = [];
                currentStep = 0;

                const neededPerGroup = (itemsPerGroup < 10) ? 10 - itemsPerGroup : 0;

                if (neededPerGroup > 0 && numGroups > 1) {
                    for (let g = 0; g < numGroups - 1; g++) {
                        animationSteps.push({
                            fromGroup: numGroups - 1,
                            toGroup: g,
                            itemsToMove: neededPerGroup
                        });
                    }
                }

                totalSteps = animationSteps.length + 1;

                updateVisualization();

                document.getElementById('cboDiagram').scrollIntoView({ behavior: 'smooth', block: 'nearest' });

            } catch (error) {
```

```
                    console.error("Error in runCBOAutomaton:", error);
                    outputElement.textContent = `Error: ${error.message}`;
                }
            };

            function drawCBODiagram(svgId, numGroups, itemsPerGroup, finalTensCount,
                finalOnesCount) {
                const svg = document.getElementById(svgId);
                if (!svg) return;
                svg.innerHTML = '';

                const svgWidth = parseFloat(svg.getAttribute('width'));
                const svgHeight = parseFloat(svg.getAttribute('height'));
                const blockUnitSize = 10;
                const tenBlockWidth = blockUnitSize;
                const tenBlockHeight = blockUnitSize * 10;
                const blockSpacing = 4;
                const groupSpacingX = 30;
                const sectionSpacingY = 180; // Increased from 150 to give more vertical
                space
                const startX = 30;
                let currentY = 40;
                const colorGroup = 'teal';
                const colorResultTen = 'lightgreen';
                const colorResultOne = 'gold';
                const arrowOffsetY = -15;
                const arrowControlOffsetY = -60;

                createText(svg, startX, currentY, `Initial State: ${numberToWord(numGroups)}
                groups of ${itemsPerGroup}`);
                currentY += 30;
                let currentX = startX;
                let section1MaxY = currentY;
                let initialGroupsData = [];

                for (let g = 0; g < numGroups; g++) {
                    let groupStartX = currentX;
                    let itemYOffset = 0;
                    let effectiveItemCount = itemsPerGroup;

                    if (currentStep > 0 && currentStep < totalSteps) {
                        for (let s = 0; s < currentStep; s++) {
                            const step = animationSteps[s];

                            if (g === step.fromGroup) {
                                effectiveItemCount -= step.itemsToMove;
                            }

                            if (g === step.toGroup) {
                                effectiveItemCount += step.itemsToMove;
                            }
                        }
                    }

```

```
                    for (let i = 0; i < effectiveItemCount; i++) {
                        let blockClass = 'initial-group-item';

                        if (currentStep > 0 && currentStep <= animationSteps.length) {
                            const step = animationSteps[currentStep - 1];

                            if (g === step.fromGroup && i >= effectiveItemCount - step.itemsToMove) {
                                blockClass += ' highlight-source';
                            }

                            if (g === step.toGroup && i >= itemsPerGroup) {
                                blockClass += ' highlight-target';
                            }
                        }

                        let blockInfo = drawBlock(svg, currentX, currentY + itemYOffset, blockUnitSize, colorGroup, blockClass);
                        initialGroupsData.push({
                            group: g,
                            item: i,
                            x: blockInfo.x,
                            y: blockInfo.y,
                            cx: blockInfo.cx,
                            cy: blockInfo.cy,
                            size: blockUnitSize
                        });

                        itemYOffset += blockUnitSize + blockSpacing;
                    }

                    currentX = groupStartX + blockUnitSize + groupSpacingX;
                    section1MaxY = Math.max(section1MaxY, currentY + itemYOffset);
                }

                if (currentStep > 0 && currentStep <= animationSteps.length) {
                    const step = animationSteps[currentStep - 1];

                    let sourceBlocks = initialGroupsData.filter(d => d.group === step.fromGroup)
                        .slice(-(step.itemsToMove));

                    let targetBlock = initialGroupsData.find(d => d.group === step.toGroup &&
                        d.item === itemsPerGroup - 1);

                    if (targetBlock && sourceBlocks.length > 0) {
                        sourceBlocks.forEach((sourceBlock, idx) => {
                            createCurvedArrow(svg,
                                sourceBlock.cx, sourceBlock.cy,
                                targetBlock.cx, targetBlock.y - 10,
                                (sourceBlock.cx + targetBlock.cx) / 2, Math.min(sourceBlock.cy, targetBlock.cy) - 50,
                                'current-step-arrow', 'current-step-arrow-head'
```

```
                                );
                        });
                    }
                }

                currentY = section1MaxY + sectionSpacingY;

                if (currentStep === totalSteps) {
                    // Add more vertical space between sections for clarity
                    currentY = section1MaxY + sectionSpacingY;

                    let finalSum = numGroups * itemsPerGroup;
                    createText(svg, startX, currentY, `Final Result (Converted to Base-10): ${finalSum}`);
                    currentY += 30;
                    currentX = startX;
                    let section2MaxY = currentY;

                    for (let i = 0; i < finalTensCount; i++) { drawTenBlock(svg, currentX, currentY, tenBlockWidth, tenBlockHeight, colorResultTen, blockUnitSize); currentX += tenBlockWidth + blockSpacing; section2MaxY = Math.max(section2MaxY, currentY + tenBlockHeight); }
                    let finalOnesY = currentY + Math.max(0, tenBlockHeight - (finalOnesCount * (blockUnitSize + blockSpacing)));
                    for (let i = 0; i < finalOnesCount; i++) { drawBlock(svg, currentX, finalOnesY + i * (blockUnitSize + blockSpacing), blockUnitSize, blockUnitSize, colorResultOne); section2MaxY = Math.max(section2MaxY, finalOnesY + (i+1)*(blockUnitSize+blockSpacing)); }
                    currentX += blockUnitSize + blockSpacing;

                    // Ensure SVG is tall enough to contain all content
                    const neededHeight = section2MaxY + 50; // Add 50px padding at bottom
                    if (neededHeight > parseInt(svg.getAttribute('height'))) {
                        svg.setAttribute('height', neededHeight);
                    }
                }

                // Add more space for larger problems
                if (numGroups > 7 || itemsPerGroup > 9) {
                    // Calculate additional vertical space needed for larger problems
                    const extraHeight = Math.max(0, numGroups - 7) * 40 + Math.max(0, itemsPerGroup - 9) * 20;
                    svg.setAttribute('height', 600 + extraHeight);
                }
            }

            runCBOAutomaton();

        });
    </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('../SMR_Multiplication_CBO.pdf',␣'_blank');
␣␣␣␣}
</script>

</body>
</html>
```

References

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