Code Documentation: Quadrilateral_Substitution

UMEDCTA Repository

December 3, 2025

Contents

1 Quadrilateral_Substitution/brandom_lesson.js

```
// brandom_lesson.js

document.addEventListener('DOMContentLoaded', () => {
    // --- Module Navigation ---
    const modules = document.querySelectorAll('.module');
    const prevButton = document.getElementById('prevButton');
    const nextButton = document.getElementById('nextButton');
    const moduleIndicator = document.getElementById('moduleIndicator');
    let currentModuleIndex = 0;
    let moduleInitializers = {}; // Store functions to initialize modules

    // --- Helper function for SVG (assuming it's used elsewhere too, place outside initializers) ---
    function getShapeSvg(shapeName) {
       let svg = `<svg viewBox="0 0 100 100" width="70" height="70" class="shape-viz">`;
       svg += `<title>${shapeName}</title>`; // Tooltip for SVG (shows on hover)
       switch (shapeName) {
            case "Square": svg += `<rect x="10" y="10" width="80" height="80" fill="#cfe2ff"
            ↪  stroke="#0d6efd" stroke-width="2"/>`; break;
            case "Rectangle": svg += `<rect x="10" y="20" width="80" height="60" fill="#d1e7dd"
            ↪  stroke="#198754" stroke-width="2"/>`; break;
            case "Rhombus": svg += `<polygon points="50,5 95,50 50,95 5,50" fill="#f8d7da"
            ↪  stroke="#dc3545" stroke-width="2"/>`; break;
            case "Parallelogram": svg += `<polygon points="25,5 95,5 75,95 5,95" fill="#fff3cd"
            ↪  stroke="#ffc107" stroke-width="2"/>`; break;
            case "Kite": svg += `<polygon points="50,10 85,50 50,90 15,50" fill="#f3d7f8"
            ↪  stroke="#a30cff" stroke-width="2"/>`; break;
            case "Trapezoid": svg += `<polygon points="30,10 70,10 100,90 0,90" fill="#e2e3e5"
            ↪  stroke="#6c757d" stroke-width="2"/>`; break;
            case "Quadrilateral": default: svg += `<polygon points="10,10 90,20 80,80 20,90"
            ↪  fill="#f8f9fa" stroke="#adb5bd" stroke-width="2"/>`; break;
       }
       svg += `</svg>`;
       return svg;
    }

    function showModule(index) {
        const totalModules = 7; // **** UPDATE THIS ****
        if (index < 0 || index >= totalModules) return;

        modules.forEach((module, i) => {
            module.style.display = 'none';
            module.classList.remove('current-module', 'hidden-module');
            if (i === index) {
                module.style.display = 'block';
                module.classList.add('current-module');
            } else {
                module.classList.add('hidden-module');
            }
        });

        currentModuleIndex = index;
        moduleIndicator.textContent = `Module ${index + 1} of ${totalModules}`; // **** UPDATE THIS ****
        prevButton.disabled = index === 0;
        nextButton.disabled = index === totalModules - 1; // **** UPDATE THIS ****

        if (typeof moduleInitializers[index] === 'function') {
            try {
                moduleInitializers[index]();
                moduleInitializers[index] = null; // Mark as run
            } catch (error) {
                console.error(`Error initializing module ${index + 1}:`, error);
            }
```

```
        }

        // Scroll to top
        const targetModule = modules[index];
        if (targetModule) {
            const headerOffset = document.querySelector('header')?.offsetHeight || 60;
            const elementPosition = targetModule.getBoundingClientRect().top;
            const offsetPosition = elementPosition + window.pageYOffset - headerOffset - 20;
            window.scrollTo({ top: offsetPosition, behavior: 'smooth' });
         }
    }

    prevButton.addEventListener('click', () => showModule(currentModuleIndex - 1));
    nextButton.addEventListener('click', () => showModule(currentModuleIndex + 1));

    // --- Data Structures ---
    const shapeHierarchy = {
        "Square": { superclasses: ["Rectangle", "Rhombus"], properties: ["4 equal sides", "4 right angles", "opposite sides parallel", "diagonals bisect perpendicularly", "diagonals are equal"], incompatibilities: ["has obtuse angle", "unequal adjacent sides"] },
        "Rectangle": { superclasses: ["Parallelogram", /*"Isosceles Trapezoid" - Removed for simpler linear hierarchy in slider */], properties: ["4 right angles", "opposite sides parallel", "opposite sides equal", "diagonals are equal", "diagonals bisect each other"], incompatibilities: ["has acute angle (internal)", "unequal diagonals"] },
        "Rhombus": { superclasses: ["Parallelogram", "Kite"], properties: ["4 equal sides", "opposite sides parallel", "opposite angles equal", "diagonals bisect perpendicularly"], incompatibilities: ["has right angle but unequal adjacent sides"] },
        "Parallelogram": { superclasses: ["Trapezoid"], properties: ["opposite sides parallel", "opposite sides equal", "opposite angles equal", "diagonals bisect each other"], incompatibilities: ["only one pair parallel sides"] },
        "Kite": { superclasses: ["Quadrilateral"], properties: ["2 pairs adjacent equal sides", "one pair opposite angles equal", "diagonals perpendicular"], incompatibilities: ["opposite sides parallel", "all sides equal"] },
        // "Isosceles Trapezoid": { superclasses: ["Trapezoid"], properties: ["one pair parallel sides", "base angles equal", "diagonals are equal"], incompatibilities: ["is equilateral", "has perpendicular diagonals"] }, // Removed for linear slider
        "Trapezoid": { superclasses: ["Quadrilateral"], properties: ["at least one pair parallel sides"], incompatibilities: ["no parallel sides", "all sides equal"] },
        "Quadrilateral": { superclasses: [], properties: ["4 sides"], incompatibilities: ["is a triangle", "has 5 sides"] }
    };
    // Define the chain for sliders - ensure it matches hierarchy logic where needed
    const hierarchyChainForSlider = ["Quadrilateral", "Trapezoid", "Parallelogram", "Rectangle", "Square"]; // Weakest to Strongest


    function isSubclass(shapeA, shapeB) {
        if (shapeA === shapeB) return true;
        if (!shapeHierarchy[shapeA] || !shapeHierarchy[shapeB]) {
            // console.warn(`isSubclass: Shape definition missing for ${shapeA} or ${shapeB}`);
            return false;
        }
        let queue = [...(shapeHierarchy[shapeA].superclasses || [])];
        let visited = new Set([shapeA]);
        while (queue.length > 0) {
            const current = queue.shift();
            if (current === shapeB) return true;
            if (!visited.has(current) && shapeHierarchy[current]) {
                visited.add(current);
                if (shapeHierarchy[current].superclasses) {
                     queue.push(...shapeHierarchy[current].superclasses);
                }
            }
        }
```

```
        return false;
    }

    // --- Module Initializers ---

    // Module 1: Intro
    moduleInitializers[0] = function() {
        const vizSquare = document.getElementById('viz-square-m1');
        const vizRectangle = document.getElementById('viz-rectangle-m1');
        if (vizSquare) vizSquare.innerHTML = getShapeSvg("Square");
        if (vizRectangle) vizRectangle.innerHTML = getShapeSvg("Rectangle");
    };

        // *** REVISED Module 2: Quadrilateral Checklist Initializer ***
        moduleInitializers[1] = function setupQuadrilateralChecklist() {
            // Define shape data using 'false' for "rejects property / incompatibility holds"
            // Corresponds to Table 1 logic where X means incompatibility (rejects the "No..." property)
            const shapesDataM2 = [
                // Property keys match checkbox IDs r1-r6
                { name: "Square",       r1: false, r2: false, r3: false, r4: false, r5: false, r6: false },
                // Rejects all "No..." props
                { name: "Rectangle",    r1: false, r2: true,  r3: false, r4: true,  r5: false, r6: false },
                // Allows r2, r4
                { name: "Rhombus",      r1: false, r2: false, r3: false, r4: false, r5: false, r6: true  },
                // Allows r6
                { name: "Parallelogram",r1: false, r2: true,  r3: false, r4: true,  r5: false, r6: true  },
                // Allows r2, r4, r6
                { name: "Trapezoid",    r1: true,  r2: true,  r3: true,  r4: true,  r5: false, r6: true  },
                // Only rejects r5
                { name: "Kite",         r1: false, r2: false, r3: true,  r4: false, r5: true,  r6: true  },
                // Allows r3, r5, r6
                { name: "Quadrilateral",r1: true,  r2: true,  r3: true,  r4: true,  r5: true,  r6: true  }
                // Allows all
            ];

            // Map names to data for easier lookup later (e.g., in Module 4)
            window.shapeDataMap = shapesDataM2.reduce((acc, shape) => {
                acc[shape.name] = shape;
                return acc;
            }, {});


            const checkboxesContainer = document.getElementById('restrictionCheckboxesM2');
            if (!checkboxesContainer) {
                 console.error("Module 2 checkboxes container not found!");
                 return; // Exit if container is missing
            }
            const checkboxes = checkboxesContainer.querySelectorAll('input[type="checkbox"]');
            const shapesContainer = document.getElementById('shapesContainerM2'); // Use the new ID

            // --- Calculate Strength ---
            // Strength = number of properties the shape REJECTS (has 'false' for)
            function calculateStrength(shape) {
                let strength = 0;
                for (let i = 1; i <= 6; i++) {
                    if (shape[`r${i}`] === false) {
                        strength++;
                    }
                }
                return strength;
            }

            // Add strength to the main data structure for easy access
            shapesDataM2.forEach(shape => {
                shape.strength = calculateStrength(shape);
```

```
        });


        function getActiveRestrictions() {
            const active = {};
            checkboxes.forEach(cb => {
                active[cb.dataset.propertyKey] = cb.checked;
            });
            return active;
        }

        function filterShapes() {
            const activeRestrictions = getActiveRestrictions();
            const targetValue = false; // A shape survives if it REJECTS the property (has 'false')
            ↪ when the restriction is active

            return shapesDataM2.filter(shape => {
                for (const restrictionKey in activeRestrictions) {
                    // If the restriction checkbox IS CHECKED (activeRestrictions[restrictionKey] is
                    ↪ true)...
                    if (activeRestrictions[restrictionKey]) {
                        // ...then the shape MUST have 'false' for this property to survive.
                        if (shape[restrictionKey] !== targetValue) {
                            return false; // Filter out this shape
                        }
                    }
                }
                return true; // Survived all active restrictions
            });
        }

        function updateShapesDisplay() {
            if (!shapesContainer) return; // Safety check
            const filteredShapes = filterShapes();
            shapesContainer.innerHTML = ''; // Clear previous shapes
            if (filteredShapes.length === 0) {
                shapesContainer.innerHTML = '<p>No quadrilaterals match the current
                ↪ restrictions.</p>';
            } else {
                // Sort shapes maybe by strength? (Optional)
                // filteredShapes.sort((a, b) => b.strength - a.strength);

                filteredShapes.forEach(shape => {
                    const div = document.createElement('div');
                    div.className = 'shape-item';
                    div.style.textAlign = 'center';
                    div.style.margin = '10px';
                    // Add SVG and strength label below it
                    div.innerHTML = getShapeSvg(shape.name) +
                        `<div style="margin-top: 8px; font-size: 0.9em; font-weight: 600; color:
                        ↪ #667eea;">Strength: ${shape.strength}</div>`;
                    shapesContainer.appendChild(div);
                });
            }
        }

        // Attach event listeners to checkboxes within this module
        checkboxes.forEach(cb => {
            cb.addEventListener('change', updateShapesDisplay);
        });

        // Initial display update for this module
        updateShapesDisplay();
    }; // End Module 2 Initializer

```

```
223
224     // Module 3: Substitution Conceptual Intro
225     moduleInitializers[2] = function() {
226         // This module is now primarily text and static examples in the HTML.
227         // No dynamic JS needed unless you add interactive highlighting later.
228     };
229
230     // Module 4: Polarity Demo
231         // *** REVISED Module 4: Polarity Demo Initializer ***
232     moduleInitializers[3] = function setupPolarityDemo() {
233         // Ensure shape data is available from Module 2
234         if (typeof window.shapeDataMap === 'undefined') {
235             console.error("Shape data not initialized from Module 2. Run Module 2 first.");
236             // Attempt to run Module 2 initializer if it hasn't run
237             if (typeof moduleInitializers[1] === 'function') {
238                 console.warn("Attempting to initialize Module 2 now...");
239                 try {
240                     moduleInitializers[1]();
241                     moduleInitializers[1] = null; // Mark as run
242                     if (typeof window.shapeDataMap === 'undefined') {
243                         // Still failed
244                         alert("Error: Module 2 data needed for Module 4. Please reload and navigate
                            ↪  sequentially.");
245                         return;
246                     }
247                 } catch(e) {
248                     alert("Error initializing Module 2 data. Please reload and navigate
                        ↪  sequentially.");
249                     return;
250                 }
251             } else {
252                 alert("Module 2 already initialized but data missing. Please reload.");
253                 return;
254             }
255         }
256
257         // Use unique IDs for Module 4 elements
258         const fixedSelect = document.getElementById('fixedConceptSelectM4');
259         const strengthSlider = document.getElementById('strengthSliderM4');
260         const varConceptLabel = document.getElementById('variableConceptLabelM4');
261         const varConceptLabelCond = document.getElementById('variableConceptLabelCondM4');
262         const vizP_El = document.getElementById('vizPM4');
263         const vizQ_R_El = document.getElementById('vizQM4'); // Renamed for clarity, shows Q
264         const relationArrowEl = document.getElementById('relationArrowM4');
265         const strengthPEl = document.getElementById('strengthPM4'); // Span for P strength
266         const strengthQEl = document.getElementById('strengthQM4'); // Span for Q strength
267
268
269         const baseInferEl = document.getElementById('baseInferM4');
270         const converseInferEl = document.getElementById('converseInferM4');
271         const contraInferEl = document.getElementById('contraInferM4');
272         const inverseInferEl = document.getElementById('inverseInferM4');
273
274         const propertyRSelect = document.getElementById('propertyRSelectM4');
275         const propertyRLabel = document.getElementById('propertyRLabelM4');
276         const condAntecedentEl = document.getElementById('condAntecedentM4');
277         const condAntecedentStatusEl = document.getElementById('condAntecedentStatusM4');
278         const condConsequentEl = document.getElementById('condConsequentM4');
279         const condConsequentStatusEl = document.getElementById('condConsequentStatusM4');
280
281         // Use the defined slider chain
282         const sliderChain = hierarchyChainForSlider; // Uses the global constant
283
284         // Populate fixed concept select
285         fixedSelect.innerHTML = sliderChain.map(t => `<option value="${t}">${t}</option>`).join('');
```

```
286         fixedSelect.value = "Rectangle"; // Default Q
287
288         // Populate Property R select (remains the same)
289         const availableProperties = {
290             "HAS_4_SIDES": { value: "has 4 sides", description: "has 4 sides" },
291             "HAS_PARALLEL_SIDES": { value: "at least one pair parallel sides", description: "at least
                ↪  one pair parallel sides" },
292             "OPPOSITE_SIDES_PARALLEL": { value: "opposite sides parallel", description: "opposite sides
                ↪  parallel" },
293             "HAS_4_RIGHT_ANGLES": { value: "4 right angles", description: "has 4 right angles" },
294             "HAS_4_EQUAL_SIDES": { value: "4 equal sides", description: "has 4 equal sides" }
295         };
296         propertyRSelect.innerHTML = Object.values(availableProperties)
297             .map(p => `<option value="${p.value}">${p.description}</option>`)
298             .join('');
299         propertyRSelect.value = "4 right angles"; // Default R
300
301         // Set Slider Labels and Range
302         document.getElementById('sliderMinLabelM4').textContent = sliderChain[0];
303         document.getElementById('sliderMaxLabelM4').textContent = sliderChain[sliderChain.length - 1];
304         strengthSlider.max = sliderChain.length - 1;
305         strengthSlider.value = sliderChain.findIndex(s => s === "Square"); // Default P = Strongest
306
307          // --- Validity Helper Functions (remain the same) ---
308          function conceptEntailsProperty(conceptP, propertyR_value) {
309               if (!shapeHierarchy[conceptP]) return false;
310               let queue = [conceptP];
311               let visited = new Set();
312               while (queue.length > 0) {
313                   let node = queue.shift();
314                   if (visited.has(node)) continue;
315                   visited.add(node);
316                   if (shapeHierarchy[node]?.properties?.includes(propertyR_value)) {
317                       return true;
318                   }
319                   if (shapeHierarchy[node]?.superclasses) {
320                       shapeHierarchy[node].superclasses.forEach(sc => { if (!visited.has(sc))
                            ↪  queue.push(sc); });
321                   }
322               }
323               return false;
324           }
325
326           function propertyEntailsConcept(propertyR_value, conceptP) {
327               if (!shapeHierarchy[conceptP]) return false;
328               let shapesThatGuaranteeR = [];
329               for (const shape in shapeHierarchy) {
330                   if (conceptEntailsProperty(shape, propertyR_value)) {
331                       shapesThatGuaranteeR.push(shape);
332                   }
333               }
334               if (shapesThatGuaranteeR.length === 0) return false;
335               // Check if *all* shapes guaranteeing R are subclasses of conceptP
336               return shapesThatGuaranteeR.every(shape => isSubclass(shape, conceptP));
337           }
338
339         function updatePolarityDemo() {
340             const fixedConceptQ = fixedSelect.value;
341             const sliderIndex = parseInt(strengthSlider.value);
342             const variableConceptP = sliderChain[sliderIndex];
343             const selectedPropertyR_value = propertyRSelect.value;
344             const selectedPropertyR_desc = propertyRSelect.options[propertyRSelect.selectedIndex].text;
345
346             // --- Get Strengths using Module 2 data ---
347             const strengthP = window.shapeDataMap[variableConceptP]?.strength ?? '?';
```

```
348             const strengthQ = window.shapeDataMap[fixedConceptQ]?.strength ?? '?';
349
350             // Update labels and strengths
351             varConceptLabel.textContent = variableConceptP;
352             strengthPEl.textContent = strengthP;
353             strengthQEl.textContent = strengthQ;
354             varConceptLabelCond.textContent = variableConceptP;
355             propertyRLabel.textContent = selectedPropertyR_desc;
356
357             vizP_El.innerHTML = getShapeSvg(variableConceptP);
358             vizQ_R_El.innerHTML = getShapeSvg(fixedConceptQ); // Display Q here
359
360             const pImpliesQ = isSubclass(variableConceptP, fixedConceptQ);
361             const qImpliesP = isSubclass(fixedConceptQ, variableConceptP);
362             let relationSymbol = '□ ';
363             if (pImpliesQ && qImpliesP) { relationSymbol = '⇔ Equivalent'; }
364             else if (pImpliesQ) { relationSymbol = `⇒ Stronger (S=${strengthP}) ⇒`; } // Show P is
                ↪  stronger
365             else if (qImpliesP) { relationSymbol = `⇐ Weaker (S=${strengthP}) ⇐`; } // Show P is weaker
366             else { relationSymbol = ' unrelated '; } // Added case for unrelated
367             relationArrowEl.textContent = relationSymbol;
368
369             // Update Base/Converse/Contra/Inverse validity displays (logic remains the same)
370             baseInferEl.innerHTML = `Base (P ⇒ Q): If X is <span class="term">${variableConceptP}</span>
                ↪  then X is <span class="term">${fixedConceptQ}</span>? <span class="status-indicator
                ↪  ${pImpliesQ ? 'valid' : 'invalid'}">${pImpliesQ ? 'Valid' : 'Invalid'}</span>`;
371             converseInferEl.innerHTML = `Converse (Q ⇒ P): If X is <span
                ↪  class="term">${fixedConceptQ}</span> then X is <span
                ↪  class="term">${variableConceptP}</span>? <span class="status-indicator ${qImpliesP ?
                ↪  'valid' : 'invalid'}">${qImpliesP ? 'Valid' : 'Invalid'}</span>`;
372             const notQImpliesNotP = pImpliesQ; // Validity matches base
373             const notPImpliesNotQ = qImpliesP; // Validity matches converse
374             contraInferEl.innerHTML = `Contrapositive (¬Q ⇒ ¬P): If X is not <span
                ↪  class="term">${fixedConceptQ}</span> then X is not <span
                ↪  class="term">${variableConceptP}</span>? <span class="status-indicator ${notQImpliesNotP
                ↪  ? 'valid' : 'invalid'}">${notQImpliesNotP ? 'Valid' : 'Invalid'}</span>`;
375             inverseInferEl.innerHTML = `Inverse (¬P ⇒ ¬Q): If X is not <span
                ↪  class="term">${variableConceptP}</span> then X is not <span
                ↪  class="term">${fixedConceptQ}</span>? <span class="status-indicator ${notPImpliesNotQ ?
                ↪  'valid' : 'invalid'}">${notPImpliesNotQ ? 'Valid' : 'Invalid'}</span>`;
376
377             // --- Calculate Validity ONCE at the beginning of this section ---
378             const ifPThenR_Valid = conceptEntailsProperty(variableConceptP, selectedPropertyR_value);
379             const ifRThenP_Valid = propertyEntailsConcept(selectedPropertyR_value, variableConceptP);
380
381             // --- Update Conditional Embedding displays (using the calculated values) ---
382
383             // Rebuild Antecedent Element Content
384             const antecedentSentenceHTML = `Antecedent Position: "If X is <span
                ↪  class='term'>${variableConceptP}</span>, then X ${selectedPropertyR_desc}"`;
385             condAntecedentEl.innerHTML = antecedentSentenceHTML; // Set the sentence HTML
386             // Use the already calculated 'ifPThenR_Valid'
387             condAntecedentStatusEl.className = `status-indicator ${ifPThenR_Valid ? 'valid' :
                ↪  'invalid'}`;
388             condAntecedentStatusEl.textContent = ifPThenR_Valid ? 'Valid' : 'Invalid';
389             condAntecedentEl.appendChild(condAntecedentStatusEl); // Append the status indicator
390
391             // Rebuild Consequent Element Content
392             const consequentSentenceHTML = `Consequent Position: "If X ${selectedPropertyR_desc}, then X
                ↪  is <span class='term'>${variableConceptP}</span>"`;
393             condConsequentEl.innerHTML = consequentSentenceHTML; // Set the sentence HTML
394             // Use the already calculated 'ifRThenP_Valid'
395             condConsequentStatusEl.className = `status-indicator ${ifRThenP_Valid ? 'valid' :
                ↪  'invalid'}`;
```

```
            condConsequentStatusEl.textContent = ifRThenP_Valid ? 'Valid' : 'Invalid';
            condConsequentEl.appendChild(condConsequentStatusEl); // Append the status indicator
        }

        // Event Listeners
        fixedSelect.addEventListener('change', updatePolarityDemo);
        strengthSlider.addEventListener('input', updatePolarityDemo);
        propertyRSelect.addEventListener('change', updatePolarityDemo);

        updatePolarityDemo(); // Initial call
    }; // End Module 4 Initializer

    // Module 5: Substitution Argument + Animation
    moduleInitializers[4] = function setupSubstitutionDemoViz() {
        const exampleSelect = document.getElementById('substExampleSelectViz');
        const frameSelect = document.getElementById('frameSelectViz');
        const frameVizEl = document.getElementById('frameViz');
        const exprA_VizEl = document.getElementById('exprA_Viz');
        const exprB_VizEl = document.getElementById('exprB_Viz');
        const animateBtn = document.getElementById('animateSubstButtonViz');
        const resetBtn = document.getElementById('resetSubstButtonViz');
        const baseSentenceEl = document.getElementById('baseSentenceViz');
        const resultSentenceEl = document.getElementById('resultSentenceViz');
        const infer1StatusEl = document.getElementById('infer1StatusViz');
        const infer2StatusEl = document.getElementById('infer2StatusViz');
        const analysisEl = document.getElementById('substAnalysisViz'); // The explanation area

        // Define frames with polarity info
        const frames = {
            simple_assertion: {
                template: "Shape S {expr}.",
                display: "Shape S _.",
                isInverting: false,
                property: null
            },
            conditional_antecedent: { // UPDATED
                template: "If Shape S {expr}, then S is a Rhombus.", // New Consequent
                display: "If Shape S _, then S is a Rhombus.", // New Display
                isInverting: true,
                property: "is a Rhombus" // Store the consequent property
            },
            negation: {
                template: "It is NOT the case that Shape S {expr}.",
                display: "It is NOT the case that Shape S _.",
                isInverting: true,
                property: null
            }
        };

        // Define examples with underlying rules
        const examples = {
            singularTerms: {
                exprA: "Mark Twain",
                exprB: "Samuel Clemens",
                isSymmetric: true,
                underlyingRule: "A ↔ B (Assumed Co-referential)",
                explanationTemplate: "Substituting co-referential terms ('${exprA}' ↔ '${exprB}').
                ↪  Significance is SYMMETRIC."
            },
            predicatesStrongerToWeaker: {
                exprA: "is a Square",
                exprB: "is a Rectangle",
                isSymmetric: false,
                underlyingRule: "A ⇒ B (Square ⇒ Rectangle)",
```

```
                explanationTemplate: "Substituting a stronger predicate ('${exprA}') with a weaker one ('${exprB}'). Significance is ASYMMETRIC."
            },
            predicatesWeakerToStronger: {
                exprA: "is a Rectangle",
                exprB: "is a Square",
                isSymmetric: false,
                underlyingRule: "A ⇏ B (Rectangle ⇏ Square), but B ⇒ A", // Note the base directionality
                explanationTemplate: "Substituting a weaker predicate ('${exprA}') with a stronger one ('${exprB}'). Significance is ASYMMETRIC."
            }
        };

         let animationTimeout = null; // To clear existing timeouts on reset/change

        // --- Helper to build sentences with spans for animation ---
         function buildSentence(template, expression) {
              // Simple replacement for now, assuming one placeholder '{expr}'
              // Escape expression to prevent HTML issues if needed, though unlikely here
              const escapedExpr = expression; // Simplification
              const placeholder = "{expr}";
              const parts = template.split(placeholder);
              if (parts.length === 2) {
                  // Wrap the expression part in a span for animation targeting
                  return parts[0] + `<span class="substituted-part">${escapedExpr}</span>` + parts[1];
              }
              return template.replace(placeholder, `<span class="substituted-part">${escapedExpr}</span>`); // Fallback
         }

        // --- Setup the initial view ---
        function setupInitialView() {
            clearTimeout(animationTimeout); // Clear any pending animation
            animateBtn.disabled = false; // Re-enable button

            const selectedExampleKey = exampleSelect.value;
            const selectedFrameKey = frameSelect.value;
            const example = examples[selectedExampleKey];
            const frame = frames[selectedFrameKey];

            exprA_VizEl.textContent = example.exprA;
            exprB_VizEl.textContent = example.exprB;
            frameVizEl.textContent = frame.display;

             // Reset styling on expression boxes
             exprA_VizEl.classList.remove('highlight-replace', 'lift-out-anim', 'fade-out-anim');
             exprB_VizEl.classList.remove('highlight-incoming');
             exprA_VizEl.style.opacity = '1'; exprA_VizEl.style.transform = '';
             exprB_VizEl.style.opacity = '1'; exprB_VizEl.style.transform = '';


            // Build and display initial sentences
            baseSentenceEl.innerHTML = buildSentence(frame.template, example.exprA);
            // Reset result sentence visually, perhaps hide it initially or show placeholder
            resultSentenceEl.innerHTML = buildSentence(frame.template, '...'); // Placeholder
            resultSentenceEl.style.opacity = 0.5; // Dim it initially

            // Clear status indicators
            infer1StatusEl.textContent = ''; infer1StatusEl.className = 'status-indicator';
            infer2StatusEl.textContent = ''; infer2StatusEl.className = 'status-indicator';

            // Reset analysis text
            analysisEl.innerHTML = `<h4>4. Analysis: What Happens?</h4><p>Select an example and context, then click  Animate to see the substitution and evaluate the inferences.</p>`;
```

```
519
520             // Ensure spans inside sentences are reset
521             baseSentenceEl.querySelectorAll('.substituted-part, .substituting-part').forEach(span => {
522                 span.style.opacity = ''; span.style.transform = '';
523                 span.className = 'substituted-part'; // Ensure it starts as the base part
524             });
525              resultSentenceEl.querySelectorAll('.substituted-part, .substituting-part').forEach(span =>
                 ↪  {
526                  span.textContent = '...';
527                  span.style.opacity = '0'; span.style.transform = '';
528                  span.className = 'substituted-part';
529              });
530
531         }
532
533
534         // --- Run the animation ---
535         function runSubstitutionAnimation() {
536             clearTimeout(animationTimeout); // Clear previous timeouts
537             animateBtn.disabled = true; // Disable button during animation
538
539             const selectedExampleKey = exampleSelect.value;
540             const selectedFrameKey = frameSelect.value;
541             const example = examples[selectedExampleKey];
542             const frame = frames[selectedFrameKey];
543
544             const baseSpan = baseSentenceEl.querySelector('.substituted-part');
545             const resultSpanTemplate = `<span class="substituting-part">${example.exprB}</span>`; //
                 ↪  Prepare the incoming part
546
547             // 1. Highlight the term to be substituted
548             exprA_VizEl.classList.add('highlight-replace');
549             if(baseSpan) baseSpan.style.backgroundColor = '#a8d5ff'; // Highlight in sentence
550
551             animationTimeout = setTimeout(() => {
552                 // 2. Lift out the original term visually
553                 exprA_VizEl.classList.add('lift-out-anim');
554                 if(baseSpan) baseSpan.classList.add('lift-out-anim');
555
556                 animationTimeout = setTimeout(() => {
557                     // 3. Fade out the original term
558                     exprA_VizEl.classList.add('fade-out-anim');
559                      if (baseSpan) {
560                          baseSpan.classList.add('fade-out-anim');
561                      }
562
563                      // Prepare result sentence content *while* base fades
564                      resultSentenceEl.innerHTML = baseSentenceEl.innerHTML.replace(/<span
                         ↪  class="substituted-part lift-out-anim fade-out-anim"[^>]*>.*?<\/span>/,
                         ↪  resultSpanTemplate);
565                      resultSentenceEl.style.opacity = 1; // Make result sentence visible
566
567                     animationTimeout = setTimeout(() => {
568                         // 4. Highlight the incoming term
569                         exprB_VizEl.classList.add('highlight-incoming');
570                         const resultSpan = resultSentenceEl.querySelector('.substituting-part'); //
                             ↪  Find the newly added span
571
572                         animationTimeout = setTimeout(() => {
573                             // 5. Move in the new term
574                             exprB_VizEl.classList.remove('highlight-incoming'); // Remove highlight
575                             if (resultSpan) {
576                                 resultSpan.classList.add('move-in-anim'); // Trigger move-in style
577                             }
578
```

```
                        // 6. Evaluate and display inferences AFTER animation settles
                        animationTimeout = setTimeout(() => {
                            evaluateAndDisplayInferences(); // Calculate validity and update text
                            animateBtn.disabled = false; // Re-enable button
                            // Optional: Clean up animation classes on spans if needed, though reset handles it
                            if(baseSpan) { baseSpan.className = 'substituted-part'; baseSpan.style = ''; }
                            const finalResultSpan = resultSentenceEl.querySelector('.substituting-part');
                            if(finalResultSpan) { finalResultSpan.className = 'substituted-part'; finalResultSpan.style = ''; } // Treat it as base now
                        }, 400); // Wait for move-in animation

                    }, 300); // Duration of incoming highlight

                }, 300); // Duration of fade-out

            }, 300); // Duration of lift-out

        }, 200); // Initial highlight duration
    }


    // --- Evaluate Inferences (Simplified Logic - NEEDS Brandom's Logic) ---
     function evaluateAndDisplayInferences() {
        const selectedKey = exampleSelect.value;
        const selectedFrameKey = frameSelect.value;
        const example = examples[selectedKey];
        const frame = frames[selectedFrameKey];
        const termA = example.exprA;
        const termB = example.exprB;

        // --- Determine Underlying Material Implications ---
        let aMateriallyImpliesB, bMateriallyImpliesA;
        const isSymmetricExample = example.isSymmetric;

        if (isSymmetricExample) {
            aMateriallyImpliesB = true; // By definition for co-referential terms
            bMateriallyImpliesA = true;
        } else {
            // Use the conceptual hierarchy for predicates
            const conceptA = termA.startsWith("is a ") ? termA.substring(5) : termA;
            const conceptB = termB.startsWith("is a ") ? termB.substring(5) : termB;
            aMateriallyImpliesB = isSubclass(conceptA, conceptB); // e.g., Square is subclass of Rectangle
            bMateriallyImpliesA = isSubclass(conceptB, conceptA); // e.g., Rectangle is NOT subclass of Square
        }

        // --- Determine Validity of the Substitution Inference (Base => Result) ---
        let infer1_valid = false;
        if (isSymmetricExample) {
            infer1_valid = true; // Symmetric substitution is always valid in both directions *if terms co-refer*
        } else {
            if (frame.isInverting) {
                // Polarity inverted: Inference Base => Result is valid IF B materially implies A
                infer1_valid = bMateriallyImpliesA;
            } else {
                // Normal polarity: Inference Base => Result is valid IF A materially implies B
                infer1_valid = aMateriallyImpliesB;
            }
        }

```

```
            // --- Determine Validity of the Reverse Substitution Inference (Result => Base) ---
            let infer2_valid = false;
            if (isSymmetricExample) {
                infer2_valid = true; // Symmetric
            } else {
                if (frame.isInverting) {
                    // Polarity inverted: Inference Result => Base is valid IF A materially implies B
                    infer2_valid = aMateriallyImpliesB;
                } else {
                    // Normal polarity: Inference Result => Base is valid IF B materially implies A
                    infer2_valid = bMateriallyImpliesA;
                }
            }

            // --- Update DOM elements ---
            infer1StatusEl.textContent = infer1_valid ? 'Valid' : 'Invalid';
            infer1StatusEl.className = `status-indicator ${infer1_valid ? 'valid' : 'invalid'}`;
            infer2StatusEl.textContent = infer2_valid ? 'Valid' : 'Invalid';
            infer2StatusEl.className = `status-indicator ${infer2_valid ? 'valid' : 'invalid'}`;

           // Update analysis text based on results
            let analysisText = `<h4>4. Analysis: What Happened?</h4>`;
            analysisText += `<p>Underlying Rule: <strong>${example.underlyingRule}</strong>. Context Polarity: <strong>${frame.isInverting ? 'Inverting' : 'Non-Inverting'}</strong>.</p>`;
            if (isSymmetricExample) {
                analysisText += `<p>  With <strong>Symmetric Terms</strong>, the substitution is valid in <strong>both directions</strong> (${infer1_valid ? '✓' : '✗'} Base⇒Result, ${infer2_valid ? '✓' : '✗'} Result⇒Base), regardless of the context's polarity. This stability is key for terms referring to objects.</p>`;
            } else {
                analysisText += `<p>  With <strong>Asymmetric Predicates</strong>:</p><ul>`;
                analysisText += `<li>Base ⇒ Result validity (${infer1_valid ? '✓ Valid' : '✗ Invalid'}) ${frame.isInverting ? 'depends on the REVERSE material rule (B⇒A)' : 'depends on the FORWARD material rule (A⇒B)'} because the context is ${frame.isInverting ? 'INVERTING' : 'Non-Inverting'}.</li>`;
                analysisText += `<li>Result ⇒ Base validity (${infer2_valid ? '✓ Valid' : '✗ Invalid'}) ${frame.isInverting ? 'depends on the FORWARD material rule (A⇒B)' : 'depends on the REVERSE material rule (B⇒A)'}.</li>`;
                if (frame.isInverting && aMateriallyImpliesB !== bMateriallyImpliesA) {
                    analysisText += `<li style="color: purple; font-weight: bold;">  Notice the flip! The valid inference direction changed compared to a simple context because this context is inverting.</li>`;
                } else if (!frame.isInverting && aMateriallyImpliesB !== bMateriallyImpliesA) {
                    analysisText += `<li>This follows the basic asymmetric pattern, as the context is not inverting.</li>`;
                }
                 analysisText += `</ul>`;
                 analysisText += `<p><strong>The Breakdown:</strong> If '${termA}'/'${termB}' were playing the basic 'Substituted-For' role but *had* this asymmetric rule, the rule wouldn't work consistently across all contexts. One rule would demand A⇒B in simple contexts but B⇒A in inverting ones!</p>`;
                 analysisText += `<p><strong>Brandom's Conclusion:</strong> Therefore, the basic 'Substituted-For' role *must* have SYMMETRIC significance (like terms). The asymmetric rules (like Square⇒Rectangle) belong to the 'Frame' role (predicates), and logic handles the polarity flips for those frames correctly.</p>`;
            }
            analysisEl.innerHTML = analysisText;
        }


        // Event Listeners
        exampleSelect.addEventListener('change', setupInitialView);
        frameSelect.addEventListener('change', setupInitialView);
        animateBtn.addEventListener('click', runSubstitutionAnimation);
```

```
        resetBtn.addEventListener('click', setupInitialView);

        // Initial setup
        setupInitialView(); // Make sure the view is correct on load
    };

    // Module 6: Matrix (Original Module 5)
    moduleInitializers[5] = function() { /* Module 6 (Original 5) - Likely static */ };

    // Module 7: Conclusion (Original Module 6)
    moduleInitializers[6] = function() { /* Module 7 (Original 6) - Likely static */ };

    // --- Initialize first module ---
    showModule(0);

}); // End DOMContentLoaded
```

2 Quadrilateral_Substitution/brandom_styles.css

```css
/* brandom_styles.css */

/* --- General Page Styling --- */
body {
    font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Oxygen, Ubuntu,
    ↪  Cantarell, "Open Sans", "Helvetica Neue", sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background-color: #f8f9fa; /* Lighter background */
    color: #212529; /* Darker text for contrast */
}

header {
    background-color: #343a40; /* Dark header */
    color: #fff;
    padding: 1.2em 0;
    text-align: center;
    margin-bottom: 2em;
}

main {
    max-width: 950px;
    margin: 2em auto;
    padding: 2em;
    background-color: #fff;
    box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1); /* Softer shadow */
    border-radius: 8px;
}

footer {
    text-align: center;
    margin-top: 3em;
    padding: 1em;
    font-size: 0.85em;
    color: #6c757d; /* Muted footer text */
}

/* --- Module Styling --- */
.module {
    border: 1px solid #dee2e6; /* Light border */
    padding: 1.5em 2em;
    margin-bottom: 2.5em;
    background-color: #fff;
    transition: opacity 0.4s ease-in-out;
    border-radius: 5px;
}

.current-module {
    display: block;
    opacity: 1;
}

.hidden-module {
    display: none;
    opacity: 0;
}

/* --- Typography and Structure --- */
h1, h2, h3, h4 {
    color: #343a40; /* Consistent dark headings */
    margin-bottom: 0.8em;
```

```
}

h2 {
    border-bottom: 2px solid #ced4da; /* Subtler heading separator */
    padding-bottom: 0.4em;
    margin-top: 0;
}

h3 {
    border-bottom: 1px solid #e9ecef; /* Very light separator for H3 */
    padding-bottom: 0.3em;
    margin-top: 1.5em;
}

h4 {
    margin-top: 1.2em;
    color: #495057; /* Slightly lighter heading */
}

p {
    margin-bottom: 1em;
}

ul, ol {
    margin-bottom: 1em;
    padding-left: 1.8em; /* Slightly more indent */
}

li {
    margin-bottom: 0.6em;
}

code, .term {
    font-family: "SFMono-Regular", Consolas, "Liberation Mono", Menlo, Courier, monospace;
    background-color: #e9ecef; /* Light grey background */
    padding: 0.2em 0.4em;
    border-radius: 3px;
    font-size: 0.9em;
    border: 1px solid #dee2e6; /* Subtle border for terms */
}

.concept {
    font-weight: 600; /* Slightly bolder */
    color: #0056b3; /* Adjusted blue */
}

/* --- Explanations and Examples --- */
.explanation {
    background: #f0f4f8; /* Very light blue-grey */
    padding: 1em;
    border: 1px solid #c9d6e5;
    border-left: 5px solid #6c757d; /* Grey left border */
    margin-top: 1.2em;
    font-size: 0.95em;
    border-radius: 0 4px 4px 0;
}

.example {
    border-left: 4px solid #17a2b8; /* Teal border */
    padding-left: 1em;
    margin: 1.2em 0;
    font-style: italic;
    background-color: #f8f9fa;
    padding: 0.8em;
    border-radius: 4px;
```

```css
}

/* --- Interactive Area --- */
.interactive-area {
    margin: 2em 0;
    padding: 1.5em;
    border: 1px dashed #adb5bd; /* Dashed border */
    background-color: #fdfdfe;
    border-radius: 5px;
}

label {
    margin-right: 0.5em;
    display: inline-block;
    min-width: 150px;
    font-weight: 500;
    margin-bottom: 0.3em; /* Add space below labels */
}

select, input[type=range], button {
    font-size: 1em;
    padding: 0.5em; /* Slightly more padding */
    margin: 0.5em 0;
    vertical-align: middle;
    border-radius: 4px;
    border: 1px solid #ced4da;
}

input[type=range] {
    width: 50%;
    margin: 0 0.5em;
}

button {
    cursor: pointer;
    background-color: #6c757d; /* Bootstrap secondary grey */
    color: white;
    border: none;
    padding: 0.5em 1em;
    transition: background-color 0.2s ease;
}

button:hover:not(:disabled) {
    background-color: #5a687d; /* Darker grey on hover */
}

button:disabled {
    cursor: not-allowed;
    opacity: 0.6;
    background-color: #adb5bd; /* Lighter grey when disabled */
}

/* --- Navigation --- */
#navigation {
    text-align: center;
    margin-top: 2.5em;
    padding-bottom: 2em;
}

#navigation button {
    padding: 0.7em 1.5em;
    margin: 0 1em;
    background-color: #007bff; /* Bootstrap primary blue */
    font-size: 1em;
    color: white;
```

```
    border: none;
    border-radius: 4px;
}

#navigation button:hover:not(:disabled) {
    background-color: #0056b3; /* Darker blue on hover */
}

#navigation button:disabled {
    background-color: #6c757d; /* Use secondary grey for disabled nav */
}

#moduleIndicator {
    font-weight: bold;
    color: #495057; /* Dark grey */
    margin: 0 1.5em;
    vertical-align: middle;
}

/* --- Visualization --- */
.viz-container {
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 1em;
    margin: 1.5em 0;
    flex-wrap: wrap;
    padding: 15px;
    background-color: #e9ecef; /* Light background for viz */
    border-radius: 5px;
}

.shape-viz svg {
    border: 1px solid #ced4da; /* Slightly darker border */
    background-color: white;
    border-radius: 3px;
    overflow: visible; /* Ensure text below is visible */
}

.arrow {
    font-size: 1.8em;
    margin: 0 0.8em;
    color: #495057;
}

/* --- Status Indicators --- */
.status-indicator {
    font-weight: bold;
    padding: 0.3em 0.6em;
    border-radius: 4px;
    display: inline-block;
    margin-left: 0.7em; /* More space */
    font-size: 0.9em;
    vertical-align: baseline;
}

.valid {
    color: #155724; /* Darker green text */
    background-color: #d4edda; /* Light green background */
    border: 1px solid #c3e6cb; /* Green border */
}

.invalid {
    color: #721c24; /* Darker red text */
    background-color: #f8d7da; /* Light red background */
```

```
    border: 1px solid #f5c6cb; /* Red border */
}

/* --- Substitution Styles --- */
.frame-style {
    color: #495057;
    background-color: #e9ecef;
    padding: 0.2em 0.4em;
    border: 1px solid #ced4da;
    border-radius: 3px;
    font-family: monospace;
}

.substituted-for-style {
    color: #0d6efd; /* Primary blue */
    font-weight: bold;
    border-bottom: 2px solid #0d6efd;
}

.substituting-style {
    color: #198754; /* Success green */
    font-weight: bold;
    border-bottom: 2px solid #198754;
}

.expression-box {
    display: inline-block;
    padding: 0.4em 0.8em; /* Slightly larger */
    border: 1px solid #dee2e6;
    background-color: #fff;
    border-radius: 4px;
    margin: 0 0.6em;
    min-width: 90px;
    text-align: center;
    transition: transform 0.3s ease-out, opacity 0.3s ease-out;
}

.sentence-display .substituted-part {
    display: inline-block;
    background-color: #cfe2ff; /* Lighter primary blue */
    padding: 1px 4px;
    border-radius: 3px;
    border: 1px dashed #9ec5fe;
    transition: background-color 0.3s, opacity 0.3s, transform 0.3s; /* Add transition */
}

.sentence-display .substituting-part {
    display: inline-block;
    background-color: #d1e7dd; /* Lighter success green */
    padding: 1px 4px;
    border-radius: 3px;
    border: 1px dashed #a3cfbb;
    opacity: 0; /* Start invisible */
    transition: opacity 0.3s ease-in, transform 0.3s ease-in; /* Add transition */
}

/* --- Animation Classes --- */
.highlight-replace {
    box-shadow: 0 0 10px 3px rgba(0, 123, 255, 0.5); /* Brighter blue glow */
}

.lift-out {
    transform: translateY(-10px) scale(1.05);
    opacity: 0.7;
}
```

```
322
323 .fade-out {
324     opacity: 0 !important; /* Ensure it stays faded out */
325     transform: translateY(-20px) scale(0.8);
326 }
327
328 .move-in {
329     opacity: 1 !important; /* Ensure it becomes fully visible */
330     transform: translateY(0) scale(1);
331 }
332
333 /* --- Matrix Table --- */
334 .matrix-table {
335     border-collapse: collapse;
336     width: 100%;
337     margin: 1.5em 0;
338     font-size: 0.9em;
339     box-shadow: 0 1px 3px rgba(0,0,0,0.1);
340 }
341 .matrix-table th, .matrix-table td {
342     border: 1px solid #dee2e6;
343     padding: 12px; /* More padding */
344     text-align: left;
345     vertical-align: top;
346 }
347 .matrix-table th {
348     background-color: #e9ecef;
349     color: #495057;
350     font-weight: 600;
351 }
352 .matrix-table td ul {
353     padding-left: 1.5em;
354     margin-top: 0.5em;
355     margin-bottom: 0; /* Reduce extra space */
356 }
357 .matrix-table td li {
358     margin-bottom: 0.3em;
359 }
360
361 /* --- Info Icons & Tooltips --- */
362 .info-icon {
363     position: relative; /* Needed for absolute positioning of tooltip */
364     display: inline-block;
365     width: 18px; height: 18px;
366     background-color: #0d6efd;
367     color: white;
368     border-radius: 50%;
369     text-align: center;
370     font-size: 12px; line-height: 18px;
371     cursor: help;
372     margin-left: 8px; /* More spacing */
373     font-weight: bold;
374     vertical-align: middle;
375     position: relative; /* Needed for tooltip positioning */
376 }
377
378 .tooltip-text {
379     visibility: hidden;
380     width: 250px; /* Wider tooltips */
381     background-color: #343a40; /* Dark background */
382     color: #fff;
383     text-align: left;
384     border-radius: 6px;
385     padding: 10px; /* More padding */
386     position: absolute;
```

```
    z-index: 10; /* Ensure tooltip is on top */
    bottom: 140%; /* Position above the icon */
    left: 50%;
    margin-left: -125px; /* Center the tooltip */
    opacity: 0;
    transition: opacity 0.3s ease;
    font-size: 0.9em;
    font-weight: normal;
    box-shadow: 0 3px 8px rgba(0,0,0,0.4);
    pointer-events: none; /* Prevent tooltip from interfering with hover */
}

/* Arrow for tooltip */
.tooltip-text::after {
    content: "";
    position: absolute;
    top: 100%; /* At the bottom of the tooltip */
    left: 50%;
    margin-left: -5px;
    border-width: 5px;
    border-style: solid;
    border-color: #343a40 transparent transparent transparent;
}
.info-icon:hover .tooltip-text {
    visibility: visible;
    opacity: 1;
}

/* Styles for substitution animation elements */
.sentence-display .substituted-part {
    display: inline-block; /* Crucial for transform */
    background-color: #cfe2ff; /* Light blue */
    padding: 1px 4px;
    border-radius: 3px;
    border: 1px dashed #9ec5fe;
    transition: background-color 0.3s, opacity 0.3s, transform 0.3s ease-out;
    /* Start with normal styling */
}
.sentence-display .substituting-part {
    display: inline-block; /* Crucial for transform */
    background-color: #d1e7dd; /* Light green */
    padding: 1px 4px;
    border-radius: 3px;
    border: 1px dashed #a3cfbb;
    opacity: 0; /* Start invisible */
    transform: translateY(10px) scale(0.9); /* Start slightly below and small */
    transition: opacity 0.3s 0.2s ease-in, transform 0.3s 0.2s ease-in; /* Delayed transition */
}

/* Animation classes */
.highlight-replace {
    box-shadow: 0 0 10px 3px rgba(0, 123, 255, 0.5); /* Blue glow */
    transition: box-shadow 0.2s ease-in-out;
}
.highlight-incoming {
     box-shadow: 0 0 10px 3px rgba(25, 135, 84, 0.5); /* Green glow */
     transition: box-shadow 0.2s ease-in-out;
}

.lift-out-anim {
    transform: translateY(-10px) scale(1.05);
    opacity: 0.5;
}

.fade-out-anim {
```

```
    opacity: 0 !important;
    transform: translateY(-20px) scale(0.8);
}

.move-in-anim {
    opacity: 1 !important;
    transform: translateY(0) scale(1) !important; /* Ensure final state */
}

/* Adjust Reset Button color */
#resetSubstButtonViz {
    background-color: #ffc107; /* Warning yellow */
    color: #343a40; /* Dark text for contrast */
}
#resetSubstButtonViz:hover {
    background-color: #e0a800;
}
```

3 Quadrilateral_Substitution/inferential_strength.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Brandom Ch 4: Why Singular Terms?</title>
    <link rel="stylesheet" href="inferential_strength_styles.css">
    <!-- MathJax for mathematical notation -->
    <script src="https://polyfill.io/v3/polyfill.min.js?features=es6"></script>
    <script id="MathJax-script" async
    ↪  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js"></script>
</head>
<body>
    <header>
        <h1>Why Singular Terms? An Interactive Guide to Brandom (AR Ch 4)</h1>
        <p style="font-size: 0.9em; color: #ccc;">Exploring Substitution, Polarity, and the Structure of
        ↪  Concepts</p>
    </header>

    <main id="lessonContainer">
        <button class="back-button" onclick="window.location.href='../Calculator/index.html'">← Back to
        ↪  Calculator</button>

        <!-- MODULE 1: Intro to Inferential Roles -->
        <section id="module1" class="module current-module">
            <h2>Module 1: Meaning as Inferential Role</h2>
            <p>Welcome! This guide explores Robert Brandom's idea that the meaning of concepts lies in
            ↪  how they are used in reasoning (<span class="concept">inference</span>), rather than
            ↪  just what they point to (<span class="concept">reference</span>).</p>
            <p>We'll use geometric shapes like squares and rectangles. Their relationships provide clear
            ↪  examples of <span class="concept">material inferences</span> — inferences valid because
            ↪  of the *content* of the concepts, not just their logical form.</p>
            <div class="example">
                Consider the inference: "If X is a <span class="term">Square</span>, then X is a <span
                ↪  class="term">Rectangle</span>."
                This is a good inference because the concept 'Square' includes all the properties of
                ↪  'Rectangle' (and more). Understanding 'Square' involves knowing this connection.
            </div>
            <div class="viz-container">
                <div class="shape-viz" id="viz-square-m1"></div>
                <div class="arrow">⇒</div>
                <div class="shape-viz" id="viz-rectangle-m1"></div>
            </div>
            <div class="explanation">
                Brandom calls 'Square' <span class="concept">inferentially stronger</span> than
                ↪  'Rectangle'. The stronger concept entails the weaker one. In the next modules, we'll
                ↪  see how this simple idea, combined with substitution, helps explain the structure of
                ↪  language.
            </div>
        </section>



<!-- MODULE 2: Quadrilateral Checklist & Strength -->
<section id="module2" class="module hidden-module">
    <h2>Module 2: Content Example - Incompatibility & Strength</h2>
    <p>In Module 1, we saw that meaning involves inferential connections (like "Square ⇒ Rectangle").
    ↪  Brandom suggests another way to grasp content is through <span
    ↪  class="concept">incompatibility</span> — what a concept *rules out*. A more specific concept
    ↪  rules out more, making it <span class="concept">inferentially stronger</span>.</p>
    <p>Let's quantify this strength using quadrilaterals. We'll define strength as the <span
    ↪  class="concept">number of incompatibility restrictions</span> a shape rejects (its "hard
    ↪  no's").</p>
```

```
    <p>Use the checkboxes below. Each checked box enforces a "hard no". See which shapes remain possible and note their calculated strength.</p>

    <div class="interactive-area">
        <h3>Quadrilateral Incompatibility Restrictions</h3>
        <div class="checkbox-group" id="restrictionCheckboxesM2"> <!-- Added ID for easier JS targeting -->
            <label title="Shape cannot have all sides of different lengths.">
                <input type="checkbox" id="r1" data-property-key="r1" checked> Reject: "No sides are equal"
            </label>
            <label title="Shape cannot have every pair of adjacent sides be of different lengths.">
                <input type="checkbox" id="r2" data-property-key="r2" checked> Reject: "No pair of adjacent sides are equal"
            </label>
            <label title="Shape cannot have every pair of opposite sides be of different lengths.">
                <input type="checkbox" id="r3" data-property-key="r3" checked> Reject: "No pair of opposite sides are equal"
            </label>
            <label title="Shape cannot have unequal non-parallel sides (if applicable).">
                <input type="checkbox" id="r4" data-property-key="r4" checked> Reject: "Non-parallel sides are not congruent"
            </label>
            <label title="Shape cannot have zero pairs of parallel sides.">
                <input type="checkbox" id="r5" data-property-key="r5" checked> Reject: "No pair of opposite sides are parallel"
            </label>
            <label title="Shape cannot lack right angles.">
                <input type="checkbox" id="r6" data-property-key="r6" checked> Reject: "No angles are right angles"
            </label>
        </div>

        <div class="explanation">
            <p><strong>How It Works:</strong> Checking a box means the shape *must be incompatible* with that property (it must have a 'No' for it in Table 1, represented as `false` in the data). Unchecking a box *eases* that restriction.</p>
            <p>A shape's <span class="concept">Strength</span> is the count of checked restrictions it satisfies (i.e., the number of properties it rejects).</p>
        </div>

        <h4>Possible Quadrilaterals (and their Strength):</h4>
        <div id="shapesContainerM2" class="viz-container" style="min-height: 100px; justify-content: flex-start;"> <!-- Added ID, changed display slightly -->
            <!-- The list of matching shapes with SVG graphics and strength will appear here -->
            <p>Initializing...</p>
        </div>
    </div>

    <div class="explanation" style="margin-top: 2em;">
        <h4>Connecting to Inferential Roles</h4>
        <p>This demonstrates how the specific incompatibilities defining a concept determine its <span class="concept">inferential strength</span>. A Square (Strength 6) rejects all restrictions, while a general Quadrilateral (Strength 0) rejects none. Crucially, a Square rejects *all* the restrictions a Rectangle rejects, which reflects the inferential entailment: <span class="term">Square ⇒ Rectangle</span>. This strength metric will help us understand polarity inversion in Module 4.</p>
    </div>
</section>

        <!-- MODULE 3: Substitution Roles & Significance -->
```

```
93
94
95          <section id="module3" class="module hidden-module">
96              <h2>Module 3: Substitution Roles & Significance</h2>
97              <p>To understand Brandom's argument, we first need to grasp how he analyzes sentences using
                ↪  <span class="concept">substitution</span>. This involves seeing sentences as built from
                ↪  parts that can be swapped out.</p>
98
99              <h3>Substitution Roles (Syntax)</h3>
100             <p>When we substitute one expression for another within a sentence, we can distinguish two
                ↪  main roles:</p>
101             <ul>
102                 <li><span class="concept substituted-for-style">Substituted-For:</span> The expression
                    ↪  being replaced (e.g., '<span class="term">Mark Twain</span>' in "Mark Twain wrote
                    ↪  HF"). These are typically the basic building blocks.</li>
103                 <li><span class="concept frame-style">Substitutional Frame:</span> The part of the
                    ↪  sentence that remains constant when substitution occurs (e.g., '<span
                    ↪  class="term">... wrote Huckleberry Finn</span>'). These frames are derived from
                    ↪  sentences by seeing them as patterns.</li>
104             </ul>
105             <div class="example">
106                 Sentence: "<span class="term">Fido is a dog</span>"<br>
107                 If we substitute '<span class="term">Fido</span>' with '<span class="term">Rex</span>',
                    ↪  the frame is "<span class="term">... is a dog</span>". '<span
                    ↪  class="term">Fido</span>' is <span
                    ↪  class="substituted-for-style">substituted-for</span>.<br>
108                 If we replace '<span class="term">... is a dog</span>' with '<span class="term">... is a
                    ↪  mammal</span>', the frame itself is replaced. '<span class="term">... is a
                    ↪  dog</span>' acts as the <span class="frame-style">frame</span>.
109             </div>
110
111             <h3>Substitution Significance (Semantics)</h3>
112             <p>The *meaning* or content contributed by an expression is revealed by how substitution
                ↪  affects inferences:</p>
113             <ul>
114                 <li><span class="concept">Symmetric Significance:</span> Substituting A for B (or B for
                    ↪  A) always preserves the correctness of inferences the sentence is involved in.
                    ↪  Typical of <span class="concept">singular terms</span> that co-refer. The inference
                    ↪  is reversible (A ⇔ B).</li>
115                 <li><span class="concept">Asymmetric Significance:</span> Substituting A for B preserves
                    ↪  correctness only one way. Typical of <span class="concept">predicates</span> related
                    ↪  by strength (e.g., 'Square' ⇒ 'Rectangle', but not vice-versa). The inference is
                    ↪  one-way (A ⇒ B).</li>
116             </ul>
117              <div class="example">
118                 Symmetric: "Mark Twain wrote HF" ⇔ "Samuel Clemens wrote HF".<br>
119                 Asymmetric: "Shape S is a Square" ⇒ "Shape S is a Rectangle", but "Shape S is a
                    ↪  Rectangle" <span style="text-decoration: line-through;">⇒</span> "Shape S is a
                    ↪  Square".
120              </div>
121             <div class="explanation">
122                 Brandom argues that expressions playing the <span
                    ↪  class="substituted-for-style">substituted-for</span> syntactic role (like terms)
                    ↪  have <span class="concept">symmetric</span> semantic significance, while expressions
                    ↪  playing the <span class="frame-style">frame</span> role (like predicates) typically
                    ↪  have <span class="concept">asymmetric</span> significance. Module 4 explores *why*
                    ↪  this specific combination is necessary in a logically expressive language.
123             </div>
124         </section>
125
126         <!-- MODULE 4: Polarity & Logical Operators -->
127 <section id="module4" class="module hidden-module">
128     <h2>Module 4: Inferential Polarity - Strength & Logic</h2> <!-- Updated Title -->
```

```
<p>Logical operators like <span class="term">if...then...</span> and <span class="term">not</span> interact with the <span class="concept">inferential strength</span> of concepts (measured by rejected restrictions from Module 2). Some logical contexts <span class="concept">invert</span> the relationship between concept strength and the strength/validity of the overall claim.</p>

<div class="interactive-area">
    <h3>Polarity Demonstration</h3>

    <div style="margin-bottom: 1em;">
       <label for="fixedConceptSelectM4">Fixed Concept (Q):</label> <!-- Added M4 to ID -->
       <select id="fixedConceptSelectM4"></select>
       <strong style="margin-left:10px;">Strength: <span id="strengthQM4">–</span></strong> <!-- Display Q Strength -->
       <span class="info-icon" title="This concept remains constant in the Base/Converse/Contrapositive/Inverse inferences.">?</span>
    </div>

    <div style="margin-bottom: 1em;">
        <label for="strengthSliderM4">Variable Concept (P):</label> <!-- Added M4 to ID -->
        <span id="sliderMinLabelM4" style="font-size: 0.8em;">Weaker</span>
        <input type="range" id="strengthSliderM4" min="0" max="4" value="4" step="1"> <!-- Adjusted max based on sliderChain -->
        <span id="sliderMaxLabelM4" style="font-size: 0.8em;">Stronger</span>
        <br>
        <span style="margin-left: 155px; font-weight: bold;" id="variableConceptLabelM4">Square</span> <!-- Updated Default -->
        <strong style="margin-left:10px;">Strength: <span id="strengthPM4">–</span></strong> <!-- Display P Strength -->
         <span class="info-icon" title="Slide to change Concept P along the hierarchy: Quadrilateral ↔ Trapezoid ↔ Parallelogram ↔ Rectangle ↔ Square. Note its strength changes.">?</span>
    </div>

    <div class="viz-container">
        <div class="shape-viz" id="vizPM4"></div> <!-- Added M4 -->
        <div class="arrow" id="relationArrowM4">?</div> <!-- Added M4 -->
        <div class="shape-viz" id="vizQM4"></div> <!-- Added M4 -->
    </div>
    <hr>

    <h4>1. Base Material Inference (P vs Q)</h4>
    <div id="baseInferM4" class="example">P ⇒ Q: ...</div> <!-- Added M4 -->
    <div id="converseInferM4" class="example">Q ⇒ P: ...</div> <!-- Added M4 -->
    <div class="explanation">Observe how validity changes as you alter P's strength relative to Q's strength. <span class="concept">P ⇒ Q is valid if P is stronger than or equivalent to Q</span> (meaning P rejects at least all the properties Q rejects, i.e., `isSubclass(P, Q)`).</div>

    <h4>2. Negation & Polarity (Contrapositive & Inverse)</h4>
    <div id="contraInferM4" class="example">¬Q ⇒ ¬P: ...</div> <!-- Added M4 -->
    <div id="inverseInferM4" class="example">¬P ⇒ ¬Q: ...</div> <!-- Added M4 -->
     <div class="explanation">
         The <span class="concept">Contrapositive</span> (¬Q ⇒ ¬P) validity matches the Base Inference (P ⇒ Q). Why? Negation reverses the direction of inference: If having P's properties *forces* having Q's properties (P ⇒ Q), then *lacking* Q's properties must *force* lacking P's properties (¬Q ⇒ ¬P). The <span class="concept">Inverse</span> (¬P ⇒ ¬Q) validity matches the Converse (Q ⇒ P) for the same reason.
     </div>

    <h4>3. Conditional Embedding Polarity</h4>
     <div style="margin-bottom: 1em;">
         <label for="propertyRSelectM4">Fixed Property (R):</label> <!-- Added M4 -->
         <select id="propertyRSelectM4"></select>
         <span class="info-icon" title="Select the property used in the 'if' or 'then' clause of the conditionals below.">?</span>
```

```
176          </div>
177          <p style="font-size: 0.9em;">Comparing conditionals using the variable concept P (<span
             ↪  style="font-weight:bold;" id="variableConceptLabelCondM4">...</span>) and the fixed
             ↪  property R (<span style="font-weight:bold;" id="propertyRLabelM4">...</span>):</p>
178
179          <div id="condAntecedentM4" class="example"> <!-- Added M4 -->
180               Antecedent Position: "If X is <span class="term">[P]</span>, then X <span
                  ↪  class="term">[R]</span>"
181               <span id="condAntecedentStatusM4" class="status-indicator"></span>
182          </div>
183          <div id="condConsequentM4" class="example"> <!-- Added M4 -->
184               Consequent Position: "If X <span class="term">[R]</span>, then X is <span
                  ↪  class="term">[P]</span>"
185               <span id="condConsequentStatusM4" class="status-indicator"></span>
186          </div>
187          <div class="explanation">
188              The antecedent ('if...') position <span class="concept">inverts polarity</span>. Making P
                 ↪  <span style="color:red;">weaker</span> (sliding left, <span
                 ↪  style="color:red;">decreasing Strength P</span>) makes the conditional statement 'If P
                 ↪  then R' <span style="color:green;">stronger overall</span> as a claim. It demands R
                 ↪  hold true under *more* conditions (e.g., for all Rectangles, not just Squares).
                 ↪  Stronger claims are *less* likely to be valid. Conversely, making P <span
                 ↪  style="color:green;">stronger</span> (sliding right, <span
                 ↪  style="color:green;">increasing Strength P</span>) makes 'If P then R' <span
                 ↪  style="color:red;">weaker overall</span> (applies to fewer cases), making it *more*
                 ↪  likely to be valid. <br>
189              The consequent ('then...') position does <span class="concept">not invert polarity</span>.
                 ↪  Making P <span style="color:red;">weaker</span> (<span style="color:red;">decreasing
                 ↪  Strength P</span>) makes the conditional 'If R then P' <span style="color:red;">weaker
                 ↪  overall</span> (easier to satisfy, *more* likely valid).
190          </div>
191      </div>
192  </section>
193
194          <!-- MODULE 5: The Argument - Why Terms Must Be Symmetric -->
195  <!-- MODULE 5: The Argument - Why Terms Must Be Symmetric -->
196  <section id="module5" class="module hidden-module">
197      <h2>Module 5: The Argument - Why Terms Must Be Symmetric</h2>
198      <p>We've seen that expressions have <span class="concept">syntactic roles</span> (<span
         ↪  class="substituted-for-style">Substituted-For</span> vs. <span class="frame-style">Frame</span>)
         ↪  and <span class="concept">semantic significance</span> (Symmetric vs. Asymmetric inferential
         ↪  behavior, reflected in <span class="concept">Strength</span>). We also saw how logical contexts
         ↪  (<code class="term">if</code>, <code class="term">not</code>) can <span class="concept">invert
         ↪  polarity</span>, flipping inferential relationships (Module 4).</p>
199      <p><strong>Brandom's Core Question:</strong> Can expressions playing the <span
         ↪  class="substituted-for-style">Substituted-For</span> role (the role singular terms play)
         ↪  consistently have *Asymmetric* significance (like predicates, e.g., 'Square' ⇒ 'Rectangle') in a
         ↪  language that also uses logic?</p>
200      <p>Let's test this. Use the visualization with the improved "If S {expr}, then S is a Rhombus"
         ↪  context. Pay close attention to how the validity of the sentences and the inferences
         ↪  *changes*.</p>
201
202      <div class="interactive-area">
203          <h3>Substitution Visualization & Argument</h3>
204
205          <label for="substExampleSelectViz">1. Choose Substitution Type:</label>
206          <select id="substExampleSelectViz">
207               <option value="singularTerms">Symmetric: Terms ('Mark Twain' ↔ 'Samuel Clemens')</option>
208               <option value="predicatesStrongerToWeaker" selected>Asymmetric: Predicates ('Square' ⇒
                  ↪  'Rectangle')</option>
209               <option value="predicatesWeakerToStronger">Asymmetric: Predicates ('Rectangle' ⇒
                  ↪  'Square')</option>
210          </select>
```

```
        <span class="info-icon">? <span class="tooltip-text">Symmetric: Inference A⇒B and B⇒A are both valid (like co-referring terms). Asymmetric: Only one direction holds (A⇒B or B⇒A, like stronger/weaker predicates).</span></span>
        <br>

        <label for="frameSelectViz">2. Choose Sentence Context (Frame):</label>
        <select id="frameSelectViz">
            <option value="simple_assertion">Simple: "Shape S {expr}." (Non-Inverting)</option>
            <!-- UPDATED OPTION TEXT -->
            <option value="conditional_antecedent">Conditional Antecedent: "If Shape S {expr}, then S is a Rhombus." (Inverting)</option>
            <option value="negation">Negation: "It is NOT the case that Shape S {expr}." (Inverting)</option>
        </select>
         <span class="info-icon">? <span class="tooltip-text">The context where substitution happens. Conditional antecedents and negations are 'polarity-inverting' - they flip the strength relationship needed for validity compared to a simple assertion.</span></span>
        <br><br>

        <!-- Animation Area remains the same -->
        <div id="substitutionAnimationArea" style="margin-top: 1.5em; text-align: center; font-size: 1.1em;">
            <p style="margin-bottom: 0.5em;"><strong>Context/Frame:</strong> <span id="frameViz" class="frame-style">Frame Text...</span></p>
            <div style="margin: 1em 0; display: flex; justify-content: center; align-items: center; gap: 1em; flex-wrap: wrap;">
                 <span style="text-align: center;">Original Expression (A):<br><span id="exprA_Viz" class="expression-box substituted-for-style">Expr A</span></span>
                 <span class="arrow" style="flex-shrink: 0;">→ Substituting with →</span>
                 <span style="text-align: center;">New Expression (B):<br><span id="exprB_Viz" class="expression-box substituting-style">Expr B</span></span>
            </div>
             <button id="animateSubstButtonViz">  Animate Substitution</button>
             <button id="resetSubstButtonViz">  Reset View</button>
        </div>
        <hr style="margin: 1.5em 0;">

        <!-- Results Area remains the same structure -->
        <div id="substitutionResultAreaViz" style="margin-top: 1em;">
            <h4>3. Resulting Sentences & Inferences:</h4>
             <p><strong>Base Sentence:</strong> <span id="baseSentenceViz" class="term sentence-display">Initial sentence structure.</span></p>
             <p><strong>Result Sentence:</strong> <span id="resultSentenceViz" class="term sentence-display">Sentence after substitution.</span></p>

            <div id="subInfer1Viz" class="example">
                Inference 1: <span class="term">[Base]</span> implies <span class="term">[Result]</span>?
                <span id="infer1StatusViz" class="status-indicator"></span>
                 <span class="info-icon">?<span class="tooltip-text">Is inferring the Result sentence FROM the Base sentence valid? Depends on the expressions' rule (A⇒B or A⇔B) AND the context's polarity.</span></span>
            </div>
            <div id="subInfer2Viz" class="example">
                Inference 2: <span class="term">[Result]</span> implies <span class="term">[Base]</span>?
                <span id="infer2StatusViz" class="status-indicator"></span>
                 <span class="info-icon">?<span class="tooltip-text">Is inferring the Base sentence FROM the Result sentence valid? Depends on the expressions' rule (B⇒A or A⇔B) AND the context's polarity.</span></span>
            </div>
        </div>
        <hr style="margin: 1.5em 0;">

```

```
        <!-- SIGNIFICANTLY UPDATED Explanation -->
        <div id="substAnalysisViz" class="explanation" aria-live="polite">
             <h4>4. Analysis: Why Terms MUST Be Symmetric</h4>
             <p>Select "Asymmetric: Predicates ('Square' ⇒ 'Rectangle')" and the "Conditional Antecedent
             ↪  (...then S is a Rhombus)" context, then Animate.</p>
             <ul>
                 <li><strong>Base Sentence:</strong> "If S is <span class='term'>Square</span>, then S
                 ↪  is a Rhombus" is <span class='valid'>Valid</span> (geometrically true).</li>
                 <li><strong>Result Sentence:</strong> "If S is <span class='term'>Rectangle</span>,
                 ↪  then S is a Rhombus" is <span class='invalid'>Invalid</span> (geometrically
                 ↪  false).</li>
                 <li><strong>Inference 1 (Base ⇒ Result):</strong> Valid Sentence ⇒ Invalid Sentence.
                 ↪  This inference is <span class='invalid'>Invalid</span>.</li>
                 <li><strong>Inference 2 (Result ⇒ Base):</strong> Invalid Sentence ⇒ Valid Sentence.
                 ↪  This inference is <span class='valid'>Valid</span> (a falsehood implies
                 ↪  anything).</li>
                 <li style="margin-top:1em;">
                    <strong style="color:purple;">The Flip in Action:</strong> The underlying material
                    ↪  rule is <span class="term">Square</span> ⇒ <span class="term">Rectangle</span>
                    ↪  (A ⇒ B).
                    <ul>
                        <li>In a Simple Context (Module 3), substituting A with weaker B makes the
                        ↪  inference A⇒B <span class="valid">Valid</span>.</li>
                        <li>But here, in the <span class="concept">Inverting Context</span> (Conditional
                        ↪  Antecedent), the valid substitution inference was Inference 2 (Result ⇒
                        ↪  Base). This direction corresponds to needing the material rule B ⇒ A (<span
                        ↪  class="term">Rectangle</span> ⇒ <span class="term">Square</span>), which is
                        ↪  false! The direction required for a valid substitution has <span
                        ↪  style="font-weight:bold;">flipped</span> compared to the simple
                        ↪  context.</li>
                    </ul>
                 </li>
                 <li style="margin-top:1em;">
                    <strong style="color:red;">The Contradiction for Hypothetical "Asymmetric
                    ↪  Terms":</strong>
                    Imagine '<span class='substituted-for-style'>SquareTerm</span>' was a
                    ↪  Substituted-For term with the rule "<span class='term'>SquareTerm ⇒
                    ↪  RectangleTerm</span>".
                    <ul>
                        <li>This single rule (A⇒B) would need to make the substitution <span
                        ↪  class="term">[Base]</span>⇒<span class="term">[Result]</span> valid in
                        ↪  simple contexts.</li>
                        <li>But it would also need to make the substitution <span
                        ↪  class="term">[Result]</span>⇒<span class="term">[Base]</span> valid in
                        ↪  inverting contexts (like the one above).</li>
                        <li>One fixed asymmetric rule (A⇒B) cannot simultaneously satisfy both
                        ↪  requirements! It leads to a contradiction depending on the logical
                        ↪  context.</li>
                    </ul>
                 </li>
                  <li style="margin-top:1em;">
                     <strong style="color:green;">  Brandom's Conclusion:</strong>
                     To allow consistent substitution across *all* logical contexts, the <span
                     ↪  class="substituted-for-style">Substituted-For</span> role *must* have <span
                     ↪  class="concept">Symmetric</span> significance (A ⇔ B). This is why we have
                     ↪  singular terms that behave like 'Mark Twain' ⇔ 'Samuel Clemens', providing
                     ↪  stable reference. The asymmetries essential to meaning (like Square ⇒
                     ↪  Rectangle) belong to the <span class="frame-style">Frame</span> (predicate)
                     ↪  role, where logical operators correctly handle the necessary polarity flips.
                  </li>
             </ul>
              <p><em>Try substituting Symmetric Terms ('Twain'/'Clemens') - notice both Inference 1 and 2
              ↪  remain Valid in *all* contexts. Symmetry provides the required stability.</em></p>
        </div>
```

```
    </div>
</section>

        <!-- MODULE 6: The Matrix of Possibilities -->
         <section id="module6" class="module hidden-module">
             <h2>Module 6: The Matrix of Substitutional Possibilities</h2>
             <p>Brandom considers four theoretical ways a language could structure substitutional roles based on the syntactic role (<span class="concept substituted-for-style">Substituted-For</span> vs. <span class="concept frame-style">Frame</span>) and the semantic significance (<span class="concept">Symmetric</span> vs. <span class="concept">Asymmetric</span>).</p>
             <table class="matrix-table">
                 <thead>
                     <tr>
                         <th></th>
                         <th>Substitutional Frame has SYMMETRIC Significance</th>
                         <th>Substitutional Frame has ASYMMETRIC Significance</th>
                     </tr>
                 </thead>
                 <tbody>
                     <tr>
                         <th>Substituted-For expression has SYMMETRIC Significance</th>
                         <td><strong>(i) Both Symmetric</strong><br>
                            <ul><li><em>Problem:</em> Fails to capture asymmetric predicate inferences (e.g., genus/species). Too expressively weak.</li></ul>
                         </td>
                         <td><strong>(iv) Terms Symmetric, Predicates Asymmetric</strong><br>
                            <ul><li><em>Result:</em> This is Brandom's proposed structure, corresponding to Singular Terms (Substituted-For, Symmetric) and Predicates (Frames, Asymmetric). It allows both stable object reference and hierarchical conceptual relations, and is compatible with logic. □ </li></ul>
                        </td>
                     </tr>
                     <tr>
                         <th>Substituted-For expression has ASYMMETRIC Significance</th>
                         <td><strong>(ii) Terms Asymmetric, Predicates Symmetric</strong><br>
                            <ul><li><em>Problem:</em> Fails to capture asymmetric predicate inferences (as roles are swapped vs. (iv)). Also too weak.</li></ul>
                         </td>
                         <td><strong>(iii) Both Asymmetric</strong><br>
                            <ul><li><em>Problem:</em> Leads to incoherence when combined with logical operators (conditionals/negation) that invert polarity, as demonstrated in Module 4. Cannot consistently project inferences.</li></ul>
                        </td>
                     </tr>
                 </tbody>
             </table>
             <div class="explanation">
                 The argument presented in Module 4 aims to rule out options (i), (ii), and especially (iii) for any language rich enough to contain basic sentential logic. This leaves only option (iv), the familiar structure distinguishing singular terms from predicates based on their different syntactic roles and semantic (substitution-inferential) significances.
             </div>
         </section>

        <!-- MODULE 7: Conclusion -->
        <section id="module7" class="module hidden-module">
            <h2>Module 7: Conclusion - The Expressive Deduction</h2>
            <p>Brandom's argument provides an <span class="concept">expressive deduction of the necessity of singular terms and predicates</span>. It's 'expressive' because it relies on what's needed for a language to explicitly express its own inferential structure using logic. It's a 'deduction' because it argues this structure is a necessary consequence of combining basic substitution with logical operators.</p>
```

```
            <p>The argument shows that:</p>
            <ul>
                <li>Languages use <span class="concept">substitution</span> to create novel sentences from existing parts (projectibility).</li>
                <li>Understanding meaning <span class="concept">inferentially</span> involves tracking symmetric vs. asymmetric substitution patterns.</li>
                <li><span class="concept">Logical vocabulary</span> (conditionals, negation) makes these inferential patterns explicit but also introduces <span class="concept">polarity-inverting</span> contexts.</li>
                <li>Compatibility with these inverting contexts <span class="concept">requires</span> that the basic substituted-for expressions have <span class="concept">symmetric</span> significance (functioning as singular terms) while the substitutional frames have <span class="concept">asymmetric</span> significance (functioning as predicates).</li>
            </ul>
            <div class="explanation">
                Ultimately, the reason we structure our talk around <span class="concept">objects</span> (named by singular terms) having <span class="concept">properties</span> (ascribed by predicates) is deeply connected to our capacity for logical, inferential reasoning. It's the structure demanded by a language that can not only make claims but reflect on the inferential connections between them.
            </div>
        </section>
    </main>

    <nav id="navigation">
        <button id="prevButton" disabled>Previous Module</button>
        <span id="moduleIndicator">Module 1 of 7</span>
        <button id="nextButton">Next Module</button>
    </nav>

    <footer>
        <p>Based on Robert Brandom's "Articulating Reasons", Chapter 4.</p>
    </footer>

    <script src="brandom_lesson.js"></script> <!-- Link to your JS file -->
</body>
</html>
```

4 Quadrilateral_Substitution/inferential_strength_styles.css

```css
/* Inferential Strength Interactive Lesson - Improved Styles */

@import url('https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&family=Inter:wght@400;500;600;700&display=swap');

/* Basic Layout */
body {
    font-family: 'Inter', -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background: linear-gradient(135deg, #e0e0e0 0%, #c0c0c0 100%);
    color: #212529;
}

header {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: #fff;
    padding: 2em 0;
    text-align: center;
    margin-bottom: 2em;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}

header h1 {
    font-family: 'Orbitron', sans-serif;
    font-size: 2em;
    margin: 0 0 0.3em 0;
    font-weight: 700;
}

header p {
    font-size: 1em;
    color: rgba(255, 255, 255, 0.85);
    margin: 0;
}

main {
    max-width: 950px;
    margin: 2em auto;
    padding: 2em;
    background-color: #fff;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
    border-radius: 15px;
}

.module {
    border: 1px solid #dee2e6;
    padding: 2em;
    margin-bottom: 2.5em;
    background-color: #fff;
    transition: opacity 0.4s ease-in-out;
    border-radius: 10px;
}

.current-module {
    display: block;
    opacity: 1;
}

.hidden-module {
    display: none;
```

```
    opacity: 0;
}

/* Typography */
h1, h2, h3, h4 {
    font-family: 'Orbitron', sans-serif;
    color: #343a40;
}

h2 {
    border-bottom: 3px solid #667eea;
    padding-bottom: 0.5em;
    margin-top: 0;
    margin-bottom: 1.2em;
    color: #667eea;
    font-weight: 700;
}

h3 {
    border-bottom: 2px solid #e9ecef;
    padding-bottom: 0.4em;
    margin-top: 1.5em;
    font-weight: 600;
}

h4 {
    margin-top: 1.2em;
    color: #495057;
    font-weight: 600;
}

p {
    margin-bottom: 1em;
    font-family: 'Inter', sans-serif;
}

ul, ol {
    margin-bottom: 1em;
    padding-left: 1.5em;
}

li {
    margin-bottom: 0.5em;
}

code, .term {
    font-family: "SFMono-Regular", Consolas, "Liberation Mono", Menlo, Courier, monospace;
    background-color: #e9ecef;
    padding: 0.2em 0.4em;
    border-radius: 4px;
    font-size: 0.9em;
}

.concept {
    font-weight: 600;
    color: #667eea;
}

/* Explanations and Examples */
.explanation {
    background: #f8f9fa;
    padding: 1.2em;
    border: 1px solid #dee2e6;
    border-left: 5px solid #667eea;
    margin-top: 1.2em;
```

```
    font-size: 0.95em;
    border-radius: 5px;
}

.example {
    border-left: 4px solid #17a2b8;
    padding: 1em 1em 1em 1.2em;
    margin: 1.2em 0;
    font-style: italic;
    background-color: #f1f8fb;
    border-radius: 5px;
}

/* Interactive Elements */
.interactive-area {
    margin: 1.5em 0;
    padding: 1.8em;
    border: 2px dashed #adb5bd;
    background-color: #fdfdfe;
    border-radius: 10px;
}

.checkbox-group {
    display: flex;
    flex-direction: column;
    gap: 0.8em;
}

.checkbox-group label {
    display: flex;
    align-items: center;
    font-weight: 500;
    padding: 0.6em;
    background: #f8f9fa;
    border-radius: 5px;
    transition: background-color 0.2s;
    cursor: pointer;
}

.checkbox-group label:hover {
    background: #e9ecef;
}

.checkbox-group input[type="checkbox"] {
    margin-right: 0.8em;
    width: 18px;
    height: 18px;
    cursor: pointer;
}

label {
    margin-right: 0.5em;
    display: inline-block;
    min-width: 150px;
    font-weight: 500;
}

select, input[type=range], button {
    font-family: 'Inter', sans-serif;
    font-size: 1em;
    padding: 0.5em;
    margin: 0.5em 0;
    vertical-align: middle;
    border-radius: 6px;
    border: 1px solid #ced4da;
```

```
}

input[type=range] {
    width: 50%;
    margin: 0 0.5em;
    cursor: pointer;
}

button {
    font-family: 'Orbitron', sans-serif;
    cursor: pointer;
    background-color: #667eea;
    color: white;
    border: none;
    padding: 0.6em 1.2em;
    transition: background-color 0.2s, transform 0.1s;
    font-weight: 500;
}

button:hover:not(:disabled) {
    background-color: #5568d3;
    transform: translateY(-1px);
}

button:active:not(:disabled) {
    transform: translateY(0);
}

button:disabled {
    cursor: not-allowed;
    opacity: 0.5;
    background-color: #6c757d;
}

/* Navigation */
#navigation {
    text-align: center;
    margin-top: 2.5em;
    padding: 2em 0;
}

#navigation button {
    padding: 0.8em 1.8em;
    margin: 0 1em;
    background-color: #667eea;
    font-size: 1em;
}

#navigation button:hover:not(:disabled) {
    background-color: #5568d3;
}

#navigation button:disabled {
    background-color: #6c757d;
}

#moduleIndicator {
    font-family: 'Orbitron', sans-serif;
    font-weight: 600;
    color: #667eea;
    margin: 0 1em;
    font-size: 1.1em;
}

.back-button {
```

```
    background-color: #90ee90;
    color: #2d5016;
    margin-bottom: 20px;
}

.back-button:hover {
    background-color: #a0ffa0;
}

/* Visualization */
.viz-container {
    display: flex;
    align-items: center;
    justify-content: center;
    gap: 1.5em;
    margin: 1.5em 0;
    flex-wrap: wrap;
    padding: 20px;
    background-color: #f8f9fa;
    border-radius: 10px;
    border: 1px solid #dee2e6;
}

.shape-viz svg {
    border: 2px solid #dee2e6;
    background-color: white;
    border-radius: 5px;
}

.arrow {
    font-size: 2em;
    margin: 0 0.8em;
    color: #667eea;
    font-weight: bold;
}

/* Status Indicators */
.status-indicator {
    font-weight: 600;
    padding: 0.4em 0.8em;
    border-radius: 5px;
    display: inline-block;
    margin-left: 0.5em;
    font-size: 0.9em;
}

.valid {
    color: #155724;
    background-color: #d4edda;
    border: 1px solid #c3e6cb;
}

.invalid {
    color: #721c24;
    background-color: #f8d7da;
    border: 1px solid #f5c6cb;
}

/* Substitution Styles */
.frame-style {
    color: #495057;
    background-color: #e9ecef;
    padding: 0.2em 0.5em;
    border: 1px solid #ced4da;
    border-radius: 4px;
```

```
    font-family: monospace;
}

.substituted-for-style {
    color: #667eea;
    font-weight: 600;
    border-bottom: 2px solid #667eea;
}

.substituting-style {
    color: #28a745;
    font-weight: 600;
    border-bottom: 2px solid #28a745;
}

.expression-box {
    display: inline-block;
    padding: 0.4em 0.8em;
    border: 2px solid #dee2e6;
    background-color: #fff;
    border-radius: 6px;
    margin: 0 0.5em;
    min-width: 100px;
    text-align: center;
    transition: transform 0.3s ease-out, opacity 0.3s ease-out, box-shadow 0.3s ease-out;
}

.sentence-display .substituted-part {
    display: inline-block;
    background-color: #e7f5ff;
    padding: 0 5px;
    border-radius: 3px;
    border: 1px dashed #99cfff;
}

.sentence-display .substituting-part {
    display: inline-block;
    background-color: #e6ffe6;
    padding: 0 5px;
    border-radius: 3px;
    border: 1px dashed #a3e9a4;
    opacity: 0;
    transition: opacity 0.3s ease-out;
}

/* Animation Classes - FIXED */
.highlight-replace {
    box-shadow: 0 0 12px 3px rgba(102, 126, 234, 0.6);
}

.lift-out {
    transform: translateY(-10px) scale(1.05);
    opacity: 0.7;
}

.lift-out-anim {
    animation: liftOut 0.4s ease-out;
}

.fade-out {
    opacity: 0 !important;
    transform: translateY(-20px) scale(0.9);
}

.fade-out-anim {
```

```
    animation: fadeOut 0.4s ease-out forwards;
}

.move-in {
    opacity: 1 !important;
    transform: translateY(0) scale(1);
}

.move-in-anim {
    animation: moveIn 0.5s ease-out forwards;
}

.highlight-incoming {
    animation: highlightPulse 0.6s ease-in-out;
}

/* Keyframe Animations */
@keyframes liftOut {
    0% {
        transform: translateY(0) scale(1);
        opacity: 1;
    }
    100% {
        transform: translateY(-10px) scale(1.05);
        opacity: 0.7;
    }
}

@keyframes fadeOut {
    0% {
        opacity: 1;
        transform: translateY(0) scale(1);
    }
    100% {
        opacity: 0;
        transform: translateY(-20px) scale(0.9);
    }
}

@keyframes moveIn {
    0% {
        opacity: 0;
        transform: translateY(10px) scale(0.95);
    }
    100% {
        opacity: 1;
        transform: translateY(0) scale(1);
    }
}

@keyframes highlightPulse {
    0%, 100% {
        box-shadow: 0 0 0 rgba(40, 167, 69, 0);
    }
    50% {
        box-shadow: 0 0 15px 5px rgba(40, 167, 69, 0.6);
    }
}

/* Matrix Table */
.matrix-table {
    border-collapse: collapse;
    width: 100%;
    margin: 1.5em 0;
    font-size: 0.9em;
```

```
}

.matrix-table th, .matrix-table td {
    border: 1px solid #dee2e6;
    padding: 12px;
    text-align: left;
    vertical-align: top;
}

.matrix-table th {
    background-color: #e9ecef;
    color: #495057;
    font-weight: 600;
}

.matrix-table td ul {
    padding-left: 1.2em;
    margin-top: 0.5em;
}

/* Info Icons & Tooltips */
.info-icon {
    display: inline-block;
    width: 20px;
    height: 20px;
    background-color: #667eea;
    color: white;
    border-radius: 50%;
    text-align: center;
    font-size: 13px;
    line-height: 20px;
    cursor: help;
    margin-left: 5px;
    font-weight: bold;
    vertical-align: middle;
    position: relative;
}

.tooltip-text {
    visibility: hidden;
    width: 240px;
    background-color: #343a40;
    color: #fff;
    text-align: left;
    border-radius: 8px;
    padding: 10px;
    position: absolute;
    z-index: 1;
    bottom: 130%;
    left: 50%;
    margin-left: -120px;
    opacity: 0;
    transition: opacity 0.3s;
    font-size: 0.9em;
    font-weight: normal;
    box-shadow: 0 4px 10px rgba(0,0,0,0.3);
}

.info-icon:hover .tooltip-text {
    visibility: visible;
    opacity: 1;
}

/* Footer */
footer {
```

```
    text-align: center;
    padding: 2em;
    background-color: #f8f9fa;
    color: #6c757d;
    margin-top: 3em;
    border-top: 2px solid #dee2e6;
}

footer p {
    margin: 0;
    font-size: 0.9em;
}

/* Fix for SVG label cutoff */
.shape-viz svg {
    overflow: visible !important;
}

.shape-item {
    display: inline-block;
    margin: 10px;
    text-align: center;
}

```