Multiplication Strategies: Distributive Reasoning

Compiled by: Theodore M. Savich

April 1, 2025

Distributive Reasoning (DR)

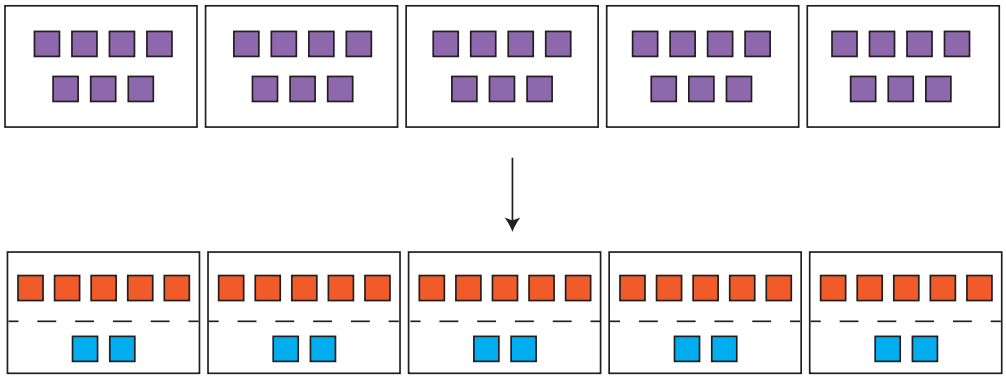
For equal groups multiplication:

$$\boxed{\text{number of groups}} \times \boxed{\text{number of items in each group}} = \boxed{\text{total number of items}}$$

Transcript

Video from Carpenter et al. (1999). Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** Sarah has five boxes of pretend turtles. There are seven turtles in each box. How many turtles does Sarah have?
- **Sarah:** 35?
- **Teacher:** How'd you get 35?
- **Sarah:** Because if there are seven turtles in each box and there's five boxes, take off two from each seven. Take two turtles away from each box; then there would be five in each box. And so you go 5, 10, 15, 20, 25. But then you have to add five 2s. And let's see, five ones would be five and so you just double it and so it would be ten. And then if you have 25 then it would be 35.



Notation Representing Sara's Solution:

$$\begin{aligned} 5 \times 7 &= 5 \times (5 + 2) \\ &= 5 \times 5 + 5 \times 2 \\ &= 25 + 10 \\ &= 35 \end{aligned}$$

Description of Strategy:

Objective: Distributive reasoning involves breaking apart the items within a group—or even the number of groups—to convert a difficult multiplication problem into several simpler ones. Alternatively, you can round the count in each group to a convenient base (or another useful number) and then subtract from each group to adjust the total.

Automaton Type:

Finite State Automaton with Registers (Counters): Used to manage partial results and sum them up.

Formal Description of the Automaton

We define the automaton as the tuple

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

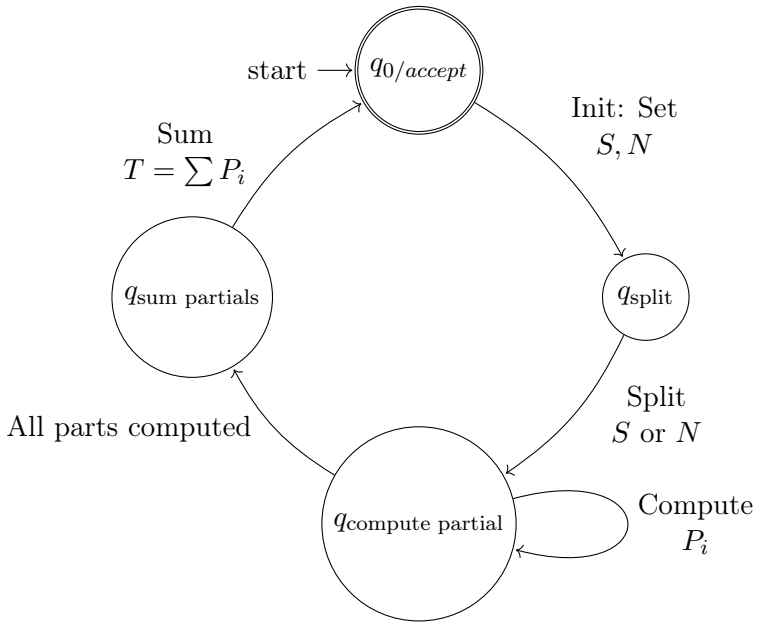
where:

- $Q = \{q_{0/accept}, q_{\text{split}}, q_{\text{compute\_partial}}, q_{\text{sum\_partials}}\}$ is the set of states. Here, $q_{0/accept}$ is both the start and the accept state.
- Σ is the input alphabet (used to initialize the problem parameters, e.g., the group size S and total groups N).
- $F = \{q_{0/accept}\}$ is the set of accepting states.
- $V = \{S, N, P_i, T\}$ is the set of registers, where:
  - S is the group size.
  - N is the total number of groups.
  - P_i are the partial products computed from the split.
  - T is the total product, $T = \sum_i P_i$.

The transition function δ is defined as follows:

1. $\delta(q_{0/accept}, \text{“}S, N\text{”}) = q_{\text{split}}$
   (Initialize the registers with S and N, then split one of the factors.)
2. $\delta(q_{\text{split}}, \varepsilon) = q_{\text{compute\_partial}}$
   (Split S or N into parts suitable for the distributive calculation.)
3. $\delta(q_{\text{compute\_partial}}, \varepsilon) = q_{\text{compute\_partial}}$
   (Loop to compute each partial product P_i.)
4. $\delta(q_{\text{compute\_partial}}, \varepsilon) = q_{\text{sum\_partials}}$
   (Once all partials are computed, proceed to sum them.)
5. $\delta(q_{\text{sum\_partials}}, \varepsilon) = q_{0/accept}$
   (Sum the partial products, setting $T = \sum_i P_i$, and output the final result.)

Automaton Diagram for Distributive Reasoning (DR)



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Distributive Reasoning Multiplication</title>
    <style>
        body { font-family: sans-serif; }
        #diagramDRSVG { border: 1px solid #d3d3d3; } /* Style SVG like canvas */
        #outputContainer { margin-top: 20px; }
        /* ... (CSS for cubes from previous example if needed) ... */
    </style>
</head>
<body>

    <h1>Multiplication Strategies: Distributive Reasoning</h1>

    <div>
        <label for="drGroups">Number of Groups:</label>
        <input type="number" id="drGroups" value="4">
    </div>
    <div>
        <label for="drItems">Items per Group:</label>
        <input type="number" id="drItems" value="9">
    </div>

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

    <div id="outputContainer">
        <h2>Explanation:</h2>
        <div id="drOutput">
```

```
            <!-- Text output will be displayed here -->
        </div>
    </div>

    <h2>Diagram:</h2>
    <svg id="diagramDRSVG" width="600" height="650"></svg> <!-- Increased height for
        subtraction diagram -->

    <script>
document.addEventListener('DOMContentLoaded', function() {
    const drOutputElement = document.getElementById('drOutput');
    const drGroupsInput = document.getElementById('drGroups');
    const drItemsInput = document.getElementById('drItems');
    const diagramDRSVG = document.getElementById('diagramDRSVG');

    if (!drOutputElement || !diagramDRSVG) {
        console.warn("Element drOutput or diagramDRSVG not found");
        return;
    }

    window.runDRAutomaton = function() {
        try {
            const groups = parseInt(drGroupsInput.value);
            const itemsPerGroup = parseInt(drItemsInput.value);

            if (isNaN(groups) || isNaN(itemsPerGroup) || groups <= 0 || itemsPerGroup <=
                0) {
                drOutputElement.textContent = "Please enter valid positive numbers for
                    groups and items per group";
                return;
            }

            let output = '';
            output += `<h2>Distributive Reasoning (DR)</h2>\n\n`;
            output += `<p><strong>Problem:</strong> ${groups} &times ${itemsPerGroup}</p>\
                n\n`;

            // --- Rounding Up and Subtracting Strategy ---
            let splitFactor1, splitFactor2, operationSymbol;
            if (itemsPerGroup >= 8 && itemsPerGroup <= 9) { // Apply for 8 or 9 items
                splitFactor1 = 10;
                splitFactor2 = 10 - itemsPerGroup;
                operationSymbol = '-'; // Subtraction for rounding up strategy
            } else if (itemsPerGroup > 5) { // Fallback to split into 5 and remainder if
                not 8 or 9 (or you can choose another default)
                splitFactor1 = 5;
                splitFactor2 = itemsPerGroup - 5;
                operationSymbol = '+'; // Addition for default split
            }
             else { // For smaller numbers, no split, or you can handle differently
                splitFactor1 = itemsPerGroup;
                splitFactor2 = 0;
                operationSymbol = '+'; // Addition, but effectively no split in calculation
            }
```

```
79
80
81          output += `<br>Step 1: Break down ${itemsPerGroup} into `;
82          if (operationSymbol === '-') {
83              output += `${splitFactor1} ${operationSymbol} ${splitFactor2}\n\n`;
84          }
85          else if (splitFactor2 === 0) {
86              output += `${splitFactor1} + 0\n\n`; // Handle case where splitFactor2 is 0
                    for cleaner output
87          }
88          else {
89              output += `${splitFactor1} + ${splitFactor2}\n\n`;
90          }
91
92
93          // Calculate using distributive property (handling subtraction now)
94          let part1Product, part2Product, total;
95          if (operationSymbol === '-') {
96              part1Product = groups * splitFactor1;
97              part2Product = groups * splitFactor2;
98              total = part1Product - part2Product; // Subtraction for final total
99          } else {
100             part1Product = groups * splitFactor1;
101             part2Product = groups * splitFactor2;
102             total = part1Product + part2Product; // Addition for default
103         }
104
105
106         output += `<br>Step 2: Apply distributive property:<br>`;
107         output += `${groups} &times ${itemsPerGroup} = ${groups} &times (${
                splitFactor1} ${operationSymbol} ${splitFactor2}) <br>`; // Multi-line
                notation
108         output += `= (${groups} &times ${splitFactor1}) ${operationSymbol} (${groups}
                &times ${splitFactor2})<br>`;
109         output += `(${groups} &times ${splitFactor1}) = ${part1Product}<br>`;
110         if (splitFactor2 !== 0 ) {
111             output += `(${groups} &times ${splitFactor2}) = ${part2Product}<br><br>`;
112         } else {
113             output += `(${groups} &times 0) = 0<br><br>`;
114         }
115
116
117         output += `<br>Step 3: Combine partial products:<br>`;
118         output += `${part1Product} ${operationSymbol} ${splitFactor2 !== 0 ?
                part2Product : 0} = ${total}<br><br>`; // Conditional output for
                part2Product
119
120         // Final result
121         output += `<strong>Result:</strong> ${groups} &times ${itemsPerGroup} = ${
                total}`;
122
123         drOutputElement.innerHTML = output;
124
125         // Draw Distributive Diagram (passing operationSymbol)
```

```
            drawDistributiveDiagram('diagramDRSVG', groups, itemsPerGroup, splitFactor1,
                splitFactor2, part1Product, part2Product, total, operationSymbol);


        } catch (error) {
            drOutputElement.textContent = `Error: ${error.message}`;
        }
    };


    function drawDistributiveDiagram(svgId, groups, itemsPerGroup, splitFactor1,
        splitFactor2, part1Product, part2Product, total, operationSymbol) {
        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 boxWidthBase = 40; // Base box width
        const boxHeightBase = 40; // Base box height
        const itemSize = 10;
        const boxSpacingX = 60;
        const boxSpacingY = 150; // Increased vertical spacing
        const startX = 50;
        let currentX = startX;
        let currentY = 50;
        const itemsPerRow = 2; // Items per row in boxes

        const colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'];
            // Item colors

        // --- Original Boxes ---
        let originalBoxesMaxHeight = 0; // Track max height for arrow positioning
        for (let i = 0; i < groups; i++) {
            // Responsive Box Size Calculation:
            const numItemRowsOriginal = Math.ceil(itemsPerGroup / itemsPerRow);
            const boxWidth = boxWidthBase; // Fixed width for now, can be adjusted if
                needed
            const boxHeight = Math.max(boxHeightBase, numItemRowsOriginal * itemSize * 1.5
                 + 20); // Adjust height based on items, ensure minimum height
            originalBoxesMaxHeight = Math.max(originalBoxesMaxHeight, boxHeight); //
                Update max height

            drawBox(svg, currentX, currentY, boxWidth, boxHeight, `Box ${i+1}`);
            for (let j = 0; j < itemsPerGroup; j++) {
                const itemX = currentX + 10 + (j % itemsPerRow) * itemSize * 1.2;
                const itemY = currentY + 15 + Math.floor(j / itemsPerRow) * itemSize * 1.2;
                drawItem(svg, itemX, itemY, itemSize, colors[j % colors.length]);
            }
            currentX += boxSpacingX;
        }

        // --- Arrow (Responsive Placement) ---
```

```
        const arrowStartY = currentY + originalBoxesMaxHeight + 20; // Use max height + spacing
        const arrowEndY = arrowStartY + 40;
        createArrow(svg, startX + (groups * boxSpacingX) / 2 - 10, arrowStartY, startX + (groups * boxSpacingX) / 2 - 10, arrowEndY);


        // --- Split Boxes (Split Factor 1 Part) ---
        currentX = startX;
        currentY = arrowEndY + 50;
        let split1BoxesMaxHeight = 0;
        for (let i = 0; i < groups; i++) {
            // Responsive Box Size Calculation for splitFactor1:
            const numItemRowsSplit1 = Math.ceil(splitFactor1 / itemsPerRow);
            const boxWidth = boxWidthBase;
            const boxHeight = Math.max(boxHeightBase, numItemRowsSplit1 * itemSize * 1.5 + 20);
            split1BoxesMaxHeight = Math.max(split1BoxesMaxHeight, boxHeight);

            drawBox(svg, currentX, currentY, boxWidth, boxHeight, `Box ${i+1}`);
␣␣␣␣␣␣␣␣␣␣␣␣for␣(let␣j␣=␣0;␣j␣<␣splitFactor1;␣j++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣const␣itemX␣=␣currentX␣+␣10␣+␣(j␣%␣itemsPerRow)␣*␣itemSize␣*␣1.2;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣const␣itemY␣=␣currentY␣+␣15␣+␣Math.floor(j␣/␣itemsPerRow)␣*␣itemSize␣*␣1.2;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣drawItem(svg,␣itemX,␣itemY,␣itemSize,␣colors[j␣%␣colors.length]);
␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣␣␣␣␣currentX␣+=␣boxSpacingX;
␣␣␣␣␣␣␣␣}

␣␣␣␣␣␣␣␣␣␣//␣---␣Split␣Boxes␣(Split␣Factor␣2␣Part)␣---
␣␣␣␣␣␣␣␣currentX␣=␣startX;
␣␣␣␣␣␣␣␣currentY␣+=␣boxSpacingY;␣//␣Keep␣consistent␣vertical␣spacing␣between␣split␣rows
␣␣␣␣␣␣␣␣for␣(let␣i␣=␣0;␣i␣<␣groups;␣i++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣//␣Responsive␣Box␣Size␣Calculation␣for␣splitFactor2:
␣␣␣␣␣␣␣␣␣␣␣␣const␣numItemRowsSplit2␣=␣Math.ceil(splitFactor2␣/␣itemsPerRow);
␣␣␣␣␣␣␣␣␣␣␣␣const␣boxWidth␣=␣boxWidthBase;
␣␣␣␣␣␣␣␣␣␣␣␣const␣boxHeight␣=␣Math.max(boxHeightBase,␣numItemRowsSplit2␣*␣itemSize␣*␣1.5␣+␣20);


␣␣␣␣␣␣␣␣␣␣␣␣drawBox(svg,␣currentX,␣currentY,␣boxWidth,␣boxHeight,␣`Box␣${i+1}`);
␣␣␣␣␣␣␣␣␣␣␣␣for␣(let␣j␣=␣0;␣j␣<␣splitFactor2;␣j++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣const␣itemX␣=␣currentX␣+␣10␣+␣(j␣%␣itemsPerRow)␣*␣itemSize␣*␣1.2;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣const␣itemY␣=␣currentY␣+␣15␣+␣Math.floor(j␣/␣itemsPerRow)␣*␣itemSize␣*␣1.2;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣drawItem(svg,␣itemX,␣itemY,␣itemSize,␣colors[j␣%␣colors.length]);
␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣␣␣␣␣currentX␣+=␣boxSpacingX;
␣␣␣␣␣␣␣␣}


␣␣␣␣␣␣␣␣//␣---␣Helper␣SVG␣drawing␣functions␣(same␣as␣before)␣---
␣␣␣␣␣␣␣␣function␣drawBox(svg,␣x,␣y,␣width,␣height,␣labelText)␣{␣/*␣...␣*/␣}
␣␣␣␣␣␣␣␣function␣drawItem(svg,␣x,␣y,␣size,␣fill)␣{␣/*␣...␣*/␣}
```

```
        function drawFadedItem(svg, x, y, size, fill) { /* ... */ }
        function createArrow(svg, x1, y1, x2, y2) { /* ... */ }
        //(SVG helper functions - same as in previous responses - keep them in your
            script)

        function drawBox(svg, x, y, width, height, labelText) {
            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', 'white');
            rect.setAttribute('stroke', 'black');
            rect.setAttribute('stroke-width', '1');
            svg.appendChild(rect);

            const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
            text.setAttribute('x', x + width / 2);
            text.setAttribute('y', y - 5); // Position label above box
            text.setAttribute('text-anchor', 'middle');
            text.setAttribute('font-size', '12px');
            text.textContent = labelText;
            svg.appendChild(text);
        }

        function drawItem(svg, x, y, size, fill) {
            const circle = document.createElementNS("http://www.w3.org/2000/svg", 'circle
                ');
            circle.setAttribute('cx', x);
            circle.setAttribute('cy', y);
            circle.setAttribute('r', size / 2);
            circle.setAttribute('fill', fill);
            circle.setAttribute('stroke', 'black');
            circle.setAttribute('stroke-width', '0.5');
            svg.appendChild(circle);
        }

        function drawFadedItem(svg, x, y, size, fill) {
            const circle = document.createElementNS("http://www.w3.org/2000/svg", 'circle
                ');
            circle.setAttribute('cx', x);
            circle.setAttribute('cy', y);
            circle.setAttribute('r', size / 2);
            circle.setAttribute('fill', fill);
            circle.setAttribute('fill-opacity', '0.3'); // Make it faded
            circle.setAttribute('stroke', 'lightgrey');
            circle.setAttribute('stroke-width', '0.5');
            svg.appendChild(circle);
        }


        function createArrow(svg, x1, y1, x2, y2) {
            const line = document.createElementNS("http://www.w3.org/2000/svg", 'line');
            line.setAttribute('x1', x1);
```

```
            line.setAttribute('y1', y1);
            line.setAttribute('x2', x2);
            line.setAttribute('y2', y2);
            line.setAttribute('stroke', 'black');
            line.setAttribute('stroke-width', '1');

            const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
            const arrowSize = 5;
            arrowHead.setAttribute('d', `M ${x2} ${y2} L ${x2 - arrowSize} ${y2 - arrowSize} L ${x2 + arrowSize} ${y2 - arrowSize} Z`);
            arrowHead.setAttribute('fill', 'black');

            svg.appendChild(line);
            svg.appendChild(arrowHead);
        }
    }

});
    </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_MULT_DR.pdf', '_blank');
        }
    </script>

</body>
</html>
```

References

Carpenter, T. P., Fennema, E., Franke, M. L., Levi, L., & Empson, S. B. (1999). *Children's mathematics: Cognitively guided instruction* [Includes supplementary material: Children's mathematics: Cognitively guided instruction – videotape logs]. Heinemann; The National Council of Teachers of Mathematics, Inc.

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