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

Compiled by: Theodore M. Savich

April 1, 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!
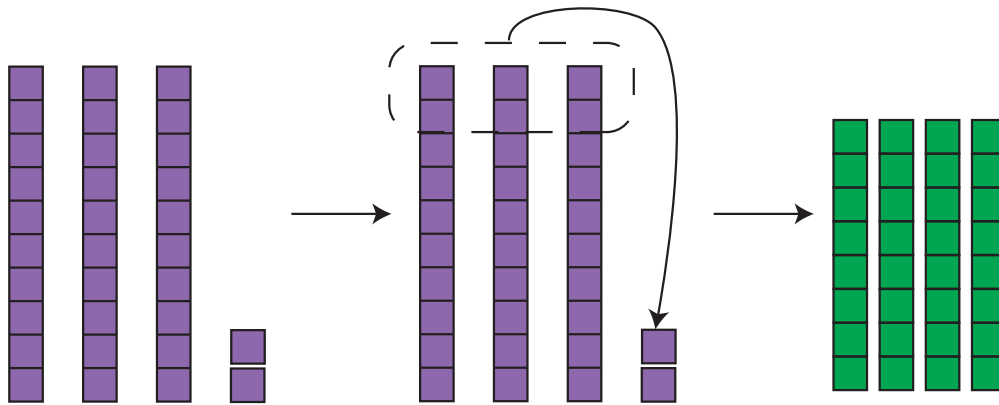
$$32 = 3 \times 10 + 2$$
$$= 3 \times 8 + 3 \times 2 + 2$$
$$= 3 \times 8 + 8$$
$$= 4 \times 8$$
$$32 \div 8 = 4$$

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
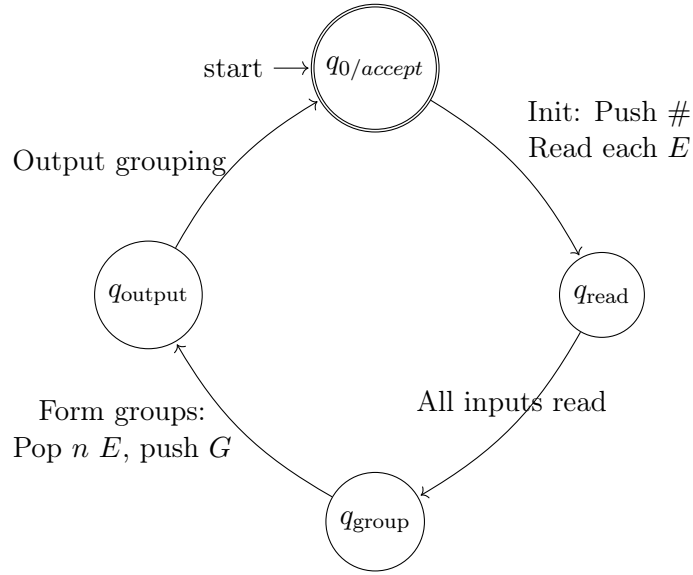
Figure 1: PDA for Conversion to Groups Other than Bases

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

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Division: Conversion to Groups Other than Bases (CGOB)</title>
5       <style>
6           body { font-family: sans-serif; }
7           #cgobDiagram { border: 1px solid #d3d3d3; min-height: 500px; width: 100%; }
8           #outputContainer { margin-top: 20px; }
9           .diagram-label { font-size: 14px; display: block; margin-bottom: 10px; font-weight
                : bold;}
10          .notation-line { margin: 0.2em 0; margin-left: 1em; font-family: monospace;}
11          .notation-line.problem { font-weight: bold; margin-left: 0;}
12          .notation-step { margin-bottom: 0.5em; }
13          /* Block Styles */
14          .block { stroke: black; stroke-width: 0.5; }
15          .ten-block-bg { stroke: black; stroke-width: 1; }
16          .hundred-block-bg { stroke: black; stroke-width: 1; }
17          .unit-block-inner { stroke: lightgrey; stroke-width: 0.5; }
18          .initial-total-block { fill: purple; } /* Color for initial total */
19          .final-group-block { fill: lightgreen; } /* Color for final groups */
20          .regrouped-block { fill: orange; opacity: 0.7; }
21          .remainder-block { fill: lightblue; }
22          .regroup-arrow {
23              fill: none; stroke: orange; stroke-width: 1.5;
24              marker-end: url(#arrowhead-orange);
25          }
26          .regroup-grouping {
27              fill: none; stroke: #555; stroke-width: 1;
28              stroke-dasharray: 4 4;
29          }
30          .section-container {
31              margin-bottom: 20px;
32              padding: 10px;
33              border-radius: 5px;
34              background-color: #f9f9f9;
35          }
36          .final-group {
37              stroke: blue;
38              stroke-width: 1;
39              fill: none;
40              stroke-dasharray: 5 3;
41          }
42      </style>
43  </head>
44  <body>
45
46  <h1>Strategic Multiplicative Reasoning: Division - Conversion to Groups Other than Bases
        (CGOB)</h1>
47
48  <div>
49      <label for="cgobTotal">Total Items (Dividend):</label>
50      <input type="number" id="cgobTotal" value="32" min="1">
```

5

```html
</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
                ');
```

```
100        unitBlock.setAttribute('x', x); unitBlock.setAttribute('y', y + i *
               unitBlockSize);
101        unitBlock.setAttribute('width', unitBlockSize); unitBlock.setAttribute('height
               ', unitBlockSize);
102        unitBlock.setAttribute('fill', fill);
103        unitBlock.setAttribute('class', 'unit-block-inner');
104        group.appendChild(unitBlock);
105      }
106      svg.appendChild(group);
107      return { x, y, width, height, type: 't', cx: x + width/2, cy: y + height/2};
108    }
109
110    function drawHundredBlock(svg, x, y, size, fill, unitBlockSize) {
111      const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
112      const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", '
               rect');
113      backgroundRect.setAttribute('x', x); backgroundRect.setAttribute('y', y);
114      backgroundRect.setAttribute('width', size); backgroundRect.setAttribute('height',
               size);
115      backgroundRect.setAttribute('fill', fill);
116      backgroundRect.setAttribute('class', 'hundred-block-bg␣block');
117      group.appendChild(backgroundRect);
118
119      for (let row = 0; row < 10; row++) {
120        for (let col = 0; col < 10; col++) {
121          const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
                   rect');
122          unitBlock.setAttribute('x', x + col * unitBlockSize);
123          unitBlock.setAttribute('y', y + row * unitBlockSize);
124          unitBlock.setAttribute('width', unitBlockSize);
125          unitBlock.setAttribute('height', unitBlockSize);
126          unitBlock.setAttribute('fill', fill);
127          unitBlock.setAttribute('class', 'unit-block-inner');
128          group.appendChild(unitBlock);
129        }
130      }
131      svg.appendChild(group);
132      return { x, y, width: size, height: size, type: 'h', cx: x + size/2, cy: y + size
               /2};
133    }
134
135    function createText(svg, x, y, textContent, className = 'diagram-label', anchor = '
             start') {
136      const uniqueId = 'text-' + Math.random().toString(36).substr(2, 9);
137
138      const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
139      text.setAttribute('x', x);
140      text.setAttribute('y', y);
141      text.setAttribute('class', className);
142      text.setAttribute('text-anchor', anchor);
143      text.setAttribute('id', uniqueId);
144      text.textContent = textContent;
145
146      if (className === 'diagram-label') {
```

```
147          const background = document.createElementNS("http://www.w3.org/2000/svg", '
                 rect');
148          const padding = 3;
149          const estimatedWidth = Math.max(7 * textContent.length, 30);
150          const estimatedHeight = 16;
151
152          let bgX = x - padding;
153          if (anchor === 'middle') {
154              bgX = x - (estimatedWidth / 2) - padding;
155          } else if (anchor === 'end') {
156              bgX = x - estimatedWidth - padding;
157          }
158
159          background.setAttribute('x', bgX);
160          background.setAttribute('y', y - estimatedHeight + padding);
161          background.setAttribute('width', estimatedWidth + (padding * 2));
162          background.setAttribute('height', estimatedHeight + padding);
163          background.setAttribute('fill', 'white');
164          background.setAttribute('fill-opacity', '0.9');
165          background.setAttribute('rx', '3');
166          svg.appendChild(background);
167      }
168
169      svg.appendChild(text);
170      return uniqueId;
171  }
172
173  function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy, arrowClass='regroup-arrow',
          headId='arrowhead-orange') {
174      const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
175      path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}`);
176      path.setAttribute('class', arrowClass);
177      path.setAttribute('marker-end', `url(#${headId})`);
178      svg.appendChild(path);
179  }
180
181  document.addEventListener('DOMContentLoaded', function() {
182      const outputElement = document.getElementById('cgobOutput');
183      const totalInput = document.getElementById('cgobTotal');
184      const groupSizeInput = document.getElementById('cgobGroupSize');
185      const diagramSVG = document.getElementById('cgobDiagram');
186
187      if (!outputElement || !totalInput || !groupSizeInput || !diagramSVG) {
188          console.error("Required HTML elements not found!");
189          return;
190      }
191
192      window.runCGOBAutomaton = function() {
193          try {
194              const totalItems = parseInt(totalInput.value);
195              const groupSize = parseInt(groupSizeInput.value);
196
197              if (isNaN(totalItems) || isNaN(groupSize) || totalItems <= 0 || groupSize
                     <= 0) {
```

```
198                    outputElement.textContent = "Please␣enter␣valid␣positive␣numbers";
199                    diagramSVG.innerHTML = ''; return;
200                }
201
202                const numGroups = Math.floor(totalItems / groupSize);
203                const remainder = totalItems % groupSize;
204
205                generateCGOBNotation(outputElement, totalItems, groupSize, numGroups,
                       remainder);
206                drawCGOBDiagram('cgobDiagram', totalItems, groupSize, numGroups, remainder)
                       ;
207
208            } catch (error) {
209                console.error("Error␣in␣runCGOBAutomaton:", error);
210                outputElement.textContent = `Error: ${error.message}`;
211            }
212        };
213
214        function generateCGOBNotation(outputElement, totalItems, groupSize, numGroups,
               remainder) {
215            let output = `<h2>Conversion to Groups Other than Bases (CGOB) - Notation</h2
                   >`;
216            output += `<div class="notation-step"><p class="notation-line␣problem">${
                   totalItems}  ${groupSize} = ?</p></div>`;
217
218            const placeValues = decomposeNumber(totalItems);
219            output += `<div class="notation-step"><p class="notation-line">Start with ${
                   totalItems} = ${placeValues.join('␣+␣')}</p></div>`;
220
221            let steps = [];
222            let remainders = [];
223            let regroupedItems = 0;
224            let runningTotal = totalItems;
225
226            let completeGroups = 0;
227
228            for (let i = 0; i < placeValues.length; i++) {
229                const placeValue = parseInt(placeValues[i]);
230                if (placeValue === 0) continue;
231
232                const base = Math.pow(10, placeValues.length - i - 1);
233                const count = placeValue / base;
234
235                if (base > 1) {
236                    const wholeGroups = Math.floor(base / groupSize);
237                    const leftover = base % groupSize;
238
239                    steps.push(`${count}  ${base} = ${count}  (${wholeGroups}  ${groupSize}
                           + ${leftover})`);
240                    steps.push(`= ${count * wholeGroups}  ${groupSize} + ${count}  ${
                           leftover}`);
241
242                    const newGroups = count * wholeGroups;
243                    completeGroups += newGroups;
```

```
244            regroupedItems += newGroups * groupSize;
245
246            if (i > 0 || count * leftover > 0) {
247                steps.push('= ${completeGroups}  ${groupSize} + ${count * leftover}
                        ${remainders.length > 0 ? '␣+␣' + remainders.join('␣+␣') : ''} =
                        ${totalItems}');
248            } else {
249                steps.push('= ${completeGroups}  ${groupSize} = ${regroupedItems}')
                        ;
250            }
251
252            if (leftover > 0) {
253                remainders.push(count * leftover);
254            }
255        } else {
256            remainders.push(placeValue);
257        }
258    }
259
260    if (remainders.length > 0) {
261        const totalRemainder = remainders.reduce((sum, val) => sum + val, 0);
262        steps.push('Combined leftovers: ${remainders.join('␣+␣')} = ${
                totalRemainder}');
263
264        const additionalGroups = Math.floor(totalRemainder / groupSize);
265        const finalRemainder = totalRemainder % groupSize;
266
267        if (additionalGroups > 0) {
268            steps.push('Leftovers form ${additionalGroups} more group${
                    additionalGroups > 1 ? 's' : ''} of ${groupSize}${finalRemainder >
                    0 ? ' with ${finalRemainder} remaining' : ''}');
269
270            completeGroups += additionalGroups;
271            steps.push('= ${completeGroups}  ${groupSize}${finalRemainder > 0 ? ' +
                    ${finalRemainder}' : ''} = ${totalItems}');
272        } else if (totalRemainder > 0) {
273            steps.push('= ${completeGroups}  ${groupSize} + ${totalRemainder} = ${
                    totalItems}');
274        }
275    }
276
277    steps.forEach(step => {
278        output += '<div class="notation-step"><p class="notation-line">${step}</p
                ></div>';
279    });
280
281    output += '<div class="notation-step"><p class="notation-line␣problem">Result:
                ${numGroups} groups${remainder > 0 ? ' with ${remainder} remaining' : ''
            }</p></div>';
282    outputElement.innerHTML = output;
283 }
284
285 function decomposeNumber(num) {
286    const result = [];
```

10

```
287            let tempNum = num;
288            let placeValue = Math.pow(10, Math.floor(Math.log10(num)));
289
290            while (placeValue >= 1) {
291                const digit = Math.floor(tempNum / placeValue);
292                if (digit > 0) {
293                    result.push(digit * placeValue);
294                }
295                tempNum %= placeValue;
296                placeValue /= 10;
297            }
298
299            return result;
300        }
301
302        function drawCGOBDiagram(svgId, totalItems, groupSize, numGroups, remainder) {
303            const svg = document.getElementById(svgId);
304            if (!svg) return;
305
306            svg.innerHTML = '';
307
308            const defs = document.createElementNS("http://www.w3.org/2000/svg", 'defs');
309            defs.innerHTML = '<marker id="arrowhead-orange" markerWidth="10" markerHeight=
                    "7"
310                            refX="9" refY="3.5" orient="auto">
311                            <polygon points="0 0, 10 3.5, 0 7" fill="orange" /></marker>';
312            svg.appendChild(defs);
313
314            const blockUnitSize = totalItems > 100 ? 8 : 10;
315            const tenBlockWidth = blockUnitSize;
316            const tenBlockHeight = blockUnitSize * 10;
317            const hundredBlockSize = blockUnitSize * 10;
318            const blockSpacing = 4;
319            const groupSpacingX = 20;
320            const sectionSpacingY = 150;
321            const startX = 30;
322            let currentY = 40;
323
324            const colorInitial = 'purple';
325            const colorFinal = 'lightgreen';
326            const colorRemainder = 'lightblue';
327            const colorRegrouped = 'orange';
328
329            const maxBlockHeight = Math.max(tenBlockHeight, hundredBlockSize,
                    blockUnitSize);
330
331            createText(svg, startX, currentY, 'Initial Total: ${totalItems}');
332            currentY += 30;
333            let currentX = startX;
334            let section1MaxY = currentY;
335
336            let hundreds = Math.floor(totalItems / 100);
337            let tens = Math.floor((totalItems % 100) / 10);
338            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, 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,
```

```
384                                numGroups, groupSize, blockUnitSize, blockSpacing,
                                   groupSpacingX);
385            }
386
387        let itemsPerRow = Math.min(groupSize, 8);
388        let finalGroupWidth = (itemsPerRow * (blockUnitSize + blockSpacing)) -
               blockSpacing + 8;
389        let finalGroupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockUnitSize +
               blockSpacing)) - blockSpacing + 8;
390        let labelOffset = Math.min(25, finalGroupHeight / 3);
391
392        let svgWidth = Math.max(800, currentX + 100);
393        let svgHeight = Math.max(section3MaxY + 50, currentY + finalGroupHeight);
394
395        const maxGroupsPerRow = Math.floor((650 - 50) / (finalGroupWidth +
               groupSpacingX));
396        const numFinalRows = Math.ceil(numGroups / maxGroupsPerRow);
397        if (numFinalRows > 1) {
398            svgHeight += (numFinalRows - 1) * (finalGroupHeight + groupSpacingX +
                   labelOffset);
399        }
400
401        svg.setAttribute('width', svgWidth);
402        svg.setAttribute('height', svgHeight);
403        svg.setAttribute('viewBox', `0 0 ${svgWidth} ${svgHeight}`);
404    }
405
406    function visualizeRegrouping(svg, startX, startY, totalItems, groupSize,
407                                blockSize, blockSpacing, groupSpacing,
408                                regroupColor, remainderColor) {
409        let currentX = startX;
410        let currentY = startY;
411        let maxY = currentY;
412
413        let hundreds = Math.floor(totalItems / 100);
414        let tens = Math.floor((totalItems % 100) / 10);
415        let ones = totalItems % 10;
416
417        let allRegroupedGroups = [];
418        let allLeftovers = [];
419
420        for (let h = 0; h < hundreds; h++) {
421            let groupsPerHundred = Math.floor(100 / groupSize);
422            let leftoverPerHundred = 100 % groupSize;
423
424            const hundredSize = 10 * blockSize;
425
426            const hundredOutline = document.createElementNS("http://www.w3.org/2000/svg
                   ", 'rect');
427            hundredOutline.setAttribute('x', currentX);
428            hundredOutline.setAttribute('y', currentY);
429            hundredOutline.setAttribute('width', hundredSize);
430            hundredOutline.setAttribute('height', hundredSize);
431            hundredOutline.setAttribute('fill', 'none');
```

```
432                    hundredOutline.setAttribute('stroke', 'gray');
433                    hundredOutline.setAttribute('stroke-dasharray', '4␣4');
434                    svg.appendChild(hundredOutline);
435
436                    const unitsPerRow = 10;
437                    const unitsPerCol = 10;
438                    const fullRows = Math.floor(groupSize / unitsPerRow);
439                    const remainingInLastRow = groupSize % unitsPerRow;
440
441                    for (let g = 0; g < groupsPerHundred; g++) {
442                        let startRow = Math.floor((g * groupSize) / unitsPerRow);
443                        let startCol = (g * groupSize) % unitsPerRow;
444
445                        for (let i = 0; i < groupSize; i++) {
446                            let row = startRow + Math.floor((startCol + i) / unitsPerRow);
447                            let col = (startCol + i) % unitsPerRow;
448
449                            if (row < unitsPerCol) {
450                                const unitRect = document.createElementNS("http://www.w3.org
                                        /2000/svg", 'rect');
451                                unitRect.setAttribute('x', currentX + col * blockSize);
452                                unitRect.setAttribute('y', currentY + row * blockSize);
453                                unitRect.setAttribute('width', blockSize);
454                                unitRect.setAttribute('height', blockSize);
455                                unitRect.setAttribute('fill', regroupColor);
456                                unitRect.setAttribute('opacity', '0.7');
457                                svg.appendChild(unitRect);
458                            }
459                        }
460
461                        let groupStartRow = Math.floor((g * groupSize) / unitsPerRow);
462                        let groupStartCol = (g * groupSize) % unitsPerRow;
463                        let groupEndRow = Math.floor(((g+1) * groupSize - 1) / unitsPerRow);
464                        let groupEndCol = ((g+1) * groupSize - 1) % unitsPerRow;
465
466                        if (groupStartRow === groupEndRow) {
467                            const groupOutline = document.createElementNS("http://www.w3.org
                                    /2000/svg", 'rect');
468                            groupOutline.setAttribute('x', currentX + groupStartCol * blockSize
                                    - 1);
469                            groupOutline.setAttribute('y', currentY + groupStartRow * blockSize
                                    - 1);
470                            groupOutline.setAttribute('width', (groupEndCol - groupStartCol +
                                    1) * blockSize + 2);
471                            groupOutline.setAttribute('height', blockSize + 2);
472                            groupOutline.setAttribute('fill', 'none');
473                            groupOutline.setAttribute('stroke', '#555');
474                            groupOutline.setAttribute('stroke-dasharray', '4␣4');
475                            svg.appendChild(groupOutline);
476
477                            allRegroupedGroups.push({
478                                x: currentX + groupStartCol * blockSize,
479                                y: currentY + groupStartRow * blockSize,
480                                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,
```

```
527                          cy: currentY + (groupStartRow + (groupEndRow - groupStartRow)/2)
                                    * blockSize,
528                          isBaseGroup: true
529                      });
530                  }
531              }
532
533          if (leftoverPerHundred > 0) {
534              let leftoverStartRow = Math.floor((groupsPerHundred * groupSize) /
                        unitsPerRow);
535              let leftoverStartCol = (groupsPerHundred * groupSize) % unitsPerRow;
536
537              for (let i = 0; i < leftoverPerHundred; i++) {
538                  let row = leftoverStartRow + Math.floor((leftoverStartCol + i) /
                            unitsPerRow);
539                  let col = (leftoverStartCol + i) % unitsPerRow;
540
541                  if (row < unitsPerCol) {
542                      const unitRect = document.createElementNS("http://www.w3.org
                                /2000/svg", 'rect');
543                      unitRect.setAttribute('x', currentX + col * blockSize);
544                      unitRect.setAttribute('y', currentY + row * blockSize);
545                      unitRect.setAttribute('width', blockSize);
546                      unitRect.setAttribute('height', blockSize);
547                      unitRect.setAttribute('fill', remainderColor);
548                      svg.appendChild(unitRect);
549                  }
550              }
551
552              let leftoverEndRow = Math.floor(((groupsPerHundred * groupSize) +
                        leftoverPerHundred - 1) / unitsPerRow);
553              let leftoverEndCol = ((groupsPerHundred * groupSize) +
                        leftoverPerHundred - 1) % unitsPerRow;
554
555              if (leftoverStartRow === leftoverEndRow) {
556                  const leftoverOutline = document.createElementNS("http://www.w3.org
                            /2000/svg", 'rect');
557                  leftoverOutline.setAttribute('x', currentX + leftoverStartCol *
                            blockSize - 1);
558                  leftoverOutline.setAttribute('y', currentY + leftoverStartRow *
                            blockSize - 1);
559                  leftoverOutline.setAttribute('width', (leftoverEndCol -
                            leftoverStartCol + 1) * blockSize + 2);
560                  leftoverOutline.setAttribute('height', blockSize + 2);
561                  leftoverOutline.setAttribute('fill', 'none');
562                  leftoverOutline.setAttribute('stroke', '#555');
563                  leftoverOutline.setAttribute('stroke-dasharray', '4␣4');
564                  svg.appendChild(leftoverOutline);
565
566                  allLeftovers.push({
567                      count: leftoverPerHundred,
568                      info: {
569                          x: currentX + leftoverStartCol * blockSize,
570                          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);
```

```
621                    groupRect.setAttribute('opacity', '0.7');
622                    svg.appendChild(groupRect);
623
624                    const groupOutline = document.createElementNS("http://www.w3.org
                           /2000/svg", 'rect');
625                    groupOutline.setAttribute('x', currentX - 1);
626                    groupOutline.setAttribute('y', currentY + g * groupSize * blockSize
                           - 1);
627                    groupOutline.setAttribute('width', tenWidth + 2);
628                    groupOutline.setAttribute('height', groupSize * blockSize + 2);
629                    groupOutline.setAttribute('fill', 'none');
630                    groupOutline.setAttribute('stroke', '#555');
631                    groupOutline.setAttribute('stroke-dasharray', '4␣4');
632                    svg.appendChild(groupOutline);
633
634                    allRegroupedGroups.push({
635                        x: currentX,
636                        y: currentY + g * groupSize * blockSize,
637                        width: tenWidth,
638                        height: groupSize * blockSize,
639                        cx: currentX + tenWidth/2,
640                        cy: currentY + (g * groupSize + groupSize/2) * blockSize,
641                        isBaseGroup: true
642                    });
643                }
644            }
645
646            if (leftoverPerTen > 0) {
647                const leftoverY = currentY + groupsPerTen * groupSize * blockSize;
648
649                const leftoverRect = document.createElementNS("http://www.w3.org/2000/
                       svg", 'rect');
650                leftoverRect.setAttribute('x', currentX);
651                leftoverRect.setAttribute('y', leftoverY);
652                leftoverRect.setAttribute('width', tenWidth);
653                leftoverRect.setAttribute('height', leftoverPerTen * blockSize);
654                leftoverRect.setAttribute('fill', remainderColor);
655                svg.appendChild(leftoverRect);
656
657                const leftoverOutline = document.createElementNS("http://www.w3.org
                       /2000/svg", 'rect');
658                leftoverOutline.setAttribute('x', currentX - 1);
659                leftoverOutline.setAttribute('y', leftoverY - 1);
660                leftoverOutline.setAttribute('width', tenWidth + 2);
661                leftoverOutline.setAttribute('height', leftoverPerTen * blockSize + 2);
662                leftoverOutline.setAttribute('fill', 'none');
663                leftoverOutline.setAttribute('stroke', '#555');
664                leftoverOutline.setAttribute('stroke-dasharray', '4␣4');
665                svg.appendChild(leftoverOutline);
666
667                allLeftovers.push({
668                    count: leftoverPerTen,
669                    info: {
670                        x: currentX,
```

18

```
                             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;
```

19

```
721              createText(svg, startX, currentY, `Combined Leftovers: ${totalLeftover}
                        items`);
722              currentY += 30;
723              currentX = startX;
724
725              let combinedGroupsInfo = [];
726
727              for (let g = 0; g < additionalGroups; g++) {
728                  let groupInfo = drawRegroupBlock(svg, currentX, currentY, groupSize
                            , blockSize,
729                                          blockSpacing, regroupColor, true);
730
731                  createText(svg, currentX + groupInfo.width/2, currentY - 15,
732                          `Group ${g+1} from Leftovers`, 'diagram-label', 'middle');
733
734                  combinedGroupsInfo.push(groupInfo);
735                  currentX += groupInfo.width + groupSpacing * 1.5;
736              }
737
738              if (finalRemainder > 0) {
739                  let remainderInfo = drawRegroupBlock(svg, currentX, currentY,
                            finalRemainder,
740                                          blockSize, blockSpacing,
                                              remainderColor, false);
741
742                  createText(svg, currentX + remainderInfo.width/2, currentY - 15,
743                          `Final Remainder`, 'diagram-label', 'middle');
744
745                  maxY = Math.max(maxY, currentY + remainderInfo.height);
746              } else {
747                  maxY = Math.max(maxY, currentY + (Math.ceil(groupSize/8) * (
                            blockSize + blockSpacing)));
748              }
749
750              let targetX = startX + (additionalGroups * groupSize * blockSize / 4);
751              let targetY = currentY - 25;
752
753              for (let leftover of allLeftovers) {
754                  let source = leftover.info;
755
756                  createCurvedArrow(svg, source.cx, source.cy,
757                              targetX, targetY,
758                              (source.cx + targetX)/2, (source.cy + targetY)/2 -
                                  20);
759              }
760
761              allRegroupedGroups.push(...combinedGroupsInfo);
762          }
763      }
764
765      return {
766          maxY: maxY,
767          groups: allRegroupedGroups,
768          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) {
```

```
914                    remainderY = startY + (numRows - 1) * (groupHeight + groupSpacing +
                           labelOffset);
915                    remainderX = startX + lastRowGroups * (groupWidth + groupSpacing);
916                } else {
917                    remainderY = startY + numRows * (groupHeight + groupSpacing +
                           labelOffset);
918                    remainderX = startX;
919                }
920            }
921
922            createText(svg, remainderX + (remainder * (blockSize + blockSpacing))/2,
                   remainderY - labelOffset/2,
923                    `Remainder: ${remainder}`, 'diagram-label', 'middle');
924
925            for (let r = 0; r < remainder; r++) {
926                drawBlock(svg,
927                        remainderX + r * (blockSize + blockSpacing),
928                        remainderY,
929                        blockSize, remainderColor, 'remainder-block');
930            }
931
932            maxY = Math.max(maxY, remainderY + blockSize);
933        }
934
935        return maxY;
936    }
937
938    runCGOBAutomaton();
939    });
940 </script>
941
942 </body>
943 </html>
```

References

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