Strategic Multiplicative Reasoning: Division - Inverse of Distributive Reasoning

Compiled by: Theodore M. Savich

April 1, 2025

Transcript

Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** A man purchases a 56-inch party sub. Each guest at the party receives 8 inches of sub. How many guests can he feed?
- **Student:** I got 7 subs.
- **Teacher:** How did you get 7?
- **Student:** Well I broke 56 inches into 40 inches and 16 inches. I knew that you could make 5 subs with 40 inches, and 2 subs with 16 inches, which would give me a total of 7 subs.

To work on this strategy, it is helpful to list out "easily known multiples" of the known number of items in a group. Then you can use this to build up to the multiple that you don't know.

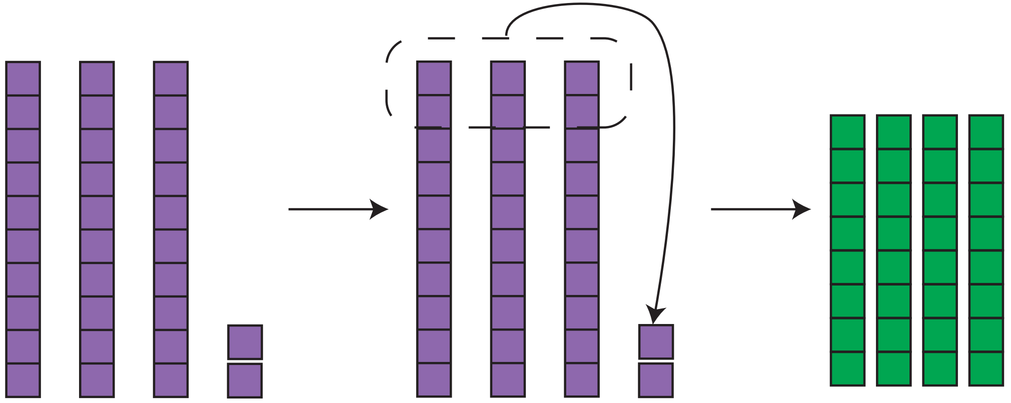
For example, the student likely knew the following:

$$\text{two 8s} = 16$$
$$\text{five 8s} = 40$$

He might have also known other 8s, like:

$$\text{three 8s} = 24$$
$$\text{eight 8s} = 64$$
$$\text{ten 8s} = 80$$

But then he used the two 8s and five 8s to help him solve his problem.



$$\begin{aligned}56 &=? \times 8\\ 56 &= 40 + 16\\ &= \text{five 8s} + \text{two 8s}\\ &= 5 \times 8 + 2 \times 8\\ &= 8(5+2)\\ &= 8 \times 7\\ \text{So, } 56 \div 8 &= 7\end{aligned}$$

Break the total number of items into multiples that are easier to work with. In other words, view the total as an unknown multiple of a given group size, then express it in terms of familiar or easily calculated multiples. This method essentially involves working backwards, highlighting the fact that division is the inverse of multiplication.

Inverse of the Distributive Property

Strategy Overview

The **Inverse of the Distributive Property** involves reversing the distributive property used in multiplication to aid in solving division problems. This strategy breaks down the total number of items into known multiples, facilitating easier division by calculating the quotient based on these decompositions.

Automaton Design

We design a **Transducing Automaton** (modeled here as a Pushdown Automaton with transduction capabilities) that applies the inverse distributive property by:

- Decomposing the total into known multiples M.
- Calculating the quotient Q by counting the number of times M fits into the total.

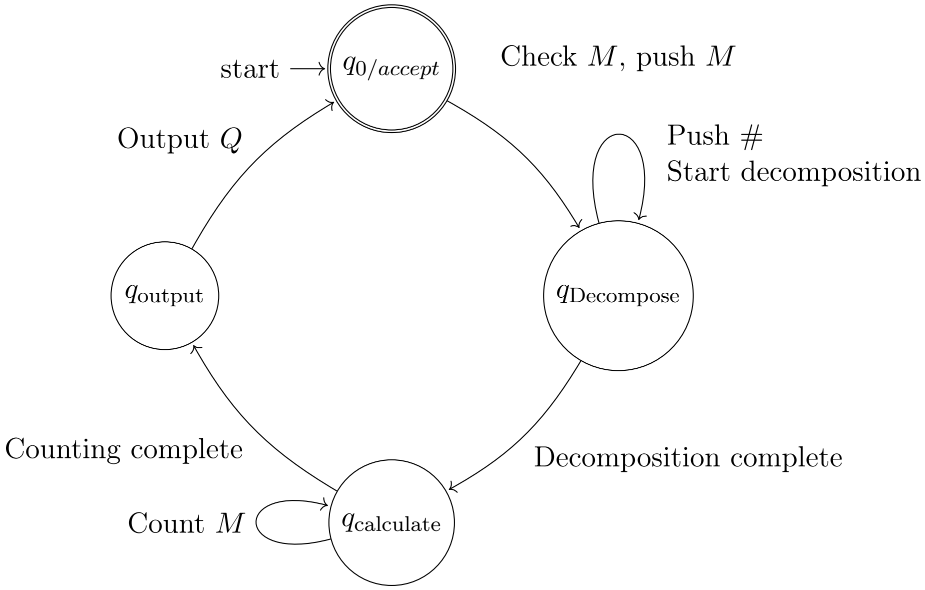
Components of the Automaton

- **States:**
  1. q_{start}: Start state.
  2. $q_{\text{Decompose}}$: Decomposes the total into known multiples.
  3. $q_{\text{calculate}}$: Calculates the quotient by counting multiples.
  4. q_{output}: Outputs the calculated quotient.
- **Input Alphabet:** $\Sigma = \{M\}$, where M represents a known multiple.
- **Stack Alphabet:** $\Gamma = \{\#, Q, M_n\}$:
  - $\#$ is the bottom-of-stack marker.
  - Q represents the quotient.
  - M_n represents an instance of the multiple M decomposed.
- **Initial Stack Symbol:** $\#$

Automaton Behavior

1. **Initialization:**
   - Start in q_{start}; push # onto the stack.
   - Transition to $q_{\text{decompose}}$ to begin decomposition.
2. **Decomposing Total:**
   - In $q_{\text{decompose}}$, for each known multiple M that fits into the remaining total, push M onto the stack.
   - Repeat until the total is fully decomposed.
   - Then transition to $q_{\text{calculate}}$.
3. **Calculating Quotient:**
   - In $q_{\text{calculate}}$, count the number of M symbols on the stack.
   - Push the count as Q onto the stack.
   - Transition to q_{output}.
4. **Outputting the Result:**
   - In q_{output}, read Q from the stack and output it as the quotient.

Circular Automaton Diagram



Example Execution

Problem: Divide 56 items by groups of 8 using the inverse distributive property.

1. **Start:**
   - Stack: #

2. **Decompose:**
    - 56 can be decomposed as 8×7.
    - Push 7 multiples of 8 onto the stack.
3. **Calculate Quotient:**
    - Count the 7 occurrences of M.
    - Push $Q = 7$ onto the stack.
4. **Output:**
    - The automaton outputs 7, meaning 7 groups of 8.

Recursive Handling of Decomposition

The automaton recursively checks for the largest multiple M that fits into the remaining total, ensuring an efficient decomposition and accurate quotient calculation.

HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Division: Inverse of Distributive Property</title>
    <style>
        body { font-family: sans-serif; }
        #invDistDiagram { border: 1px solid #d3d3d3; width: 100%; }
        #outputContainer { margin-top: 20px; }
        .diagram-label { font-size: 14px; display: block; margin-bottom: 5px; font-weight: bold;}
        .notation-line { margin: 0.2em 0; margin-left: 1em; font-family: monospace;}
        .notation-line.problem { font-weight: bold; margin-left: 0;}
        .notation-step { margin-bottom: 0.5em; }
        /* SVG Styles */
        .total-bar { fill: lightblue; stroke: black; stroke-width: 1; }
        .multiple-segment { stroke: black; stroke-width: 1; }
        .segment-label { font-size: 12px; text-anchor: middle; }
        .factor-label { font-size: 10px; text-anchor: middle; fill: #555; }
        .remainder-segment { fill: lightcoral; stroke: black; stroke-width: 1; }
        .quotient-calc { font-size: 14px; font-weight: bold; }
        .stopping-point { fill: red; }
        .number-line-label { font-size: 10px; fill: #333; }
    </style>
</head>
<body>

<h1>Strategic Multiplicative Reasoning: Division - Inverse of Distributive Property</h1>

<div>
    <label for="invDistTotal">Total (Dividend):</label>
    <input type="number" id="invDistTotal" value="56" min="1"> <!-- Example -->
</div>
```

```
<div>
    <label for="invDistGroupSize">Group Size (Divisor):</label>
    <input type="number" id="invDistGroupSize" value="8" min="1"> <!-- Example -->
</div>

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

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

<h2>Diagram:</h2>
<svg id="invDistDiagram" preserveAspectRatio="xMinYMin␣meet" viewBox="0␣0␣700␣300"></svg> <!-- Viewbox for scaling -->


<script>
    // --- Helper SVG Functions ---
    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 drawRect(svg, x, y, width, height, fill, className = ’’) {
        const rect = document.createElementNS("http://www.w3.org/2000/svg", ’rect’);
        rect.setAttribute(’x’, x); rect.setAttribute(’y’, y);
        rect.setAttribute(’width’, Math.max(0, width)); // Ensure width is not negative
        rect.setAttribute(’height’, height);
        rect.setAttribute(’fill’, fill);
        rect.setAttribute(’class’, className);
        svg.appendChild(rect);
    }
    // --- End Helper Functions ---


    // --- Main Inverse Distributive Automaton Function ---
    document.addEventListener(’DOMContentLoaded’, function() {
        const outputElement = document.getElementById(’invDistOutput’);
        const totalInput = document.getElementById(’invDistTotal’);
        const groupSizeInput = document.getElementById(’invDistGroupSize’);
        const diagramSVG = document.getElementById(’invDistDiagram’);

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

        window.runInvDistAutomaton = function() {
            try {
                const total = parseInt(totalInput.value);
                const divisor = parseInt(groupSizeInput.value);

                if (isNaN(total) || isNaN(divisor) || total <= 0 || divisor <= 0) {
                    outputElement.textContent = "Please␣enter␣valid␣positive␣numbers";
                    diagramSVG.innerHTML = ’’; return;
                }

                let output = ‘<h2>Inverse of Distributive Property</h2>\n\n‘;
                output += ‘<p class="notation-line␣problem">${total}  ${divisor} = ?</p>\n‘;

                // --- Decomposition Logic ---
                // Define "known" factors (could be dynamic later)
                const knownFactors = [10, 5, 2, 1]; // Prioritize larger factors
                let remainingTotal = total;
                let decomposition = []; // Stores { multiple: M, factor: k }
                let quotientFactors = []; // Stores k values

                output += ‘<p class="notation-line">Decompose ${total} into known multiples of ${divisor}:</p>\n‘;

                while (remainingTotal >= divisor) {
                    let foundMultiple = false;
                    for (const factor of knownFactors) {
                        let multiple = divisor * factor;
                        if (multiple > 0 && multiple <= remainingTotal) {
                            decomposition.push({ multiple: multiple, factor: factor });
                            quotientFactors.push(factor);
                            remainingTotal -= multiple;
                             output += ‘<p class="notation-line␣indent-1">- Found ${multiple} (${factor}  ${divisor}). Remainder: ${remainingTotal}</p>\n‘;
                            foundMultiple = true;
                            break; // Move to next iteration with reduced remainingTotal
                        }
                    }
                     // Safety break if no known multiple fits but remainder >= divisor
                     if (!foundMultiple) {
                         // This might happen if divisor itself is the only option left
                         if (divisor <= remainingTotal) {
                             let factor = 1;
                              let multiple = divisor;
                              decomposition.push({ multiple: multiple, factor: factor });
                              quotientFactors.push(factor);
                              remainingTotal -= multiple;
                              output += ‘<p class="notation-line␣indent-1">- Found ${multiple} (${factor}  ${divisor}). Remainder: ${remainingTotal}</p>\n‘;
                         } else {
```

```
                        console.warn("Could␣not␣decompose␣further,␣remainder:", remainingTotal);
                        break; // Exit loop
                    }
                }
            }

            const quotient = quotientFactors.reduce((sum, factor) => sum + factor, 0);
            const remainder = remainingTotal;

            output += ‘<br><p class="notation-line">Sum the factors of the multiples:</p>\n‘;
            output += ‘<p class="notation-line␣indent-1">${quotientFactors.join(’␣+␣’)} = ${quotient}</p>\n‘;
            output += ‘<br><p class="notation-line␣problem">Result: ${quotient}${remainder > 0 ? ‘ Remainder ${remainder}‘ : ’’}</p>‘;


            outputElement.innerHTML = output;
            typesetMath();

            // --- Draw Diagram ---
            drawInverseDistributiveDiagram(’invDistDiagram’, total, divisor, decomposition, quotient, remainder);

        } catch (error) {
            console.error("Error␣in␣runInvDistAutomaton:", error);
            outputElement.textContent = ‘Error: ${error.message}‘;
        }
    };

    function drawInverseDistributiveDiagram(svgId, total, divisor, decomposition, quotient, remainder) {
        const svg = document.getElementById(svgId);
        if (!svg) return;
        svg.innerHTML = ’’;

        const svgWidth = 700; // Use fixed width from viewBox
        const svgHeight = 300; // Use fixed height from viewBox
        const startX = 30;
        const endX = svgWidth - 30;
        const totalBarY = 50;
        const totalBarHeight = 30;
        const decompBarY = totalBarY + totalBarHeight + 40;
        const decompBarHeight = 30;
        const labelOffsetY = -10; // Above bars
        const factorLabelOffsetY = 15; // Below decomp bars

        // --- Scaling ---
        const availableWidth = endX - startX;
        const scale = availableWidth / total; // Scale based on total value

        // --- Draw Total Bar ---
```

```
            createText(svg, startX, totalBarY + labelOffsetY, `Total: ${total}`, 'diagram-label');
            drawRect(svg, startX, totalBarY, total * scale, totalBarHeight, 'lightblue', 'total-bar');

            // --- Draw Decomposition Segments ---
            createText(svg, startX, decompBarY + labelOffsetY, `Decomposition into Multiples of ${divisor}`);
            let currentX = startX;
            decomposition.forEach(part => {
                const segmentWidth = part.multiple * scale;
                drawRect(svg, currentX, decompBarY, segmentWidth, decompBarHeight, `hsl(${part.factor * 25}, 70%, 70%)`, 'multiple-segment'); // Vary color by factor
                // Label with the multiple value
                createText(svg, currentX + segmentWidth / 2, decompBarY + decompBarHeight / 2 + 5, `${part.multiple}`, 'segment-label', 'middle');
                 // Label with the multiplication fact
                 createText(svg, currentX + segmentWidth / 2, decompBarY + decompBarHeight + factorLabelOffsetY, `(${part.factor}  ${divisor})`, 'factor-label', 'middle');
                currentX += segmentWidth;
            });

            // --- Draw Remainder Segment ---
            if (remainder > 0) {
                const segmentWidth = remainder * scale;
                 drawRect(svg, currentX, decompBarY, segmentWidth, decompBarHeight, 'lightcoral', 'remainder-segment');
                 createText(svg, currentX + segmentWidth / 2, decompBarY + decompBarHeight / 2 + 5, `${remainder}`, 'segment-label', 'middle');
                 createText(svg, currentX + segmentWidth / 2, decompBarY + decompBarHeight + factorLabelOffsetY, `(Rem)`, 'factor-label', 'middle');
                 currentX += segmentWidth;
            }

            // --- Display Quotient Calculation ---
             let quotientY = decompBarY + decompBarHeight + factorLabelOffsetY + 40;
             createText(svg, startX, quotientY, `Quotient = ${decomposition.map(p => p.factor).join('␣+␣')} = ${quotient}`, 'quotient-calc');


            // --- Adjust ViewBox ---
             // No need to adjust height dynamically for this layout if 300 is enough
             // svg.setAttribute('viewBox', `0 0 ${svgWidth} ${svgHeight}`);
        }

        function typesetMath() { /* Placeholder */ }

        // Initialize on page load
        runInvDistAutomaton();

    }); // End DOMContentLoaded
</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_DIV_IDP.pdf', '_blank');
   }
</script>

</body>
</html>
```

References

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