Strategic Multiplicative Reasoning: Division - Conversion to Groups Other than Bases (CBO)

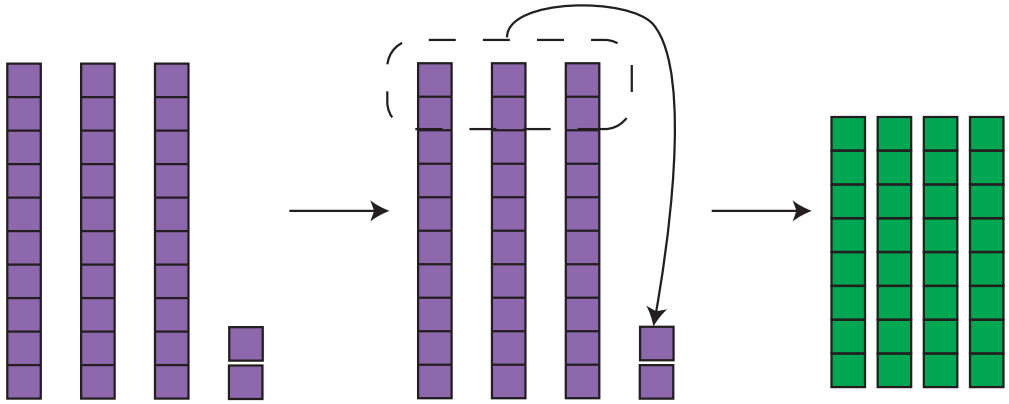
Compiled by: Theodore M. Savich

March 31, 2025

Transcript

Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:**There are 8 pencils in a package. You buy some packages at the store, and have a total of 32 pencils. Determine the total number of packages you bought.
- **Student:** Well, I know that there are 3 groups of ten pencils in 32 pencils. I could make 3 packages of 8 pencils from the 3 groups of ten. Then I would be left with 8 pencils (2 from each ten and 2 more from the units), which would make a fourth package of 8.
- **Teacher:** Great!



$$\begin{aligned} 32 &= 3 \times 10 + 2 \\ &= 3 \times 8 + 3 \times 2 + 2 \\ &= 3 \times 8 + 8 \\ &= 4 \times 8 \\ 32 \div 8 &= 4 \end{aligned}$$

Thus, the total number of packages bought is 4.

Begin with a collection of bases and individual ones—this represents the total number of items. Identify the fixed group size contained within each base. Then, remove an equal number of individual ones from every base to form complete groups of that size, and combine any leftover ones to create additional groups.

For CGOB, using block diagrams works well because they illustrate how an equal number of ones is taken from each base to form groups of the predetermined size, and how those ones can be rearranged to complete the groups.

Conversion to Groups Other than Bases

Strategy Overview

Conversion to Groups Other than Bases involves reorganizing the total number of items into groups that are not aligned with the base system (e.g., base twelve). This strategy is useful when the group size does not neatly fit into the base units, requiring a flexible approach to grouping.

Automaton Design

We design a **Pushdown Automaton (PDA)** that converts a total number of items into groups of a specified size (which is different from the standard base). The PDA uses two stacks: one for tracking the total items and another for forming the new groups.

Components of the PDA

- **States:**
  1. q_{start}: Start state.
  2. q_{read}: Reads the total number of items.
  3. q_{group}: Forms new groups.
  4. q_{output}: Outputs the new grouping.
  5. q_{accept}: Accepting state.
- **Input Alphabet:** $\Sigma = \{E\}$, where E represents an element.
- **Stack Alphabet:** $\Gamma = \{\#, G, E_1, E_2, \ldots\}$, where:
  - $\#$ is the bottom-of-stack marker.
  - G represents a group identifier.
  - E_n represents an element (or the count of elements in a group).
- **Initial Stack Symbol:** $\#$

Automaton Behavior

1. **Initialization:**
   - Begin in q_{start} and push $\#$ onto the stack.
   - Transition to q_{read} to start reading the total number of items.
2. **Reading Total Items:**
   - In q_{read}, for each element E read from the input, push E onto the stack.
   - When all inputs have been read, transition to q_{group}.
3. **Forming New Groups:**

- In q_{group}, pop a fixed number n of E symbols (representing the desired group size) and then push a group identifier G onto the stack.
- Repeat this process until all elements have been grouped.

4. **Outputting New Grouping:**
   - In q_{output}, traverse the stack to read the new grouping.
   - Transition to q_{accept} when the grouping is complete.

Automaton Diagram

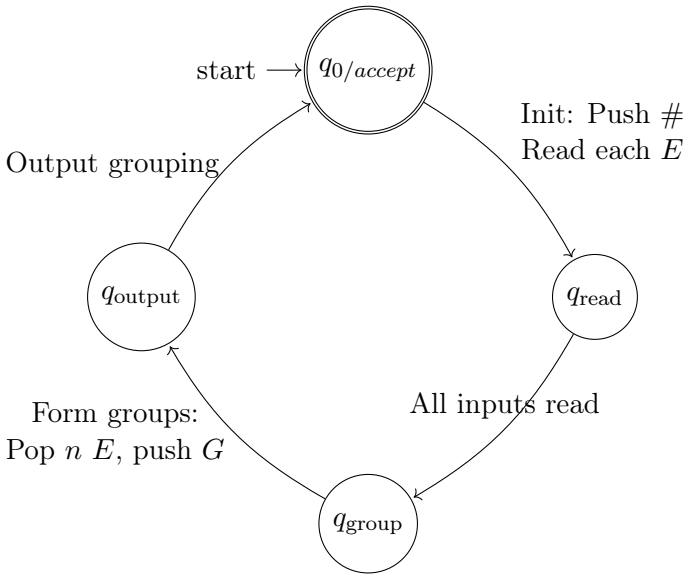


Figure 1: PDA for Conversion to Groups Other than Bases

Example Execution

Problem: Convert 32 items into groups of 8 in base ten.

1. **Initialization:**
   - Start with the stack: #.
2. **Reading Total Items:**
   - Read 32 elements, pushing 32 E symbols onto the stack.
3. **Forming Groups of 8:**
   - Pop 8 E symbols and push G onto the stack.
   - Repeat this process 4 times to form 4 groups.
4. **Final Stack Configuration:** $\#\ G\ G\ G\ G$

Recursive Handling of Group Formation

The PDA recursively forms groups by repeatedly popping a fixed number of elements and pushing a group identifier until all elements are grouped. This ensures the conversion of the total into groups that are not aligned with the standard base system.

HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Division: Conversion to Groups Other than Bases (CGOB)</title>
    <style>
        body { font-family: sans-serif; }
        #cgobDiagram { border: 1px solid #d3d3d3; min-height: 500px; width: 100%; }
        #outputContainer { margin-top: 20px; }
        .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;}
        .notation-step { margin-bottom: 0.5em; }
        /* 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-total-block { fill: purple; } /* Color for initial total */
        .final-group-block { fill: lightgreen; } /* Color for final groups */
        .regrouped-block { fill: orange; opacity: 0.7; }
        .remainder-block { fill: lightblue; }
        .regroup-arrow {
            fill: none; stroke: orange; stroke-width: 1.5;
            marker-end: url(#arrowhead-orange);
        }
        .regroup-grouping {
            fill: none; stroke: #555; stroke-width: 1;
            stroke-dasharray: 4 4;
        }
        .section-container {
            margin-bottom: 20px;
            padding: 10px;
            border-radius: 5px;
            background-color: #f9f9f9;
        }
        .final-group {
            stroke: blue;
            stroke-width: 1;
            fill: none;
            stroke-dasharray: 5 3;
        }
    </style>
</head>
<body>

<h1>Strategic Multiplicative Reasoning: Division - Conversion to Groups Other than Bases
    (CGOB)</h1>

<div>
    <label for="cgobTotal">Total Items (Dividend):</label>
    <input type="number" id="cgobTotal" value="32" min="1">
```

```
</div>
<div>
    <label for="cgobGroupSize">Items per Group (Divisor):</label>
    <input type="number" id="cgobGroupSize" value="8" min="1">
</div>

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

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

<h2>Diagram:</h2>
<div style="overflow-x: auto; overflow-y: auto; max-height: 800px;">
    <svg id="cgobDiagram" preserveAspectRatio="xMinYMin meet">
        <defs>
            <marker id="arrowhead-orange" markerWidth="10" markerHeight="7" refX="9" refY=
                "3.5" orient="auto">
                <polygon points="0 0, 10 3.5, 0 7" fill="orange" />
            </marker>
        </defs>
    </svg>
</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 };
    }

    function drawTenBlock(svg, x, y, width, height, fill, unitBlockSize) {
        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 drawHundredBlock(svg, x, y, size, fill, unitBlockSize) {
        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', size); backgroundRect.setAttribute('height',
            size);
        backgroundRect.setAttribute('fill', fill);
        backgroundRect.setAttribute('class', 'hundred-block-bg␣block');
        group.appendChild(backgroundRect);

        for (let row = 0; row < 10; row++) {
            for (let col = 0; col < 10; col++) {
                const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
                    rect');
                unitBlock.setAttribute('x', x + col * unitBlockSize);
                unitBlock.setAttribute('y', y + row * 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: size, height: size, type: 'h', cx: x + size/2, cy: y + size
            /2};
    }

    function createText(svg, x, y, textContent, className = 'diagram-label', anchor = '
        start') {
        const uniqueId = 'text-' + Math.random().toString(36).substr(2, 9);

        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.setAttribute('id', uniqueId);
        text.textContent = textContent;

        if (className === 'diagram-label') {
```

```
        const background = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
        const padding = 3;
        const estimatedWidth = Math.max(7 * textContent.length, 30);
        const estimatedHeight = 16;

        let bgX = x - padding;
        if (anchor === 'middle') {
            bgX = x - (estimatedWidth / 2) - padding;
        } else if (anchor === 'end') {
            bgX = x - estimatedWidth - padding;
        }

        background.setAttribute('x', bgX);
        background.setAttribute('y', y - estimatedHeight + padding);
        background.setAttribute('width', estimatedWidth + (padding * 2));
        background.setAttribute('height', estimatedHeight + padding);
        background.setAttribute('fill', 'white');
        background.setAttribute('fill-opacity', '0.9');
        background.setAttribute('rx', '3');
        svg.appendChild(background);
    }

    svg.appendChild(text);
    return uniqueId;
}

function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy, arrowClass='regroup-arrow', headId='arrowhead-orange') {
    const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
    path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}`);
    path.setAttribute('class', arrowClass);
    path.setAttribute('marker-end', `url(#${headId})`);
    svg.appendChild(path);
}

document.addEventListener('DOMContentLoaded', function() {
    const outputElement = document.getElementById('cgobOutput');
    const totalInput = document.getElementById('cgobTotal');
    const groupSizeInput = document.getElementById('cgobGroupSize');
    const diagramSVG = document.getElementById('cgobDiagram');

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

    window.runCGOBAutomaton = function() {
        try {
            const totalItems = parseInt(totalInput.value);
            const groupSize = parseInt(groupSizeInput.value);

            if (isNaN(totalItems) || isNaN(groupSize) || totalItems <= 0 || groupSize <= 0) {
```

```
            outputElement.textContent = "Please␣enter␣valid␣positive␣numbers";
            diagramSVG.innerHTML = ''; return;
        }

        const numGroups = Math.floor(totalItems / groupSize);
        const remainder = totalItems % groupSize;

        generateCGOBNotation(outputElement, totalItems, groupSize, numGroups,
            remainder);
        drawCGOBDiagram('cgobDiagram', totalItems, groupSize, numGroups, remainder)
            ;

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

function generateCGOBNotation(outputElement, totalItems, groupSize, numGroups,
    remainder) {
    let output = `<h2>Conversion to Groups Other than Bases (CGOB) - Notation</h2
        >`;
    output += `<div class="notation-step"><p class="notation-line␣problem">${
        totalItems}  ${groupSize} = ?</p></div>`;

    const placeValues = decomposeNumber(totalItems);
    output += `<div class="notation-step"><p class="notation-line">Start with ${
        totalItems} = ${placeValues.join('␣+␣')}</p></div>`;

    let steps = [];
    let remainders = [];
    let regroupedItems = 0;
    let runningTotal = totalItems;

    let completeGroups = 0;

    for (let i = 0; i < placeValues.length; i++) {
        const placeValue = parseInt(placeValues[i]);
        if (placeValue === 0) continue;

        const base = Math.pow(10, placeValues.length - i - 1);
        const count = placeValue / base;

        if (base > 1) {
            const wholeGroups = Math.floor(base / groupSize);
            const leftover = base % groupSize;

            steps.push(`${count}  ${base} = ${count}  (${wholeGroups}  ${groupSize}
                 + ${leftover})`);
            steps.push(`= ${count * wholeGroups}  ${groupSize} + ${count}  ${
                leftover}`);

            const newGroups = count * wholeGroups;
            completeGroups += newGroups;
```

```
                regroupedItems += newGroups * groupSize;

                if (i > 0 || count * leftover > 0) {
                    steps.push(`= ${completeGroups}  ${groupSize} + ${count * leftover}
                        ${remainders.length > 0 ? '␣+␣' + remainders.join('␣+␣') : ''} =
                         ${totalItems}`);
                } else {
                    steps.push(`= ${completeGroups}  ${groupSize} = ${regroupedItems}`)
                        ;
                }

                if (leftover > 0) {
                    remainders.push(count * leftover);
                }
            } else {
                remainders.push(placeValue);
            }
        }

        if (remainders.length > 0) {
            const totalRemainder = remainders.reduce((sum, val) => sum + val, 0);
            steps.push(`Combined leftovers: ${remainders.join('␣+␣')} = ${
                totalRemainder}`);

            const additionalGroups = Math.floor(totalRemainder / groupSize);
            const finalRemainder = totalRemainder % groupSize;

            if (additionalGroups > 0) {
                steps.push(`Leftovers form ${additionalGroups} more group${
                    additionalGroups > 1 ? 's' : ''} of ${groupSize}${finalRemainder >
                    0 ? ` with ${finalRemainder} remaining` : ''}`);

                completeGroups += additionalGroups;
                steps.push(`= ${completeGroups}  ${groupSize}${finalRemainder > 0 ? ` +
                     ${finalRemainder}` : ''} = ${totalItems}`);
            } else if (totalRemainder > 0) {
                steps.push(`= ${completeGroups}  ${groupSize} + ${totalRemainder} = ${
                    totalItems}`);
            }
        }

        steps.forEach(step => {
            output += `<div class="notation-step"><p class="notation-line">${step}</p
                ></div>`;
        });

        output += `<div class="notation-step"><p class="notation-line␣problem">Result:
             ${numGroups} groups${remainder > 0 ? ` with ${remainder} remaining` : ''
            }</p></div>`;
        outputElement.innerHTML = output;
    }

    function decomposeNumber(num) {
        const result = [];
```

```
        let tempNum = num;
        let placeValue = Math.pow(10, Math.floor(Math.log10(num)));

        while (placeValue >= 1) {
            const digit = Math.floor(tempNum / placeValue);
            if (digit > 0) {
                result.push(digit * placeValue);
            }
            tempNum %= placeValue;
            placeValue /= 10;
        }

        return result;
    }

    function drawCGOBDiagram(svgId, totalItems, groupSize, numGroups, remainder) {
        const svg = document.getElementById(svgId);
        if (!svg) return;

        svg.innerHTML = '';

        const defs = document.createElementNS("http://www.w3.org/2000/svg", 'defs');
        defs.innerHTML = `<marker id="arrowhead-orange" markerWidth="10" markerHeight="7"
                        refX="9" refY="3.5" orient="auto">
                        <polygon points="0␣0,␣10␣3.5,␣0␣7" fill="orange" /></marker>`;
        svg.appendChild(defs);

        const blockUnitSize = totalItems > 100 ? 8 : 10;
        const tenBlockWidth = blockUnitSize;
        const tenBlockHeight = blockUnitSize * 10;
        const hundredBlockSize = blockUnitSize * 10;
        const blockSpacing = 4;
        const groupSpacingX = 20;
        const sectionSpacingY = 150;
        const startX = 30;
        let currentY = 40;

        const colorInitial = 'purple';
        const colorFinal = 'lightgreen';
        const colorRemainder = 'lightblue';
        const colorRegrouped = 'orange';

        const maxBlockHeight = Math.max(tenBlockHeight, hundredBlockSize,
            blockUnitSize);

        createText(svg, startX, currentY, `Initial Total: ${totalItems}`);
        currentY += 30;
        let currentX = startX;
        let section1MaxY = currentY;

        let hundreds = Math.floor(totalItems / 100);
        let tens = Math.floor((totalItems % 100) / 10);
        let ones = totalItems % 10;
```

```
339
340             let initialBlocksData = [];
341
342             for (let i = 0; i < hundreds; i++) {
343                 let info = drawHundredBlock(svg, currentX, currentY, hundredBlockSize,
                        colorInitial, blockUnitSize);
344                 initialBlocksData.push(info);
345                 currentX += hundredBlockSize + groupSpacingX;
346                 section1MaxY = Math.max(section1MaxY, currentY + hundredBlockSize);
347             }
348
349             for (let i = 0; i < tens; i++) {
350                 let info = drawTenBlock(svg, currentX, currentY, tenBlockWidth,
                        tenBlockHeight, colorInitial, blockUnitSize);
351                 initialBlocksData.push(info);
352                 currentX += tenBlockWidth + blockSpacing;
353                 section1MaxY = Math.max(section1MaxY, currentY + tenBlockHeight);
354             }
355
356             for (let i = 0; i < ones; i++) {
357                 let info = drawBlock(svg, currentX, currentY + maxBlockHeight -
                        blockUnitSize, blockUnitSize, blockUnitSize, colorInitial);
358                 initialBlocksData.push(info);
359                 currentX += blockUnitSize + blockSpacing;
360                 section1MaxY = Math.max(section1MaxY, currentY + blockUnitSize);
361             }
362
363             currentY = section1MaxY + sectionSpacingY;
364
365             createText(svg, startX, currentY, `Regrouping into groups of ${groupSize}`);
366             currentY += 30;
367
368             let allRegroupData = visualizeRegrouping(svg, startX, currentY, totalItems,
                    groupSize,
369                                                      blockUnitSize, blockSpacing, groupSpacingX
                                                             ,
370                                                      colorRegrouped, colorRemainder);
371
372             let section2MaxY = allRegroupData.maxY;
373
374             currentY = section2MaxY + sectionSpacingY;
375
376             createText(svg, startX, currentY, `Final Result: ${numGroups} groups of ${
                    groupSize}${remainder > 0 ? ` with ${remainder} remaining` : ''}`);
377             currentY += 30;
378
379             let section3MaxY = drawFinalGroups(svg, startX, currentY, numGroups, groupSize
                    , remainder,
380                                                blockUnitSize, blockSpacing, groupSpacingX,
                                                       colorFinal, colorRemainder);
381
382             if (allRegroupData.groups && allRegroupData.groups.length > 0) {
383                 drawConnectionArrows(svg, allRegroupData.groups, startX, currentY,
```

```
                                numGroups, groupSize, blockUnitSize, blockSpacing,
                                    groupSpacingX);
        }

        let itemsPerRow = Math.min(groupSize, 8);
        let finalGroupWidth = (itemsPerRow * (blockUnitSize + blockSpacing)) -
            blockSpacing + 8;
        let finalGroupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockUnitSize +
            blockSpacing)) - blockSpacing + 8;
        let labelOffset = Math.min(25, finalGroupHeight / 3);

        let svgWidth = Math.max(800, currentX + 100);
        let svgHeight = Math.max(section3MaxY + 50, currentY + finalGroupHeight);

        const maxGroupsPerRow = Math.floor((650 - 50) / (finalGroupWidth +
            groupSpacingX));
        const numFinalRows = Math.ceil(numGroups / maxGroupsPerRow);
        if (numFinalRows > 1) {
            svgHeight += (numFinalRows - 1) * (finalGroupHeight + groupSpacingX +
                labelOffset);
        }

        svg.setAttribute('width', svgWidth);
        svg.setAttribute('height', svgHeight);
        svg.setAttribute('viewBox', `0 0 ${svgWidth} ${svgHeight}`);
    }

    function visualizeRegrouping(svg, startX, startY, totalItems, groupSize,
                                 blockSize, blockSpacing, groupSpacing,
                                 regroupColor, remainderColor) {
        let currentX = startX;
        let currentY = startY;
        let maxY = currentY;

        let hundreds = Math.floor(totalItems / 100);
        let tens = Math.floor((totalItems % 100) / 10);
        let ones = totalItems % 10;

        let allRegroupedGroups = [];
        let allLeftovers = [];

        for (let h = 0; h < hundreds; h++) {
            let groupsPerHundred = Math.floor(100 / groupSize);
            let leftoverPerHundred = 100 % groupSize;

            const hundredSize = 10 * blockSize;

            const hundredOutline = document.createElementNS("http://www.w3.org/2000/svg
                ", 'rect');
            hundredOutline.setAttribute('x', currentX);
            hundredOutline.setAttribute('y', currentY);
            hundredOutline.setAttribute('width', hundredSize);
            hundredOutline.setAttribute('height', hundredSize);
            hundredOutline.setAttribute('fill', 'none');
```

```
            hundredOutline.setAttribute('stroke', 'gray');
            hundredOutline.setAttribute('stroke-dasharray', '4␣4');
            svg.appendChild(hundredOutline);

            const unitsPerRow = 10;
            const unitsPerCol = 10;
            const fullRows = Math.floor(groupSize / unitsPerRow);
            const remainingInLastRow = groupSize % unitsPerRow;

            for (let g = 0; g < groupsPerHundred; g++) {
                let startRow = Math.floor((g * groupSize) / unitsPerRow);
                let startCol = (g * groupSize) % unitsPerRow;

                for (let i = 0; i < groupSize; i++) {
                    let row = startRow + Math.floor((startCol + i) / unitsPerRow);
                    let col = (startCol + i) % unitsPerRow;

                    if (row < unitsPerCol) {
                        const unitRect = document.createElementNS("http://www.w3.org
                            /2000/svg", 'rect');
                        unitRect.setAttribute('x', currentX + col * blockSize);
                        unitRect.setAttribute('y', currentY + row * blockSize);
                        unitRect.setAttribute('width', blockSize);
                        unitRect.setAttribute('height', blockSize);
                        unitRect.setAttribute('fill', regroupColor);
                        unitRect.setAttribute('opacity', '0.7');
                        svg.appendChild(unitRect);
                    }
                }

                let groupStartRow = Math.floor((g * groupSize) / unitsPerRow);
                let groupStartCol = (g * groupSize) % unitsPerRow;
                let groupEndRow = Math.floor(((g+1) * groupSize - 1) / unitsPerRow);
                let groupEndCol = ((g+1) * groupSize - 1) % unitsPerRow;

                if (groupStartRow === groupEndRow) {
                    const groupOutline = document.createElementNS("http://www.w3.org
                        /2000/svg", 'rect');
                    groupOutline.setAttribute('x', currentX + groupStartCol * blockSize
                         - 1);
                    groupOutline.setAttribute('y', currentY + groupStartRow * blockSize
                         - 1);
                    groupOutline.setAttribute('width', (groupEndCol - groupStartCol +
                        1) * blockSize + 2);
                    groupOutline.setAttribute('height', blockSize + 2);
                    groupOutline.setAttribute('fill', 'none');
                    groupOutline.setAttribute('stroke', '#555');
                    groupOutline.setAttribute('stroke-dasharray', '4␣4');
                    svg.appendChild(groupOutline);

                    allRegroupedGroups.push({
                        x: currentX + groupStartCol * blockSize,
                        y: currentY + groupStartRow * blockSize,
                        width: (groupEndCol - groupStartCol + 1) * blockSize,
```

```
                    height: blockSize,
                    cx: currentX + (groupStartCol + (groupEndCol - groupStartCol)/2)
                        * blockSize,
                    cy: currentY + (groupStartRow + 0.5) * blockSize,
                    isBaseGroup: true
                });
            } else {
                const firstRowWidth = (unitsPerRow - groupStartCol) * blockSize;
                const firstRowOutline = document.createElementNS("http://www.w3.org
                    /2000/svg", 'rect');
                firstRowOutline.setAttribute('x', currentX + groupStartCol *
                    blockSize - 1);
                firstRowOutline.setAttribute('y', currentY + groupStartRow *
                    blockSize - 1);
                firstRowOutline.setAttribute('width', firstRowWidth + 2);
                firstRowOutline.setAttribute('height', blockSize + 2);
                firstRowOutline.setAttribute('fill', 'none');
                firstRowOutline.setAttribute('stroke', '#555');
                firstRowOutline.setAttribute('stroke-dasharray', '4␣4');
                svg.appendChild(firstRowOutline);

                for (let r = groupStartRow + 1; r < groupEndRow; r++) {
                    const rowOutline = document.createElementNS("http://www.w3.org
                        /2000/svg", 'rect');
                    rowOutline.setAttribute('x', currentX - 1);
                    rowOutline.setAttribute('y', currentY + r * blockSize - 1);
                    rowOutline.setAttribute('width', unitsPerRow * blockSize + 2);
                    rowOutline.setAttribute('height', blockSize + 2);
                    rowOutline.setAttribute('fill', 'none');
                    rowOutline.setAttribute('stroke', '#555');
                    rowOutline.setAttribute('stroke-dasharray', '4␣4');
                    svg.appendChild(rowOutline);
                }

                const lastRowWidth = (groupEndCol + 1) * blockSize;
                const lastRowOutline = document.createElementNS("http://www.w3.org
                    /2000/svg", 'rect');
                lastRowOutline.setAttribute('x', currentX - 1);
                lastRowOutline.setAttribute('y', currentY + groupEndRow * blockSize
                    - 1);
                lastRowOutline.setAttribute('width', lastRowWidth + 2);
                lastRowOutline.setAttribute('height', blockSize + 2);
                lastRowOutline.setAttribute('fill', 'none');
                lastRowOutline.setAttribute('stroke', '#555');
                lastRowOutline.setAttribute('stroke-dasharray', '4␣4');
                svg.appendChild(lastRowOutline);

                allRegroupedGroups.push({
                    x: currentX,
                    y: currentY + groupStartRow * blockSize,
                    width: hundredSize,
                    height: (groupEndRow - groupStartRow + 1) * blockSize,
                    cx: currentX + hundredSize/2,
```

```
                    cy: currentY + (groupStartRow + (groupEndRow - groupStartRow)/2)
                        * blockSize,
                    isBaseGroup: true
                });
            }
        }

        if (leftoverPerHundred > 0) {
            let leftoverStartRow = Math.floor((groupsPerHundred * groupSize) /
                unitsPerRow);
            let leftoverStartCol = (groupsPerHundred * groupSize) % unitsPerRow;

            for (let i = 0; i < leftoverPerHundred; i++) {
                let row = leftoverStartRow + Math.floor((leftoverStartCol + i) /
                    unitsPerRow);
                let col = (leftoverStartCol + i) % unitsPerRow;

                if (row < unitsPerCol) {
                    const unitRect = document.createElementNS("http://www.w3.org
                        /2000/svg", 'rect');
                    unitRect.setAttribute('x', currentX + col * blockSize);
                    unitRect.setAttribute('y', currentY + row * blockSize);
                    unitRect.setAttribute('width', blockSize);
                    unitRect.setAttribute('height', blockSize);
                    unitRect.setAttribute('fill', remainderColor);
                    svg.appendChild(unitRect);
                }
            }

            let leftoverEndRow = Math.floor(((groupsPerHundred * groupSize) +
                leftoverPerHundred - 1) / unitsPerRow);
            let leftoverEndCol = ((groupsPerHundred * groupSize) +
                leftoverPerHundred - 1) % unitsPerRow;

            if (leftoverStartRow === leftoverEndRow) {
                const leftoverOutline = document.createElementNS("http://www.w3.org
                    /2000/svg", 'rect');
                leftoverOutline.setAttribute('x', currentX + leftoverStartCol *
                    blockSize - 1);
                leftoverOutline.setAttribute('y', currentY + leftoverStartRow *
                    blockSize - 1);
                leftoverOutline.setAttribute('width', (leftoverEndCol -
                    leftoverStartCol + 1) * blockSize + 2);
                leftoverOutline.setAttribute('height', blockSize + 2);
                leftoverOutline.setAttribute('fill', 'none');
                leftoverOutline.setAttribute('stroke', '#555');
                leftoverOutline.setAttribute('stroke-dasharray', '4␣4');
                svg.appendChild(leftoverOutline);

                allLeftovers.push({
                    count: leftoverPerHundred,
                    info: {
                        x: currentX + leftoverStartCol * blockSize,
                        y: currentY + leftoverStartRow * blockSize,
```

```
                            width: (leftoverEndCol - leftoverStartCol + 1) * blockSize,
                            height: blockSize,
                            cx: currentX + (leftoverStartCol + (leftoverEndCol -
                                leftoverStartCol)/2) * blockSize,
                            cy: currentY + (leftoverStartRow + 0.5) * blockSize
                        }
                    });
                } else {
                    allLeftovers.push({
                        count: leftoverPerHundred,
                        info: {
                            x: currentX,
                            y: currentY + leftoverStartRow * blockSize,
                            width: hundredSize,
                            height: (leftoverEndRow - leftoverStartRow + 1) * blockSize,
                            cx: currentX + hundredSize/2,
                            cy: currentY + (leftoverStartRow + (leftoverEndRow -
                                leftoverStartRow)/2) * blockSize
                        }
                    });
                }
            }

            currentX += hundredSize + groupSpacing;
            maxY = Math.max(maxY, currentY + hundredSize);
        }

        for (let t = 0; t < tens; t++) {
            let groupsPerTen = Math.floor(10 / groupSize);
            let leftoverPerTen = 10 % groupSize;

            const tenHeight = 10 * blockSize;
            const tenWidth = blockSize;

            const tenOutline = document.createElementNS("http://www.w3.org/2000/svg", '
                rect');
            tenOutline.setAttribute('x', currentX);
            tenOutline.setAttribute('y', currentY);
            tenOutline.setAttribute('width', tenWidth);
            tenOutline.setAttribute('height', tenHeight);
            tenOutline.setAttribute('fill', 'none');
            tenOutline.setAttribute('stroke', 'gray');
            tenOutline.setAttribute('stroke-dasharray', '4␣4');
            svg.appendChild(tenOutline);

            if (groupsPerTen > 0) {
                for (let g = 0; g < groupsPerTen; g++) {
                    const groupRect = document.createElementNS("http://www.w3.org/2000/
                        svg", 'rect');
                    groupRect.setAttribute('x', currentX);
                    groupRect.setAttribute('y', currentY + g * groupSize * blockSize);
                    groupRect.setAttribute('width', tenWidth);
                    groupRect.setAttribute('height', groupSize * blockSize);
                    groupRect.setAttribute('fill', regroupColor);
```

```
                groupRect.setAttribute('opacity', '0.7');
                svg.appendChild(groupRect);

                const groupOutline = document.createElementNS("http://www.w3.org
                    /2000/svg", 'rect');
                groupOutline.setAttribute('x', currentX - 1);
                groupOutline.setAttribute('y', currentY + g * groupSize * blockSize
                     - 1);
                groupOutline.setAttribute('width', tenWidth + 2);
                groupOutline.setAttribute('height', groupSize * blockSize + 2);
                groupOutline.setAttribute('fill', 'none');
                groupOutline.setAttribute('stroke', '#555');
                groupOutline.setAttribute('stroke-dasharray', '4␣4');
                svg.appendChild(groupOutline);

                allRegroupedGroups.push({
                    x: currentX,
                    y: currentY + g * groupSize * blockSize,
                    width: tenWidth,
                    height: groupSize * blockSize,
                    cx: currentX + tenWidth/2,
                    cy: currentY + (g * groupSize + groupSize/2) * blockSize,
                    isBaseGroup: true
                });
            }
        }

        if (leftoverPerTen > 0) {
            const leftoverY = currentY + groupsPerTen * groupSize * blockSize;

            const leftoverRect = document.createElementNS("http://www.w3.org/2000/
                svg", 'rect');
            leftoverRect.setAttribute('x', currentX);
            leftoverRect.setAttribute('y', leftoverY);
            leftoverRect.setAttribute('width', tenWidth);
            leftoverRect.setAttribute('height', leftoverPerTen * blockSize);
            leftoverRect.setAttribute('fill', remainderColor);
            svg.appendChild(leftoverRect);

            const leftoverOutline = document.createElementNS("http://www.w3.org
                /2000/svg", 'rect');
            leftoverOutline.setAttribute('x', currentX - 1);
            leftoverOutline.setAttribute('y', leftoverY - 1);
            leftoverOutline.setAttribute('width', tenWidth + 2);
            leftoverOutline.setAttribute('height', leftoverPerTen * blockSize + 2);
            leftoverOutline.setAttribute('fill', 'none');
            leftoverOutline.setAttribute('stroke', '#555');
            leftoverOutline.setAttribute('stroke-dasharray', '4␣4');
            svg.appendChild(leftoverOutline);

            allLeftovers.push({
                count: leftoverPerTen,
                info: {
                    x: currentX,
```

```
                        y: leftoverY,
                        width: tenWidth,
                        height: leftoverPerTen * blockSize,
                        cx: currentX + tenWidth/2,
                        cy: leftoverY + leftoverPerTen * blockSize/2
                    }
                });
            }

            currentX += tenWidth + blockSpacing;
            maxY = Math.max(maxY, currentY + tenHeight);
        }

        if (ones > 0) {
            let onesStartX = currentX;

            for (let i = 0; i < ones; i++) {
                let oneBlock = drawBlock(svg, currentX, currentY, blockSize,
                    remainderColor);

                allLeftovers.push({
                    count: 1,
                    info: oneBlock
                });

                currentX += blockSize + blockSpacing;
            }

            if (ones > 1) {
                const onesOutline = document.createElementNS("http://www.w3.org/2000/
                    svg", 'rect');
                onesOutline.setAttribute('x', onesStartX - 1);
                onesOutline.setAttribute('y', currentY - 1);
                onesOutline.setAttribute('width', ones * (blockSize + blockSpacing) -
                    blockSpacing + 2);
                onesOutline.setAttribute('height', blockSize + 2);
                onesOutline.setAttribute('fill', 'none');
                onesOutline.setAttribute('stroke', '#555');
                onesOutline.setAttribute('stroke-dasharray', '4␣4');
                svg.appendChild(onesOutline);
            }

            maxY = Math.max(maxY, currentY + blockSize);
        }

        if (allLeftovers.length > 0) {
            let totalLeftover = allLeftovers.reduce((sum, item) => sum + item.count, 0)
                ;

            if (totalLeftover >= groupSize) {
                let additionalGroups = Math.floor(totalLeftover / groupSize);
                let finalRemainder = totalLeftover % groupSize;

                currentY = maxY + 40;
```

```
                createText(svg, startX, currentY, `Combined Leftovers: ${totalLeftover}
                    items`);
                currentY += 30;
                currentX = startX;

                let combinedGroupsInfo = [];

                for (let g = 0; g < additionalGroups; g++) {
                    let groupInfo = drawRegroupBlock(svg, currentX, currentY, groupSize
                        , blockSize,
                                                     blockSpacing, regroupColor, true);

                    createText(svg, currentX + groupInfo.width/2, currentY - 15,
                            `Group ${g+1} from Leftovers`, 'diagram-label', 'middle');

                    combinedGroupsInfo.push(groupInfo);
                    currentX += groupInfo.width + groupSpacing * 1.5;
                }

                if (finalRemainder > 0) {
                    let remainderInfo = drawRegroupBlock(svg, currentX, currentY,
                        finalRemainder,
                                                         blockSize, blockSpacing,
                                                             remainderColor, false);

                    createText(svg, currentX + remainderInfo.width/2, currentY - 15,
                            `Final Remainder`, 'diagram-label', 'middle');

                    maxY = Math.max(maxY, currentY + remainderInfo.height);
                } else {
                    maxY = Math.max(maxY, currentY + (Math.ceil(groupSize/8) * (
                        blockSize + blockSpacing)));
                }

                let targetX = startX + (additionalGroups * groupSize * blockSize / 4);
                let targetY = currentY - 25;

                for (let leftover of allLeftovers) {
                    let source = leftover.info;

                    createCurvedArrow(svg, source.cx, source.cy,
                                      targetX, targetY,
                                      (source.cx + targetX)/2, (source.cy + targetY)/2 -
                                          20);
                }

                allRegroupedGroups.push(...combinedGroupsInfo);
            }
        }

        return {
            maxY: maxY,
            groups: allRegroupedGroups,
            leftovers: allLeftovers
```

```
769             };
770         }
771 
772         function drawRegroupBlock(svg, x, y, count, blockSize, blockSpacing, color,
                addOutline = true) {
773             let itemsPerRow = Math.min(count, 8);
774             let rows = Math.ceil(count / itemsPerRow);
775 
776             let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing;
777             let groupHeight = (rows * (blockSize + blockSpacing)) - blockSpacing;
778 
779             for (let i = 0; i < count; i++) {
780                 let row = Math.floor(i / itemsPerRow);
781                 let col = i % itemsPerRow;
782 
783                 drawBlock(svg,
784                         x + col * (blockSize + blockSpacing),
785                         y + row * (blockSize + blockSpacing),
786                         blockSize, color);
787             }
788 
789             if (addOutline) {
790                 const outline = document.createElementNS("http://www.w3.org/2000/svg", '
                        rect');
791                 outline.setAttribute('x', x - 2);
792                 outline.setAttribute('y', y - 2);
793                 outline.setAttribute('width', groupWidth + 4);
794                 outline.setAttribute('height', groupHeight + 4);
795                 outline.setAttribute('fill', 'none');
796                 outline.setAttribute('stroke', '#555');
797                 outline.setAttribute('stroke-dasharray', '4␣4');
798                 svg.appendChild(outline);
799             }
800 
801             return {
802                 x: x,
803                 y: y,
804                 width: groupWidth,
805                 height: groupHeight,
806                 cx: x + groupWidth/2,
807                 cy: y + groupHeight/2
808             };
809         }
810 
811         function drawConnectionArrows(svg, sourceGroups, targetStartX, targetStartY,
812                                       numGroups, groupSize, blockSize, blockSpacing,
                                          groupSpacing) {
813             const combinedGroups = sourceGroups.filter(group => !group.isBaseGroup);
814             if (combinedGroups.length === 0) return;
815 
816             const baseGroupCount = sourceGroups.filter(group => group.isBaseGroup).length;
817 
818             let itemsPerRow = Math.min(groupSize, 8);
```

```
819             let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing +
                    8;
820             let groupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockSize +
                    blockSpacing)) - blockSpacing + 8;
821             let labelOffset = Math.min(25, groupHeight / 3);
822
823             const svgContainerWidth = 650;
824             const maxGroupsPerRow = Math.max(1, Math.floor((svgContainerWidth - 50) / (
                    groupWidth + groupSpacing)));
825
826             for (let i = 0; i < combinedGroups.length; i++) {
827                 let source = combinedGroups[i];
828
829                 const targetGroupIndex = baseGroupCount + i;
830
831                 const targetRow = Math.floor(targetGroupIndex / maxGroupsPerRow);
832                 const targetCol = targetGroupIndex % maxGroupsPerRow;
833
834                 let targetX = targetStartX + (targetCol * (groupWidth + groupSpacing)) +
                        groupWidth/2;
835                 let arrowEndY = targetStartY + (targetRow * (groupHeight + groupSpacing +
                        labelOffset));
836
837                 let controlY = (source.cy + arrowEndY)/2;
838                 if (arrowEndY < source.cy) {
839                     controlY = arrowEndY + (source.cy - arrowEndY)/2;
840                 }
841
842                 createCurvedArrow(
843                     svg,
844                     source.cx, source.cy + source.height/2 + 5,
845                     targetX, arrowEndY + 10,
846                     source.cx, controlY
847                 );
848             }
849         }
850
851         function drawFinalGroups(svg, startX, startY, numGroups, groupSize, remainder,
852                                  blockSize, blockSpacing, groupSpacing, groupColor,
                                         remainderColor) {
853             let maxY = startY;
854
855             let itemsPerRow = Math.min(groupSize, 8);
856             let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing +
                    8;
857             let groupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockSize +
                    blockSpacing)) - blockSpacing + 8;
858
859             const svgContainerWidth = 650;
860             const maxGroupsPerRow = Math.max(1, Math.floor((svgContainerWidth - 60) / (
                    groupWidth + groupSpacing)));
861             const labelOffset = Math.min(25, groupHeight / 3);
862
863             const numRows = Math.ceil(numGroups / maxGroupsPerRow);
```

```
        for (let row = 0; row < numRows; row++) {
            let currentY = startY + row * (groupHeight + groupSpacing + labelOffset);
            let currentX = startX;

            const startGroup = row * maxGroupsPerRow;
            const endGroup = Math.min(numGroups, (row + 1) * maxGroupsPerRow);

            for (let g = startGroup; g < endGroup; g++) {
                let groupStartX = currentX;

                const groupRect = document.createElementNS("http://www.w3.org/2000/svg"
                    , 'rect');
                groupRect.setAttribute('x', groupStartX - 4);
                groupRect.setAttribute('y', currentY - 4);
                groupRect.setAttribute('width', groupWidth);
                groupRect.setAttribute('height', groupHeight);
                groupRect.setAttribute('class', 'final-group');
                svg.appendChild(groupRect);

                createText(svg, groupStartX + groupWidth/2, currentY - labelOffset/2,
                        `Group ${g+1}`, 'diagram-label', 'middle');

                for (let r = 0; r < Math.ceil(groupSize / itemsPerRow); r++) {
                    for (let c = 0; c < itemsPerRow; c++) {
                        const index = r * itemsPerRow + c;
                        if (index < groupSize) {
                            drawBlock(svg,
                                    groupStartX + c * (blockSize + blockSpacing),
                                    currentY + r * (blockSize + blockSpacing),
                                    blockSize, groupColor, 'final-group-block');
                        }
                    }
                }

                currentX += groupWidth + groupSpacing;
                maxY = Math.max(maxY, currentY + groupHeight);
            }
        }

        if (remainder > 0) {
            let remainderY, remainderX;

            if (numRows === 1 && numGroups * (groupWidth + groupSpacing) + remainder *
                (blockSize + blockSpacing) < svgContainerWidth - 60) {
                remainderY = startY;
                remainderX = startX + numGroups * (groupWidth + groupSpacing);
            } else {
                const lastRowGroups = numGroups % maxGroupsPerRow || maxGroupsPerRow;
                const remainderWidth = remainder * (blockSize + blockSpacing);

                if (lastRowGroups * (groupWidth + groupSpacing) + remainderWidth <
                    svgContainerWidth - 60) {
```

```
                    remainderY = startY + (numRows - 1) * (groupHeight + groupSpacing +
                        labelOffset);
                    remainderX = startX + lastRowGroups * (groupWidth + groupSpacing);
                } else {
                    remainderY = startY + numRows * (groupHeight + groupSpacing +
                        labelOffset);
                    remainderX = startX;
                }
            }

            createText(svg, remainderX + (remainder * (blockSize + blockSpacing))/2,
                remainderY - labelOffset/2,
                    `Remainder: ${remainder}`, 'diagram-label', 'middle');

            for (let r = 0; r < remainder; r++) {
                drawBlock(svg,
                        remainderX + r * (blockSize + blockSpacing),
                        remainderY,
                        blockSize, remainderColor, 'remainder-block');
            }

            maxY = Math.max(maxY, remainderY + blockSize);
        }

        return maxY;
    }

    runCGOBAutomaton();
});
</script>

</body>
</html>
```

References

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