Subtraction Strategies: Sliding to Make Bases

Compiled by: Theodore M. Savich

March 30, 2025

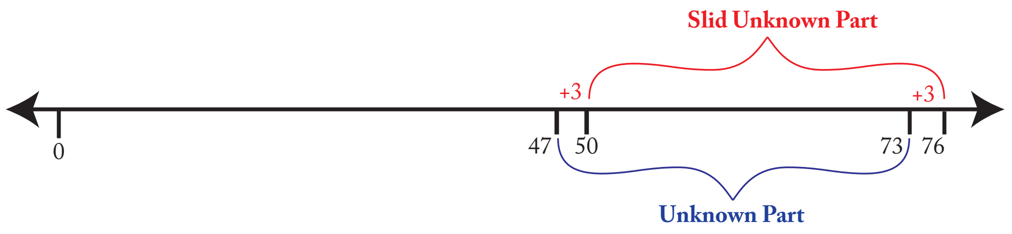
Transcript

Strategy descriptions and examples adapted from Hackenberg (2025). This is not based on a CGI video. I fake a student example.

- Teacher: John had 73 pieces of halloween candy. He gave 47 pieces to his friend. How many pieces of candy does John have left?
- Student: I can pretend I gave away 50 pieces and also pretend I had three more than I did. So that's like 76-50, which is 26.

Notation Representing Rita's Solution:

$$\begin{aligned} 73 - 47 &= \square \\ 73 + 3 &= 76 \\ 47 + 3 &= 50 \\ 73 - 47 &= 76 - 50 \\ &= 26 \end{aligned}$$



In the sliding strategy, you adjust both the number you're subtracting from (the whole) and the number being subtracted (the part) by the same amount. The goal is to shift the subtrahend into a 'friendly' number (usually a multiple of a base). By doing this, the difference between the adjusted values remains identical to the original difference, simplifying the subtraction process.

Description of Strategy

- **Objective:** Adjust both the minuend (known whole) and subtrahend (known part) by the same amount to make the subtraction easier, keeping the difference the same.

Automaton Type

Finite State Automaton (FSA): Adjustments are made consistently and can be tracked without additional memory.

Formal Description of the Automaton

We define the automaton as the tuple

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

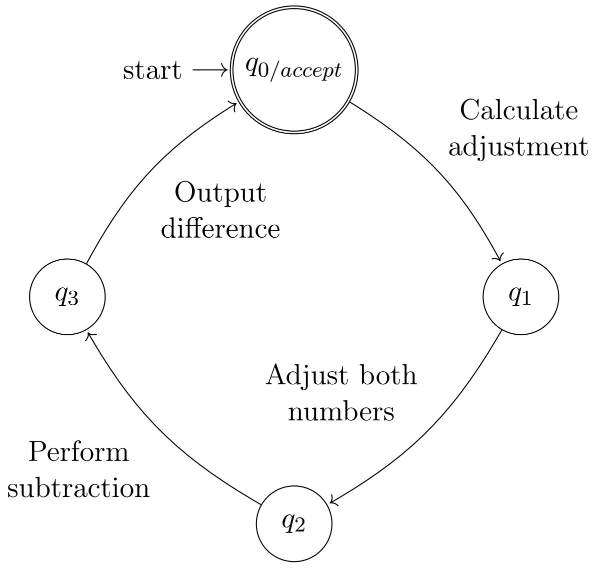
where:

- $Q = \{q_{0/accept},\, q_1,\, q_2,\, q_3\}$ is the set of states.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the input alphabet (representing the digits of the minuend M and subtrahend S).
- $q_{0/accept}$ 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 follows:

1. $\delta(q_{0/accept},\, \text{“}M, S\text{”}) = q_1$ (Calculate the adjustment needed to make the subtrahend a base multiple.)
2. $\delta(q_1,\, \varepsilon) = q_2$ (Adjust both the minuend and subtrahend by the same amount.)
3. $\delta(q_2,\, \varepsilon) = q_3$ (Perform the subtraction on the adjusted numbers.)
4. $\delta(q_3,\, \varepsilon) = q_{0/accept}$ (Output the final difference.)

Automaton Diagram for Sliding to Make Bases



HTML Implementation

```html
<!DOCTYPE html>
<html>
<head>
    <title>Subtraction Strategies: Sliding to Make Bases</title>
    <style>
        body { font-family: sans-serif; }
        #diagramSlidingSVG { border: 1px solid #d3d3d3; }
        #outputContainer { margin-top: 20px; }
        .number-line-tick { stroke: black; stroke-width: 1; }
        .number-line-break { stroke: black; stroke-width: 1; } /* Solid for zig-zag */
        .number-line-label { font-size: 12px; text-anchor: middle; }
        .original-marker { fill: blue; }
        .adjusted-marker { fill: green; }
        .slide-arrow { fill: none; stroke: darkorange; stroke-width: 1.5; }
        .slide-arrow-head { fill: darkorange; stroke: darkorange; }
        .slide-label { font-size: 10px; fill: darkorange; text-anchor: middle; }
        .difference-bracket { stroke: red; stroke-width: 1.5; fill: none; }
        .difference-label { font-size: 12px; fill: red; text-anchor: middle; }
        .number-line-arrow { fill: black; stroke: black;} /* Arrowhead for the main line
            */
    </style>
</head>
<body>

<h1>Subtraction Strategies: Sliding to Make Bases</h1>

<div>
    <label for="slideMinuend">Minuend:</label>
    <input type="number" id="slideMinuend" value="73">
</div>
<div>
    <label for="slideSubtrahend">Subtrahend:</label>
    <input type="number" id="slideSubtrahend" value="47">
</div>

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

<div id="outputContainer">
    <h2>Explanation:</h2>
    <div id="slidingOutput">
        <!-- Text output will be displayed here -->
    </div>
</div>

<h2>Diagram:</h2>
<svg id="diagramSlidingSVG" width="700" height="300"></svg>

<script>
document.addEventListener('DOMContentLoaded', function() {
    const outputElement = document.getElementById('slidingOutput');
    const minuendInput = document.getElementById('slideMinuend');
    const subtrahendInput = document.getElementById('slideSubtrahend');
    const diagramSVG = document.getElementById('diagramSlidingSVG');

    // --- Helper SVG Functions ---
     function createText(svg, x, y, textContent, className = 'number-line-label') {
        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', 'middle');
        text.textContent = textContent;
        svg.appendChild(text);
    }

    function drawTick(svg, x, y, size, colorClass = '') { // Added colorClass option
        const tick = document.createElementNS('http://www.w3.org/2000/svg', 'line');
        tick.setAttribute('x1', x);
        tick.setAttribute('y1', y - size / 2);
        tick.setAttribute('x2', x);
        tick.setAttribute('y2', y + size / 2);
        tick.setAttribute('class', `number-line-tick ${colorClass}`.trim()); // Apply color class if provided
        tick.setAttribute('stroke', colorClass ? 'currentColor' : 'black'); // Use CSS color or default black
        svg.appendChild(tick);
    }

     function drawScaleBreakSymbol(svg, x, y) {
        const breakOffset = 4;
        const breakHeight = 8;
        const breakLine1 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
        breakLine1.setAttribute('x1', x - breakOffset); breakLine1.setAttribute('y1', y - breakHeight);
        breakLine1.setAttribute('x2', x + breakOffset); breakLine1.setAttribute('y2', y + breakHeight);
        breakLine1.setAttribute('class', 'number-line-break'); svg.appendChild(breakLine1);
        const breakLine2 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
        breakLine2.setAttribute('x1', x + breakOffset); breakLine2.setAttribute('y1', y - breakHeight);
        breakLine2.setAttribute('x2', x - breakOffset); breakLine2.setAttribute('y2', y + breakHeight);
        breakLine2.setAttribute('class', 'number-line-break'); svg.appendChild(breakLine2);
    }

    function createStraightArrow(svg, x1, y1, x2, y2, arrowClass = 'slide-arrow', headClass = 'slide-arrow-head', arrowSize = 5) {
        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('class', arrowClass);
        svg.appendChild(line);

        // Arrowhead pointing right assumed for slide
        const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        arrowHead.setAttribute('d', `M ${x2 - arrowSize} ${y2 - arrowSize/2} L ${x2} ${y2}
             L ${x2 - arrowSize} ${y2 + arrowSize/2} Z`);
        arrowHead.setAttribute('class', headClass);
        svg.appendChild(arrowHead);
    }

    function drawDifferenceBracket(svg, x1, x2, y, label, colorClass = 'difference-') {
        const bracketHeight = 10;
        const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        path.setAttribute('d', `M ${x1} ${y - bracketHeight} L ${x1} ${y} L ${x2} ${y} L $
            {x2} ${y - bracketHeight}`);
        path.setAttribute('class', `${colorClass}bracket`);
        svg.appendChild(path);
        createText(svg, (x1 + x2) / 2, y + 15, label, `${colorClass}label`);
    }
    // --- End Helper Functions ---


    // --- Main Sliding Automaton Function ---
    window.runSlidingAutomaton = function() {
        try {
            const minuend = parseInt(minuendInput.value);
            const subtrahend = parseInt(subtrahendInput.value);

            if (isNaN(minuend) || isNaN(subtrahend)) {
               outputElement.textContent = 'Please␣enter␣valid␣numbers␣for␣Minuend␣and␣
                   Subtrahend';
               diagramSVG.innerHTML = ''; return;
            }
             if (subtrahend > minuend) {
                outputElement.textContent = 'Subtrahend␣cannot␣be␣greater␣than␣Minuend.';
                diagramSVG.innerHTML = ''; return;
             }

            let output = `<h2>Sliding to Make Bases</h2>\n\n`;
            output += `<p><strong>Problem:</strong> ${minuend} - ${subtrahend}</p>\n\n`;

            // Calculate adjustment (usually round subtrahend UP)
            const adjustment = (10 - (subtrahend % 10)) % 10;

            const adjustedMinuend = minuend + adjustment;
            const adjustedSubtrahend = subtrahend + adjustment;
            const difference = adjustedMinuend - adjustedSubtrahend; // Should equal
               minuend - subtrahend

            if (adjustment > 0) {
                output += `Step 1: Calculate adjustment to make ${subtrahend} a multiple
                    of 10.\n`;
                output += `<p>Adjustment = +${adjustment}</p>\n`;
                output += `Step 2: Adjust (slide) both numbers by +${adjustment}.\n`
                output += `<p>New Minuend: ${minuend} + ${adjustment} = ${adjustedMinuend
                    }</p>\n`;
                output += `<p>New Subtrahend: ${subtrahend} + ${adjustment} = ${adjustedSubtrahend}</p>\n`;
                output += `Step 3: Subtract adjusted numbers.\n`;
                output += `<p>${adjustedMinuend} - ${adjustedSubtrahend} = ${difference}</p>\n\n`;
            } else {
                output += `Subtrahend ${subtrahend} is already a multiple of 10. No slide needed.\n`;
                output += `<p>Direct Subtraction: ${minuend} - ${subtrahend} = ${difference}</p>\n\n`;
            }


            output += `<strong>Result:</strong> ${difference}`;
            outputElement.innerHTML = output;
            typesetMath();

            // Draw Diagram
            drawSlidingNumberLine(diagramSVG, minuend, subtrahend, adjustedMinuend, adjustedSubtrahend, adjustment, difference);

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

    function drawSlidingNumberLine(svg, M, S, M_adj, S_adj, adj, diff) {
         if (!svg || typeof svg.setAttribute !== 'function') { console.error("Invalid␣SVG␣element..."); return; }
         svg.innerHTML = '';

         const svgWidth = parseFloat(svg.getAttribute('width'));
         const svgHeight = parseFloat(svg.getAttribute('height'));
         const startX = 50;
         const endX = svgWidth - 50;
         const numberLineY = svgHeight * 0.6; // Position number line lower
         const tickHeight = 10;
         const labelOffsetY = 20; // Offset for labels below line
         const slideArrowY = numberLineY - 40; // Y position for slide arrows
         const diffBracketY = numberLineY + 40; // Y position for difference bracket
         const arrowSize = 5;
         const scaleBreakThreshold = 40;

         // Determine range for scaling
         let diagramMin = Math.min(0, S);
         let diagramMax = M_adj; // Need to show the adjusted minuend

         // Calculate scale and handle potential break
         let displayRangeStart = diagramMin;
         let scaleStartX = startX;
         let drawScaleBreak = false;

         if (diagramMin > scaleBreakThreshold) { // Break logic focuses on start
```

```
            displayRangeStart = diagramMin - 10;
            scaleStartX = startX + 30;
            drawScaleBreak = true;
            drawScaleBreakSymbol(svg, scaleStartX - 15, numberLineY);
            drawTick(svg, startX, numberLineY, tickHeight);
            createText(svg, startX, numberLineY + labelOffsetY, '0');
        } else {
            displayRangeStart = 0; // Include 0
            drawTick(svg, startX, numberLineY, tickHeight);
            createText(svg, startX, numberLineY + labelOffsetY, '0');
        }

        const displayRangeEnd = diagramMax + 10;
        const displayRange = Math.max(displayRangeEnd - displayRangeStart, 1);
        const scale = (endX - scaleStartX) / displayRange;

        // Function to convert value to X coordinate
        function valueToX(value) {
            if (value < displayRangeStart && drawScaleBreak) { return scaleStartX - 10; }
            const scaledValue = scaleStartX + (value - displayRangeStart) * scale;
            return Math.max(scaleStartX, Math.min(scaledValue, endX));
        }

        // Draw main line segment
        const mainLineStartX = valueToX(displayRangeStart);
        const mainLineEndX = valueToX(displayRangeEnd);
        const numberLine = document.createElementNS('http://www.w3.org/2000/svg', 'line');
        numberLine.setAttribute('x1', mainLineStartX); numberLine.setAttribute('y1', numberLineY);
        numberLine.setAttribute('x2', mainLineEndX); numberLine.setAttribute('y2', numberLineY);
        numberLine.setAttribute('class', 'number-line-tick'); svg.appendChild(numberLine);
        // Add arrowhead
        const mainArrowHead = document.createElementNS('http://www.w3.org/2000/svg', 'path');
        mainArrowHead.setAttribute('d', `M ${mainLineEndX - arrowSize} ${numberLineY - arrowSize/2} L ${mainLineEndX} ${numberLineY} L ${mainLineEndX - arrowSize} ${numberLineY + arrowSize/2} Z`);
        mainArrowHead.setAttribute('class', 'number-line-arrow'); svg.appendChild(mainArrowHead);


        // Mark Original Points (Blue)
        const xS = valueToX(S);
        const xM = valueToX(M);
        drawTick(svg, xS, numberLineY, tickHeight, 'original-marker');
        createText(svg, xS, numberLineY + labelOffsetY, S.toString(), 'original-marker');
        drawTick(svg, xM, numberLineY, tickHeight, 'original-marker');
        createText(svg, xM, numberLineY + labelOffsetY, M.toString(), 'original-marker');

        if (adj > 0) { // Only draw adjusted points and arrows if there was a slide
            // Mark Adjusted Points (Green)
            const xS_adj = valueToX(S_adj);
            const xM_adj = valueToX(M_adj);
            drawTick(svg, xS_adj, numberLineY, tickHeight, 'adjusted-marker');
            createText(svg, xS_adj, numberLineY + labelOffsetY + 15, S_adj.toString(), '
                adjusted-marker'); // Offset adjusted label slightly more
            drawTick(svg, xM_adj, numberLineY, tickHeight, 'adjusted-marker');
            createText(svg, xM_adj, numberLineY + labelOffsetY + 15, M_adj.toString(), '
                adjusted-marker'); // Offset adjusted label

            // Draw Slide Arrows (Orange)
            createStraightArrow(svg, xS, slideArrowY, xS_adj, slideArrowY);
            createText(svg, (xS + xS_adj) / 2, slideArrowY - 10, `+${adj}`, 'slide-label'
                );
            createStraightArrow(svg, xM, slideArrowY, xM_adj, slideArrowY);
            createText(svg, (xM + xM_adj) / 2, slideArrowY - 10, `+${adj}`, 'slide-label'
                );

            // Draw Difference Bracket (Red) below adjusted points
             drawDifferenceBracket(svg, xS_adj, xM_adj, diffBracketY, `Difference = ${
                 diff}`);
        } else {
            // Draw Difference Bracket (Red) below original points if no slide
             drawDifferenceBracket(svg, xS, xM, diffBracketY, `Difference = ${diff}`);
        }

    }

    function typesetMath() { /* Placeholder */ }

    // Initial run on page load
    runSlidingAutomaton();

});
</script>

</body>
</html>
```

References

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