Multiplication Strategies: Distributive Reasoning

Compiled by: Theodore M. Savich

March 11, 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.
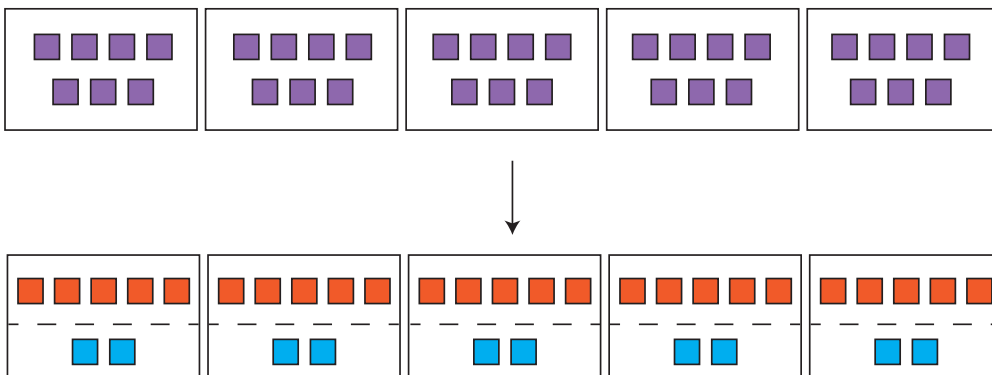
Notation Representing Sara's Solution:

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

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)
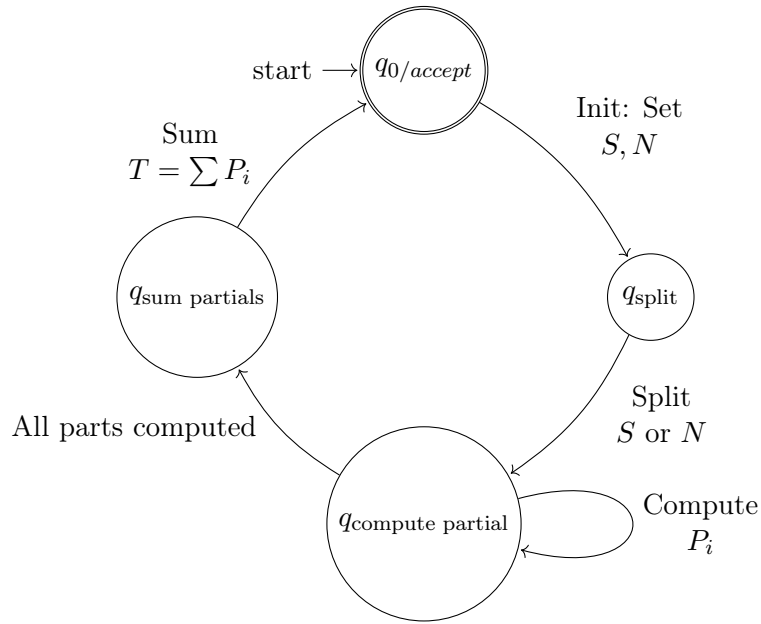
HTML Implementation

```html
<!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">
```

```
30              <!-- Text output will be displayed here -->
31          </div>
32      </div>
33
34      <h2>Diagram:</h2>
35      <svg id="diagramDRSVG" width="600" height="650"></svg> <!-- Increased height for
            subtraction diagram -->
36
37      <script>
38  document.addEventListener('DOMContentLoaded', function() {
39      const drOutputElement = document.getElementById('drOutput');
40      const drGroupsInput = document.getElementById('drGroups');
41      const drItemsInput = document.getElementById('drItems');
42      const diagramDRSVG = document.getElementById('diagramDRSVG');
43
44      if (!drOutputElement || !diagramDRSVG) {
45          console.warn("Element drOutput or diagramDRSVG not found");
46          return;
47      }
48
49      window.runDRAutomaton = function() {
50          try {
51              const groups = parseInt(drGroupsInput.value);
52              const itemsPerGroup = parseInt(drItemsInput.value);
53
54              if (isNaN(groups) || isNaN(itemsPerGroup) || groups <= 0 || itemsPerGroup <=
                    0) {
55                  drOutputElement.textContent = "Please enter valid positive numbers for
                        groups and items per group";
56                  return;
57              }
58
59              let output = '';
60              output += '<h2>Distributive Reasoning (DR)</h2>\n\n';
61              output += '<p><strong>Problem:</strong> ${groups} &times ${itemsPerGroup}</p>\
                    n\n';
62
63              // --- Rounding Up and Subtracting Strategy ---
64              let splitFactor1, splitFactor2, operationSymbol;
65              if (itemsPerGroup >= 8 && itemsPerGroup <= 9) { // Apply for 8 or 9 items
66                  splitFactor1 = 10;
67                  splitFactor2 = 10 - itemsPerGroup;
68                  operationSymbol = '-'; // Subtraction for rounding up strategy
69              } else if (itemsPerGroup > 5) { // Fallback to split into 5 and remainder if
                    not 8 or 9 (or you can choose another default)
70                  splitFactor1 = 5;
71                  splitFactor2 = itemsPerGroup - 5;
72                  operationSymbol = '+'; // Addition for default split
73              }
74              else { // For smaller numbers, no split, or you can handle differently
75                  splitFactor1 = itemsPerGroup;
76                  splitFactor2 = 0;
77                  operationSymbol = '+'; // Addition, but effectively no split in calculation
78              }
```

4

```javascript
79
80
          output += '<br>Step 1: Break down ${itemsPerGroup} into ';
          if (operationSymbol === '-') {
              output += '${splitFactor1} ${operationSymbol} ${splitFactor2}\n\n';
          }
          else if (splitFactor2 === 0) {
              output += '${splitFactor1} + 0\n\n'; // Handle case where splitFactor2 is 0
                  for cleaner output
          }
          else {
              output += '${splitFactor1} + ${splitFactor2}\n\n';
          }


          // Calculate using distributive property (handling subtraction now)
          let part1Product, part2Product, total;
          if (operationSymbol === '-') {
              part1Product = groups * splitFactor1;
              part2Product = groups * splitFactor2;
              total = part1Product - part2Product; // Subtraction for final total
          } else {
              part1Product = groups * splitFactor1;
              part2Product = groups * splitFactor2;
              total = part1Product + part2Product; // Addition for default
          }


          output += '<br>Step 2: Apply distributive property:<br>';
          output += '${groups} &times ${itemsPerGroup} = ${groups} &times (${
              splitFactor1} ${operationSymbol} ${splitFactor2}) <br>'; // Multi-line
              notation
          output += '= (${groups} &times ${splitFactor1}) ${operationSymbol} (${groups}
              &times ${splitFactor2})<br>';
          output += '(${groups} &times ${splitFactor1}) = ${part1Product}<br>';
          if (splitFactor2 !== 0 ) {
               output += '(${groups} &times ${splitFactor2}) = ${part2Product}<br><br>';
          } else {
              output += '(${groups} &times 0) = 0<br><br>';
          }


          output += '<br>Step 3: Combine partial products:<br>';
          output += '${part1Product} ${operationSymbol} ${splitFactor2 !== 0 ?
              part2Product : 0} = ${total}<br><br>'; // Conditional output for
              part2Product

          // Final result
          output += '<strong>Result:</strong> ${groups} &times ${itemsPerGroup} = ${
              total}';

          drOutputElement.innerHTML = output;

          // Draw Distributive Diagram (passing operationSymbol)
```

```
126            drawDistributiveDiagram('diagramDRSVG', groups, itemsPerGroup, splitFactor1,
                   splitFactor2, part1Product, part2Product, total, operationSymbol);
127
128
129        } catch (error) {
130            drOutputElement.textContent = `Error: ${error.message}`;
131        }
132    };
133
134
135    function drawDistributiveDiagram(svgId, groups, itemsPerGroup, splitFactor1,
               splitFactor2, part1Product, part2Product, total, operationSymbol) {
136        const svg = document.getElementById(svgId);
137        if (!svg) return;
138        svg.innerHTML = ''; // Clear SVG
139
140        const svgWidth = parseFloat(svg.getAttribute('width'));
141        const svgHeight = parseFloat(svg.getAttribute('height'));
142        const boxWidthBase = 40; // Base box width
143        const boxHeightBase = 40; // Base box height
144        const itemSize = 10;
145        const boxSpacingX = 60;
146        const boxSpacingY = 150; // Increased vertical spacing
147        const startX = 50;
148        let currentX = startX;
149        let currentY = 50;
150        const itemsPerRow = 2; // Items per row in boxes
151
152        const colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'];
                   // Item colors
153
154        // --- Original Boxes ---
155        let originalBoxesMaxHeight = 0; // Track max height for arrow positioning
156        for (let i = 0; i < groups; i++) {
157            // Responsive Box Size Calculation:
158            const numItemRowsOriginal = Math.ceil(itemsPerGroup / itemsPerRow);
159            const boxWidth = boxWidthBase; // Fixed width for now, can be adjusted if
                   needed
160            const boxHeight = Math.max(boxHeightBase, numItemRowsOriginal * itemSize * 1.5
                   + 20); // Adjust height based on items, ensure minimum height
161            originalBoxesMaxHeight = Math.max(originalBoxesMaxHeight, boxHeight); //
                   Update max height
162
163            drawBox(svg, currentX, currentY, boxWidth, boxHeight, `Box ${i+1}`);
164            for (let j = 0; j < itemsPerGroup; j++) {
165                const itemX = currentX + 10 + (j % itemsPerRow) * itemSize * 1.2;
166                const itemY = currentY + 15 + Math.floor(j / itemsPerRow) * itemSize * 1.2;
167                drawItem(svg, itemX, itemY, itemSize, colors[j % colors.length]);
168            }
169            currentX += boxSpacingX;
170        }
171
172        // --- Arrow (Responsive Placement) ---
```

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


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

189                drawBox(svg, currentX, currentY, boxWidth, boxHeight, `Box ${i+1}`);
190                for (let j = 0; j < splitFactor1; j++) {
191                    const itemX = currentX + 10 + (j % itemsPerRow) * itemSize * 1.2;
192                    const itemY = currentY + 15 + Math.floor(j / itemsPerRow) * itemSize *
         1.2;
193                    drawItem(svg, itemX, itemY, itemSize, colors[j % colors.length]);
194                }
195                currentX += boxSpacingX;
196            }

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


208                drawBox(svg, currentX, currentY, boxWidth, boxHeight, `Box ${i+1}`);
209                for (let j = 0; j < splitFactor2; j++) {
210                    const itemX = currentX + 10 + (j % itemsPerRow) * itemSize * 1.2;
211                    const itemY = currentY + 15 + Math.floor(j / itemsPerRow) * itemSize *
         1.2;
212                    drawItem(svg, itemX, itemY, itemSize, colors[j % colors.length]);
213                }
214                currentX += boxSpacingX;
215            }


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

```javascript
        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);
```

```
272            line.setAttribute('y1', y1);
273            line.setAttribute('x2', x2);
274            line.setAttribute('y2', y2);
275            line.setAttribute('stroke', 'black');
276            line.setAttribute('stroke-width', '1');

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

283            svg.appendChild(line);
284            svg.appendChild(arrowHead);
285          }
286      }

288  });
289      </script>

291  </body>
292  </html>
```

References

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

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