

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

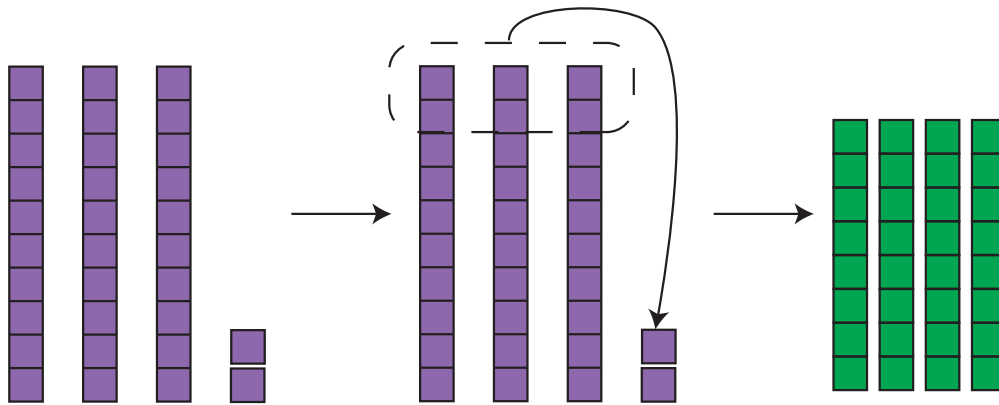
Compiled by: Theodore M. Savich

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:** $\Gamma = \{\#, G, E_1, E_2, \dots\}$, 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

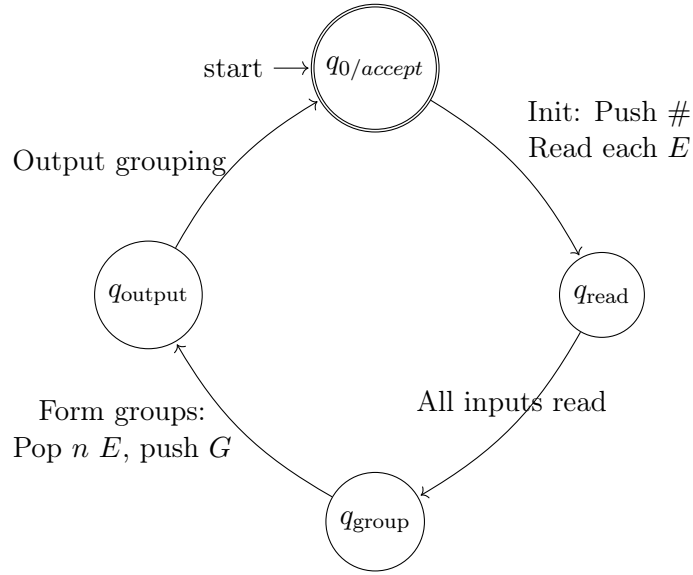


Figure 1: PDA for Conversion to Groups Other than Bases

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

```

51 </div>
52 <div>
53   <label for="cgobGroupSize">Items per Group (Divisor):</label>
54   <input type="number" id="cgobGroupSize" value="8" min="1">
55 </div>
56
57 <button onclick="runCGOBAutomaton()">Calculate and Visualize</button>
58
59 <div id="outputContainer">
60   <h2>Explanation (Notation):</h2>
61   <div id="cgobOutput">
62     <!-- Text output will be displayed here -->
63   </div>
64 </div>
65
66 <h2>Diagram:</h2>
67 <div style="overflow-x:auto;overflow-y:auto;max-height:800px;">
68   <svg id="cgobDiagram" preserveAspectRatio="xMinYMin meet">
69     <defs>
70       <marker id="arrowhead-orange" markerWidth="10" markerHeight="7" refX="9" refY=
71         "3.5" orient="auto">
72         <polygone points="0,0,10,3.5,0,7" fill="orange" />
73       </marker>
74     </defs>
75   </svg>
76 </div>
77 <script>
78   // --- Helper SVG Functions ---
79   function drawBlock(svg, x, y, size, fill, className = 'block') {
80     const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
81     rect.setAttribute('x', x); rect.setAttribute('y', y);
82     rect.setAttribute('width', size); rect.setAttribute('height', size);
83     rect.setAttribute('fill', fill);
84     rect.setAttribute('class', className);
85     svg.appendChild(rect);
86     return { x, y, width: size, height: size, type: 'o', cx: x + size/2, cy: y + size
87       /2 };
88   }
89   function drawTenBlock(svg, x, y, width, height, fill, unitBlockSize) {
90     const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
91     const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", '
92       rect');
93     backgroundRect.setAttribute('x', x); backgroundRect.setAttribute('y', y);
94     backgroundRect.setAttribute('width', width); backgroundRect.setAttribute('height',
95       height);
96     backgroundRect.setAttribute('fill', fill);
97     backgroundRect.setAttribute('class', 'ten-block-bg_block');
98     group.appendChild(backgroundRect);
99     for (let i = 0; i < 10; i++) {
100       const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", 'rect
101         ');

```

```

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", '
148         rect');
149     const padding = 3;
150     const estimatedWidth = Math.max(7 * textContent.length, 30);
151     const estimatedHeight = 16;
152
153     let bgX = x - padding;
154     if (anchor === 'middle') {
155         bgX = x - (estimatedWidth / 2) - padding;
156     } else if (anchor === 'end') {
157         bgX = x - estimatedWidth - padding;
158     }
159
160     background.setAttribute('x', bgX);
161     background.setAttribute('y', y - estimatedHeight + padding);
162     background.setAttribute('width', estimatedWidth + (padding * 2));
163     background.setAttribute('height', estimatedHeight + padding);
164     background.setAttribute('fill', 'white');
165     background.setAttribute('fill-opacity', '0.9');
166     background.setAttribute('rx', '3');
167     svg.appendChild(background);
168
169     svg.appendChild(text);
170     return uniqueId;
171 }
172
173 function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy, arrowClass='regroup-arrow',
174     headId='arrowhead-orange') {
175     const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
176     path.setAttribute('d', 'M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}');
177     path.setAttribute('class', arrowClass);
178     path.setAttribute('marker-end', 'url(#${headId})');
179     svg.appendChild(path);
180 }
181
182 document.addEventListener('DOMContentLoaded', function() {
183     const outputElement = document.getElementById('cgobOutput');
184     const totalInput = document.getElementById('cgobTotal');
185     const groupSizeInput = document.getElementById('cgobGroupSize');
186     const diagramSVG = document.getElementById('cgobDiagram');
187
188     if (!outputElement || !totalInput || !groupSizeInput || !diagramSVG) {
189         console.error("Required HTML elements not found!");
190         return;
191     }
192
193     window.runCGOBAutomaton = function() {
194         try {
195             const totalItems = parseInt(totalInput.value);
196             const groupSize = parseInt(groupSizeInput.value);
197
198             if (isNaN(totalItems) || isNaN(groupSize) || totalItems <= 0 || groupSize
199                 <= 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,
206         remainder);
207     drawCGBDiagram('cgobDiagram', totalItems, groupSize, numGroups, remainder)
208         ;
209
210     } catch (error) {
211         console.error("Error in runCGBAutomaton:", error);
212         outputElement.textContent = 'Error: ${error.message}';
213     }
214
215     };
216
217     function generateCGBNotation(outputElement, totalItems, groupSize, numGroups,
218         remainder) {
219         let output = '<h2>Conversion to Groups Other than Bases (CGB) - Notation</h2>';
220
221         output += '<div class="notation-step"><p class="notation-line_problem">${
222             totalItems}  ${groupSize} = ?</p></div>';
223
224         const placeValues = decomposeNumber(totalItems);
225         output += '<div class="notation-step"><p class="notation-line">Start with ${
226             totalItems} = ${placeValues.join(' + ')}</p></div>';
227
228         let steps = [];
229         let remainders = [];
230         let regroupedItems = 0;
231         let runningTotal = totalItems;
232
233         let completeGroups = 0;
234
235         for (let i = 0; i < placeValues.length; i++) {
236             const placeValue = parseInt(placeValues[i]);
237             if (placeValue === 0) continue;
238
239             const base = Math.pow(10, placeValues.length - i - 1);
240             const count = placeValue / base;
241
242             if (base > 1) {
243                 const wholeGroups = Math.floor(base / groupSize);
244                 const leftover = base % groupSize;
245
246                 steps.push(`${count}  ${base} = ${count}  (${wholeGroups}  ${groupSize}
247                     + ${leftover})`);
248                 steps.push(`= ${count * wholeGroups}  ${groupSize} + ${count}  ${
249                     leftover}`);
250
251                 const newGroups = count * wholeGroups;
252                 completeGroups += newGroups;

```

```

244         regroupedItems += newGroups * groupSize;
245
246         if (i > 0 || count * leftover > 0) {
247             steps.push(' = ${completeGroups} ${groupSize} + ${count * leftover}
248                 ${remainders.length > 0 ? '␣' + remainders.join('␣') : ''} =
249                 ${totalItems}');
250         } else {
251             steps.push(' = ${completeGroups} ${groupSize} = ${regroupedItems}');
252         };
253
254         if (leftover > 0) {
255             remainders.push(count * leftover);
256         }
257     } else {
258         remainders.push(placeValue);
259     }
260
261     if (remainders.length > 0) {
262         const totalRemainder = remainders.reduce((sum, val) => sum + val, 0);
263         steps.push('Combined leftovers: ${remainders.join('␣')} = ${
264             totalRemainder}');
265
266         const additionalGroups = Math.floor(totalRemainder / groupSize);
267         const finalRemainder = totalRemainder % groupSize;
268
269         if (additionalGroups > 0) {
270             steps.push('Leftovers form ${additionalGroups} more group${
271                 additionalGroups > 1 ? 's' : ''} of ${groupSize}${finalRemainder >
272                 0 ? ' with ${finalRemainder} remaining' : ''}');
273
274             completeGroups += additionalGroups;
275             steps.push(' = ${completeGroups} ${groupSize}${finalRemainder > 0 ? ' +
276                 ${finalRemainder}' : ''} = ${totalItems}');
277         } else if (totalRemainder > 0) {
278             steps.push(' = ${completeGroups} ${groupSize} + ${totalRemainder} = ${
279                 totalItems}');
280         }
281     }
282
283     steps.forEach(step => {
284         output += '<div class="notation-step"><p class="notation-line">${step}</p
285             ></div>';
286     });
287
288     output += '<div class="notation-step"><p class="notation-line␣problem">Result:
289         ${numGroups} groups${remainder > 0 ? ' with ${remainder} remaining' : ''}
290     </p></div>';
291     outputElement.innerHTML = output;
292 }
293
294 function decomposeNumber(num) {
295     const result = [];

```

```

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                     <polygone 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   let initialBlocksData = [];
340
341
342   for (let i = 0; i < hundreds; i++) {
343     let info = drawHundredBlock(svg, currentX, currentY, hundredBlockSize,
344       colorInitial, blockUnitSize);
345     initialBlocksData.push(info);
346     currentX += hundredBlockSize + groupSpacingX;
347     section1MaxY = Math.max(section1MaxY, currentY + hundredBlockSize);
348   }
349
350   for (let i = 0; i < tens; i++) {
351     let info = drawTenBlock(svg, currentX, currentY, tenBlockWidth,
352       tenBlockHeight, colorInitial, blockUnitSize);
353     initialBlocksData.push(info);
354     currentX += tenBlockWidth + blockSpacing;
355     section1MaxY = Math.max(section1MaxY, currentY + tenBlockHeight);
356   }
357
358   for (let i = 0; i < ones; i++) {
359     let info = drawBlock(svg, currentX, currentY + maxBlockHeight -
360       blockUnitSize, blockUnitSize, colorInitial);
361     initialBlocksData.push(info);
362     currentX += blockUnitSize + blockSpacing;
363     section1MaxY = Math.max(section1MaxY, currentY + blockUnitSize);
364   }
365
366   currentY = section1MaxY + sectionSpacingY;
367
368   createText(svg, startX, currentY, 'Regrouping into groups of ${groupSize}');
369   currentY += 30;
370
371   let allRegroupData = visualizeRegrouping(svg, startX, currentY, totalItems,
372     groupSize,
373     blockUnitSize, blockSpacing, groupSpacingX
374     ,
375     colorRegrouped, colorRemainder);
376
377   let section2MaxY = allRegroupData.maxY;
378
379   currentY = section2MaxY + sectionSpacingY;
380
381   createText(svg, startX, currentY, 'Final Result: ${numGroups} groups of ${
382     groupSize}${remainder > 0 ? ' with ${remainder} remaining' : ''}');
383   currentY += 30;
384
385   let section3MaxY = drawFinalGroups(svg, startX, currentY, numGroups, groupSize
386     , remainder,
387     blockUnitSize, blockSpacing, groupSpacingX,
388     colorFinal, colorRemainder);
389
390   if (allRegroupData.groups && allRegroupData.groups.length > 0) {
391     drawConnectionArrows(svg, allRegroupData.groups, startX, currentY,

```

```

384         numGroups, groupSize, blockUnitSize, blockSpacing,
385         groupSpacingX);
386     }
387     let itemsPerRow = Math.min(groupSize, 8);
388     let finalGroupWidth = (itemsPerRow * (blockUnitSize + blockSpacing)) -
389         blockSpacing + 8;
390     let finalGroupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockUnitSize +
391         blockSpacing)) - blockSpacing + 8;
392     let labelOffset = Math.min(25, finalGroupHeight / 3);
393
394     let svgWidth = Math.max(800, currentX + 100);
395     let svgHeight = Math.max(section3MaxY + 50, currentY + finalGroupHeight);
396
397     const maxGroupsPerRow = Math.floor((650 - 50) / (finalGroupWidth +
398         groupSpacingX));
399     const numFinalRows = Math.ceil(numGroups / maxGroupsPerRow);
400     if (numFinalRows > 1) {
401         svgHeight += (numFinalRows - 1) * (finalGroupHeight + groupSpacingX +
402             labelOffset);
403     }
404     svg.setAttribute('width', svgWidth);
405     svg.setAttribute('height', svgHeight);
406     svg.setAttribute('viewBox', '0 0 ${svgWidth} ${svgHeight}');
407 }
408
409 function visualizeRegrouping(svg, startX, startY, totalItems, groupSize,
410     blockSize, blockSpacing, groupSpacing,
411     regroupColor, remainderColor) {
412     let currentX = startX;
413     let currentY = startY;
414     let maxY = currentY;
415
416     let hundreds = Math.floor(totalItems / 100);
417     let tens = Math.floor((totalItems % 100) / 10);
418     let ones = totalItems % 10;
419
420     let allRegroupedGroups = [];
421     let allLeftovers = [];
422
423     for (let h = 0; h < hundreds; h++) {
424         let groupsPerHundred = Math.floor(100 / groupSize);
425         let leftoverPerHundred = 100 % groupSize;
426
427         const hundredSize = 10 * blockSize;
428
429         const hundredOutline = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
430         hundredOutline.setAttribute('x', currentX);
431         hundredOutline.setAttribute('y', currentY);
432         hundredOutline.setAttribute('width', hundredSize);
433         hundredOutline.setAttribute('height', hundredSize);
434         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
451         /2000/svg", 'rect');
452       unitRect.setAttribute('x', currentX + col * blockSize);
453       unitRect.setAttribute('y', currentY + row * blockSize);
454       unitRect.setAttribute('width', blockSize);
455       unitRect.setAttribute('height', blockSize);
456       unitRect.setAttribute('fill', regroupColor);
457       unitRect.setAttribute('opacity', '0.7');
458       svg.appendChild(unitRect);
459     }
460   }
461
462   let groupStartRow = Math.floor((g * groupSize) / unitsPerRow);
463   let groupStartCol = (g * groupSize) % unitsPerRow;
464   let groupEndRow = Math.floor((g+1) * groupSize - 1) / unitsPerRow);
465   let groupEndCol = ((g+1) * groupSize - 1) % unitsPerRow;
466
467   if (groupStartRow === groupEndRow) {
468     const groupOutline = document.createElementNS("http://www.w3.org
469       /2000/svg", 'rect');
470     groupOutline.setAttribute('x', currentX + groupStartCol * blockSize
471       - 1);
472     groupOutline.setAttribute('y', currentY + groupStartRow * blockSize
473       - 1);
474     groupOutline.setAttribute('width', (groupEndCol - groupStartCol +
475       1) * blockSize + 2);
476     groupOutline.setAttribute('height', blockSize + 2);
477     groupOutline.setAttribute('fill', 'none');
478     groupOutline.setAttribute('stroke', '#555');
479     groupOutline.setAttribute('stroke-dasharray', '4_4');
480     svg.appendChild(groupOutline);
481
482     allRegroupedGroups.push({
483       x: currentX + groupStartCol * blockSize,
484       y: currentY + groupStartRow * blockSize,
485       width: (groupEndCol - groupStartCol + 1) * blockSize,

```

```

481         height: blockSize,
482         cx: currentX + (groupStartCol + (groupEndCol - groupStartCol)/2)
           * blockSize,
483         cy: currentY + (groupStartRow + 0.5) * blockSize,
484         isBaseGroup: true
485     });
486 } else {
487     const firstRowWidth = (unitsPerRow - groupStartCol) * blockSize;
488     const firstRowOutline = document.createElementNS("http://www.w3.org
           /2000/svg", 'rect');
489     firstRowOutline.setAttribute('x', currentX + groupStartCol *
           blockSize - 1);
490     firstRowOutline.setAttribute('y', currentY + groupStartRow *
           blockSize - 1);
491     firstRowOutline.setAttribute('width', firstRowWidth + 2);
492     firstRowOutline.setAttribute('height', blockSize + 2);
493     firstRowOutline.setAttribute('fill', 'none');
494     firstRowOutline.setAttribute('stroke', '#555');
495     firstRowOutline.setAttribute('stroke-dasharray', '4_4');
496     svg.appendChild(firstRowOutline);
497
498     for (let r = groupStartRow + 1; r < groupEndRow; r++) {
499         const rowOutline = document.createElementNS("http://www.w3.org
           /2000/svg", 'rect');
500         rowOutline.setAttribute('x', currentX - 1);
501         rowOutline.setAttribute('y', currentY + r * blockSize - 1);
502         rowOutline.setAttribute('width', unitsPerRow * blockSize + 2);
503         rowOutline.setAttribute('height', blockSize + 2);
504         rowOutline.setAttribute('fill', 'none');
505         rowOutline.setAttribute('stroke', '#555');
506         rowOutline.setAttribute('stroke-dasharray', '4_4');
507         svg.appendChild(rowOutline);
508     }
509
510     const lastRowWidth = (groupEndCol + 1) * blockSize;
511     const lastRowOutline = document.createElementNS("http://www.w3.org
           /2000/svg", 'rect');
512     lastRowOutline.setAttribute('x', currentX - 1);
513     lastRowOutline.setAttribute('y', currentY + groupEndRow * blockSize
           - 1);
514     lastRowOutline.setAttribute('width', lastRowWidth + 2);
515     lastRowOutline.setAttribute('height', blockSize + 2);
516     lastRowOutline.setAttribute('fill', 'none');
517     lastRowOutline.setAttribute('stroke', '#555');
518     lastRowOutline.setAttribute('stroke-dasharray', '4_4');
519     svg.appendChild(lastRowOutline);
520
521     allRegroupedGroups.push({
522         x: currentX,
523         y: currentY + groupStartRow * blockSize,
524         width: hundredSize,
525         height: (groupEndRow - groupStartRow + 1) * blockSize,
526         cx: currentX + hundredSize/2,

```

```

527         cy: currentY + (groupStartRow + (groupEndRow - groupStartRow)/2)
528             * blockSize,
529         isBaseGroup: true
530     });
531 }
532
533 if (leftoverPerHundred > 0) {
534     let leftoverStartRow = Math.floor((groupsPerHundred * groupSize) /
535         unitsPerRow);
536     let leftoverStartCol = (groupsPerHundred * groupSize) % unitsPerRow;
537
538     for (let i = 0; i < leftoverPerHundred; i++) {
539         let row = leftoverStartRow + Math.floor((leftoverStartCol + i) /
540             unitsPerRow);
541         let col = (leftoverStartCol + i) % unitsPerRow;
542
543         if (row < unitsPerCol) {
544             const unitRect = document.createElementNS("http://www.w3.org
545                 /2000/svg", 'rect');
546             unitRect.setAttribute('x', currentX + col * blockSize);
547             unitRect.setAttribute('y', currentY + row * blockSize);
548             unitRect.setAttribute('width', blockSize);
549             unitRect.setAttribute('height', blockSize);
550             unitRect.setAttribute('fill', remainderColor);
551             svg.appendChild(unitRect);
552         }
553     }
554
555     let leftoverEndRow = Math.floor(((groupsPerHundred * groupSize) +
556         leftoverPerHundred - 1) / unitsPerRow);
557     let leftoverEndCol = ((groupsPerHundred * groupSize) +
558         leftoverPerHundred - 1) % unitsPerRow;
559
560     if (leftoverStartRow === leftoverEndRow) {
561         const leftoverOutline = document.createElementNS("http://www.w3.org
562             /2000/svg", 'rect');
563         leftoverOutline.setAttribute('x', currentX + leftoverStartCol *
564             blockSize - 1);
565         leftoverOutline.setAttribute('y', currentY + leftoverStartRow *
566             blockSize - 1);
567         leftoverOutline.setAttribute('width', (leftoverEndCol -
568             leftoverStartCol + 1) * blockSize + 2);
569         leftoverOutline.setAttribute('height', blockSize + 2);
570         leftoverOutline.setAttribute('fill', 'none');
571         leftoverOutline.setAttribute('stroke', '#555');
572         leftoverOutline.setAttribute('stroke-dasharray', '4 4');
573         svg.appendChild(leftoverOutline);
574
575         allLeftovers.push({
576             count: leftoverPerHundred,
577             info: {
578                 x: currentX + leftoverStartCol * blockSize,
579                 y: currentY + leftoverStartRow * blockSize,

```



```

571         width: (leftoverEndCol - leftoverStartCol + 1) * blockSize,
572         height: blockSize,
573         cx: currentX + (leftoverStartCol + (leftoverEndCol -
                    leftoverStartCol)/2) * blockSize,
574         cy: currentY + (leftoverStartRow + 0.5) * blockSize
575     }
576 });
577 } else {
578     allLeftovers.push({
579         count: leftoverPerHundred,
580         info: {
581             x: currentX,
582             y: currentY + leftoverStartRow * blockSize,
583             width: hundredSize,
584             height: (leftoverEndRow - leftoverStartRow + 1) * blockSize,
585             cx: currentX + hundredSize/2,
586             cy: currentY + (leftoverStartRow + (leftoverEndRow -
                    leftoverStartRow)/2) * blockSize
587         }
588     });
589 }
590 }
591
592 currentX += hundredSize + groupSpacing;
593 maxY = Math.max(maxY, currentY + hundredSize);
594 }
595
596 for (let t = 0; t < tens; t++) {
597     let groupsPerTen = Math.floor(10 / groupSize);
598     let leftoverPerTen = 10 % groupSize;
599
600     const tenHeight = 10 * blockSize;
601     const tenWidth = blockSize;
602
603     const tenOutline = document.createElementNS("http://www.w3.org/2000/svg", '
        rect');
604     tenOutline.setAttribute('x', currentX);
605     tenOutline.setAttribute('y', currentY);
606     tenOutline.setAttribute('width', tenWidth);
607     tenOutline.setAttribute('height', tenHeight);
608     tenOutline.setAttribute('fill', 'none');
609     tenOutline.setAttribute('stroke', 'gray');
610     tenOutline.setAttribute('stroke-dasharray', '4 4');
611     svg.appendChild(tenOutline);
612
613     if (groupsPerTen > 0) {
614         for (let g = 0; g < groupsPerTen; g++) {
615             const groupRect = document.createElementNS("http://www.w3.org/2000/
                svg", 'rect');
616             groupRect.setAttribute('x', currentX);
617             groupRect.setAttribute('y', currentY + g * groupSize * blockSize);
618             groupRect.setAttribute('width', tenWidth);
619             groupRect.setAttribute('height', groupSize * blockSize);
620             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,

```

```

671         y: leftoverY,
672         width: tenWidth,
673         height: leftoverPerTen * blockSize,
674         cx: currentX + tenWidth/2,
675         cy: leftoverY + leftoverPerTen * blockSize/2
676     }
677 });
678 }
679
680 currentX += tenWidth + blockSpacing;
681 maxY = Math.max(maxY, currentY + tenHeight);
682 }
683
684 if (ones > 0) {
685     let onesStartX = currentX;
686
687     for (let i = 0; i < ones; i++) {
688         let oneBlock = drawBlock(svg, currentX, currentY, blockSize,
689             remainderColor);
690
691         allLeftovers.push({
692             count: 1,
693             info: oneBlock
694         });
695
696         currentX += blockSize + blockSpacing;
697     }
698
699     if (ones > 1) {
700         const onesOutline = document.createElementNS("http://www.w3.org/2000/
701             svg", 'rect');
702         onesOutline.setAttribute('x', onesStartX - 1);
703         onesOutline.setAttribute('y', currentY - 1);
704         onesOutline.setAttribute('width', ones * (blockSize + blockSpacing) -
705             blockSpacing + 2);
706         onesOutline.setAttribute('height', blockSize + 2);
707         onesOutline.setAttribute('fill', 'none');
708         onesOutline.setAttribute('stroke', '#555');
709         onesOutline.setAttribute('stroke-dasharray', '4 4');
710         svg.appendChild(onesOutline);
711     }
712
713     maxY = Math.max(maxY, currentY + blockSize);
714 }
715
716 if (allLeftovers.length > 0) {
717     let totalLeftover = allLeftovers.reduce((sum, item) => sum + item.count, 0)
718     ;
719
720     if (totalLeftover >= groupSize) {
721         let additionalGroups = Math.floor(totalLeftover / groupSize);
722         let finalRemainder = totalLeftover % groupSize;
723
724         currentY = maxY + 40;

```

```

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                                     blockSize, 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, blockSize,
              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 + blockSize)));
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,
773     addOutline = true) {
774     let itemsPerRow = Math.min(count, 8);
775     let rows = Math.ceil(count / itemsPerRow);
776
777     let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing;
778     let groupHeight = (rows * (blockSize + blockSpacing)) - blockSpacing;
779
780     for (let i = 0; i < count; i++) {
781         let row = Math.floor(i / itemsPerRow);
782         let col = i % itemsPerRow;
783
784         drawBlock(svg,
785             x + col * (blockSize + blockSpacing),
786             y + row * (blockSize + blockSpacing),
787             blockSize, color);
788     }
789
790     if (addOutline) {
791         const outline = document.createElementNS("http://www.w3.org/2000/svg", '
792             rect');
793         outline.setAttribute('x', x - 2);
794         outline.setAttribute('y', y - 2);
795         outline.setAttribute('width', groupWidth + 4);
796         outline.setAttribute('height', groupHeight + 4);
797         outline.setAttribute('fill', 'none');
798         outline.setAttribute('stroke', '#555');
799         outline.setAttribute('stroke-dasharray', '4 4');
800         svg.appendChild(outline);
801     }
802
803     return {
804         x: x,
805         y: y,
806         width: groupWidth,
807         height: groupHeight,
808         cx: x + groupWidth/2,
809         cy: y + groupHeight/2
810     };
811 }
812
813 function drawConnectionArrows(svg, sourceGroups, targetStartX, targetStartY,
814     numGroups, groupSize, blockSize, blockSpacing,
815     groupSpacing) {
816     const combinedGroups = sourceGroups.filter(group => !group.isBaseGroup);
817     if (combinedGroups.length === 0) return;
818
819     const baseGroupCount = sourceGroups.filter(group => group.isBaseGroup).length;
820
821     let itemsPerRow = Math.min(groupSize, 8);

```

```

819     let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing +
820         8;
821     let groupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockSize +
822         blockSpacing)) - blockSpacing + 8;
823     let labelOffset = Math.min(25, groupHeight / 3);
824
825     const svgContainerWidth = 650;
826     const maxGroupsPerRow = Math.max(1, Math.floor((svgContainerWidth - 50) / (
827         groupWidth + groupSpacing)));
828
829     for (let i = 0; i < combinedGroups.length; i++) {
830         let source = combinedGroups[i];
831
832         const targetGroupIndex = baseGroupCount + i;
833
834         const targetRow = Math.floor(targetGroupIndex / maxGroupsPerRow);
835         const targetCol = targetGroupIndex % maxGroupsPerRow;
836
837         let targetX = targetStartX + (targetCol * (groupWidth + groupSpacing)) +
838             groupWidth/2;
839         let arrowEndY = targetStartY + (targetRow * (groupHeight + groupSpacing +
840             labelOffset));
841
842         let controlY = (source.cy + arrowEndY)/2;
843         if (arrowEndY < source.cy) {
844             controlY = arrowEndY + (source.cy - arrowEndY)/2;
845         }
846
847         createCurvedArrow(
848             svg,
849             source.cx, source.cy + source.height/2 + 5,
850             targetX, arrowEndY + 10,
851             source.cx, controlY
852         );
853     }
854 }
855
856 function drawFinalGroups(svg, startX, startY, numGroups, groupSize, remainder,
857     blockSize, blockSpacing, groupSpacing, groupColor,
858     remainderColor) {
859     let maxY = startY;
860
861     let itemsPerRow = Math.min(groupSize, 8);
862     let groupWidth = (itemsPerRow * (blockSize + blockSpacing)) - blockSpacing +
863         8;
864     let groupHeight = (Math.ceil(groupSize / itemsPerRow) * (blockSize +
865         blockSpacing)) - blockSpacing + 8;
866
867     const svgContainerWidth = 650;
868     const maxGroupsPerRow = Math.max(1, Math.floor((svgContainerWidth - 60) / (
869         groupWidth + groupSpacing)));
870     const labelOffset = Math.min(25, groupHeight / 3);
871
872     const numRows = Math.ceil(numGroups / maxGroupsPerRow);

```

```

864
865 for (let row = 0; row < numRows; row++) {
866     let currentY = startY + row * (groupHeight + groupSpacing + labelOffset);
867     let currentX = startX;
868
869     const startGroup = row * maxGroupsPerRow;
870     const endGroup = Math.min(numGroups, (row + 1) * maxGroupsPerRow);
871
872     for (let g = startGroup; g < endGroup; g++) {
873         let groupStartX = currentX;
874
875         const groupRect = document.createElementNS("http://www.w3.org/2000/svg"
            , 'rect');
876         groupRect.setAttribute('x', groupStartX - 4);
877         groupRect.setAttribute('y', currentY - 4);
878         groupRect.setAttribute('width', groupWidth);
879         groupRect.setAttribute('height', groupHeight);
880         groupRect.setAttribute('class', 'final-group');
881         svg.appendChild(groupRect);
882
883         createText(svg, groupStartX + groupWidth/2, currentY - labelOffset/2,
884             'Group ${g+1}', 'diagram-label', 'middle');
885
886         for (let r = 0; r < Math.ceil(groupSize / itemsPerRow); r++) {
887             for (let c = 0; c < itemsPerRow; c++) {
888                 const index = r * itemsPerRow + c;
889                 if (index < groupSize) {
890                     drawBlock(svg,
891                         groupStartX + c * (blockSize + blockSpacing),
892                         currentY + r * (blockSize + blockSpacing),
893                         blockSize, groupColor, 'final-group-block');
894                 }
895             }
896         }
897
898         currentX += groupWidth + groupSpacing;
899         maxY = Math.max(maxY, currentY + groupHeight);
900     }
901 }
902
903 if (remainder > 0) {
904     let remainderY, remainderX;
905
906     if (numRows === 1 && numGroups * (groupWidth + groupSpacing) + remainder *
        (blockSize + blockSpacing) < svgContainerWidth - 60) {
907         remainderY = startY;
908         remainderX = startX + numGroups * (groupWidth + groupSpacing);
909     } else {
910         const lastRowGroups = numGroups % maxGroupsPerRow || maxGroupsPerRow;
911         const remainderWidth = remainder * (blockSize + blockSpacing);
912
913         if (lastRowGroups * (groupWidth + groupSpacing) + remainderWidth <
            svgContainerWidth - 60) {

```

```

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

```

References

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