Supplementary Information for: Spectral Sheaf Heuristics for Consistency Detection in Multi-Agent Systems

Sepehr Bayat
Hooshex AI Lab
sepehrbayat@hooshex.com

February 8, 2026

Contents

1 Practical Guide to Sheaf Construction **3**
1.1 General Principles . . . 3
1.1.1 Step 1: Identify the Belief Graph . . . 3
1.1.2 Step 2: Define the Stalks (Local Data) . . . 4
1.1.3 Step 3: Define Restriction Maps (Consistency Constraints) . . . 4
1.1.4 Step 4: Construct the Connection Laplacian . . . 5
1.1.5 Step 5: Validate the Sheaf . . . 5
1.2 Walk-Through 1: Logic Maze . . . 6
1.2.1 Problem Description . . . 6
1.2.2 Step 1: Belief Graph . . . 6
1.2.3 Step 2: Stalks . . . 6
1.2.4 Step 3: Restriction Maps . . . 6
1.2.5 Step 4: Injecting a Contradiction . . . 7
1.2.6 Implementation Code Snippet . . . 7
1.3 Walk-Through 2: Safety Gym (Safe RL) . . . 8
1.3.1 Problem Description . . . 8
1.3.2 Step 1: Belief Graph . . . 8
1.3.3 Step 2: Stalks . . . 8
1.3.4 Step 3: Restriction Maps (Bellman Consistency) . . . 8
1.3.5 Step 4: Detecting Safety Violations . . . 9
1.3.6 Implementation Code Snippet . . . 9
1.4 Walk-Through 3: Multi-Robot Coordination . . . 10
1.4.1 Problem Description . . . 10
1.4.2 Step 1: Belief Graph . . . 10
1.4.3 Step 2: Stalks . . . 10
1.4.4 Step 3: Restriction Maps (Spatial Consistency) . . . 10
1.4.5 Step 4: Injecting a Contradiction . . . 11
1.4.6 Step 5: Resolution via Negotiation . . . 11
1.4.7 Implementation Code Snippet . . . 12
1.5 Summary: Decision Tree for Sheaf Construction . . . 12

Theorem 1 (Spectral-Cohomological Correspondence). *Let $\mathcal{F}$ be a cellular sheaf on a connected graph $G = (V, E)$ with $N = |V|$ vertices. Let $\mathcal{L} \in \mathbb{R}^{Nd \times Nd}$ be the Connection Laplacian, where d is the stalk dimension. Then:*

$$h^1_{true} = \dim(\ker(\mathcal{L})) - 1 \tag{1}$$

where $\dim(\ker(\mathcal{L}))$ is the multiplicity of the zero eigenvalue of $\mathcal{L}$, and $h^1_{true} = \dim(H^1(\mathcal{F}))$ is the true first cohomology dimension.

Proof Sketch. By the fundamental theorem of sheaf cohomology [**?**], the kernel of $\mathcal{L}$ decomposes as $\ker(\mathcal{L}) \cong H^0(\mathcal{F}) \oplus H^1(\mathcal{F})$. Since G is connected, $\dim(H^0(\mathcal{F})) = 1$ (constant global sections). Thus $\dim(\ker(\mathcal{L})) = 1 + h^1_{\text{true}}$. Rearranging gives the result. See Appendix **??** for the full proof. □ □

Theorem 2 (Error Bound for Spectral Approximation). *Let $\mathcal{L}_0 \in \mathbb{R}^{Nd \times Nd}$ be the ideal Connection Laplacian for a graph with N vertices and stalk dimension d. Let $\mathcal{L} = \mathcal{L}_0 + E$ be a perturbed version with $\|E\|_2 \leq \sigma$, where $\sigma > 0$ is the noise level and $\|\cdot\|_2$ denotes the spectral norm (largest singular value).*

*Let $\delta > 0$ be the **spectral gap**: the minimum distance between distinct eigenvalues of $\mathcal{L}_0$. Assume:*

1. *$\sigma < \delta/4$ (noise is small compared to spectral gap)*
2. *$\epsilon = \delta/2$ (threshold is half the spectral gap)*

Then the error in the spectral approximation of h^1 is bounded by:

$$|h^1_\epsilon(\mathcal{L}) - h^1_{true}(\mathcal{L}_0)| \leq \left\lceil \frac{2\sigma}{\delta} \right\rceil \tag{2}$$

where $h^1_\epsilon(\mathcal{L}) = \#\{i : \lambda_i(\mathcal{L}) < \epsilon\} - 1$ is the spectral approximation.

Proof Sketch. By Weyl's inequality, each eigenvalue of $\mathcal{L}$ is within σ of the corresponding eigenvalue of $\mathcal{L}_0$. The ideal zero block (from $H^0 \oplus H^1$) has $1 + h^1_{\text{true}}$ eigenvalues, all exactly zero. Under perturbation, these shift to $[-\sigma, \sigma]$. The positive block starts at δ and shifts to $[\delta - \sigma, \infty)$. With $\epsilon = \delta/2$ and $\sigma < \delta/4$, the threshold cleanly separates the two blocks, with at most $\lceil 2\sigma/\delta \rceil$ eigenvalues potentially crossing the threshold. See Appendix **??** for the detailed proof with all steps. □ □

Theorem 3 (Computational Complexity). *Consider a graph $G = (V, E)$ with $N = |V|$ vertices, $M = |E|$ edges, and $M = O(N)$ (sparse graph). Let d be the stalk dimension, assumed to be a small constant (e.g., $d \in \{2, 4, 8\}$).*

*The STPGC algorithm (Algorithm **??**) computes the Phronesis Index Φ in:*

$$O(Nd \log(Nd)) \text{ time} \tag{3}$$

Proof Sketch. The algorithm has three steps: (1) Construct $\mathcal{L}$: $O(Nd^2)$ time (iterating over edges and filling sparse matrix). (2) Compute $k = 20$ smallest eigenvalues via Lanczos: $O(Nd \cdot k)$ per iteration, $O(\log(Nd))$ iterations to converge, total $O(Nd \log(Nd))$. (3) Compute Φ: $O(k) = O(1)$. Since d is constant and $M = O(N)$, the dominant term is $O(Nd \log(Nd))$. See Appendix **??** for the full complexity analysis. □ □

Remark 4 (Conditions for Theorem 2). *The condition $\sigma < \delta/4$ is necessary for the error bound to hold. If noise exceeds this level, eigenvalues from the positive block can be pushed below ϵ, causing h^1_ϵ to overestimate h^1_{true} arbitrarily. In practice, this means:*

- *For systems with small spectral gap ($\delta < 0.01$), the method is sensitive to noise.*
- *For systems with large spectral gap ($\delta > 0.1$), the method is robust to moderate noise ($\sigma \approx 0.02$).*
- *If $\sigma \geq \delta/4$, use noise filtering (Appendix **??**) or increase ϵ adaptively.*

Remark 5 (Conditions for Theorem 3). *The $O(Nd \log(Nd))$ complexity assumes:*

1. ***Sparse graph:*** *$M = O(N)$. For dense graphs ($M = O(N^2)$), Laplacian construction becomes $O(N^2 d^2)$, dominating the eigenvalue computation.*
2. ***Small stalk dimension:*** *$d = O(1)$. For large d (e.g., $d = 100$), the d^2 factor in Laplacian construction may dominate.*
3. ***Well-conditioned Laplacian:*** *Lanczos converges in $O(\log(Nd))$ iterations. For ill-conditioned matrices (e.g., very small spectral gap), convergence may be slower.*

In our experiments, these conditions hold: graphs have $M \approx 4N$ (grid-like), stalks have $d \in \{2, 4\}$, and spectral gaps are $\delta > 0.01$.

1 Practical Guide to Sheaf Construction

This section provides step-by-step guidance for constructing cellular sheaves for multi-agent consistency monitoring. We present general principles followed by detailed walk-throughs of our three experimental scenarios.

1.1 General Principles

1.1.1 Step 1: Identify the Belief Graph

Question: What is the underlying network structure?
Action: Define the graph $G = (V, E)$ where:

- **Vertices V:** Represent *belief states* or *knowledge locations*
  - In multi-agent systems: vertices = agents or local knowledge bases
  - In state-space problems: vertices = states or situations
  - In sensor networks: vertices = sensor nodes or spatial locations
- **Edges E:** Represent *consistency relationships* or *information flow*
  - Connect vertices that share information or have overlapping beliefs
  - In physical systems: edges = communication links or spatial adjacency
  - In logical systems: edges = inference steps or constraint relationships

Example: In a team of 3 robots exploring an environment:

- Vertices: Robot 1, Robot 2, Robot 3
- Edges: (Robot 1, Robot 2), (Robot 2, Robot 3), (Robot 3, Robot 1) if they communicate in a triangle topology

1.1.2 Step 2: Define the Stalks (Local Data)

Question: What information does each vertex hold?

Action: For each vertex $v \in V$, define the stalk $\mathcal{F}(v)$ as a vector space representing the local belief or data:

- **Dimension d:** Depends on the type of information
  - Scalar beliefs: $d = 1$ (e.g., temperature readings)
  - Vector beliefs: $d = 2, 3, \ldots$ (e.g., 2D positions, RGB colors, Q-values for multiple actions)
  - Structured beliefs: $d =$ size of feature vector
- **Interpretation:** Each element of $\mathcal{F}(v) \cong \mathbb{R}^d$ represents a possible belief state at vertex v

Example: For robots tracking a target:

- $\mathcal{F}(\text{Robot 1}) = \mathbb{R}^2$: Robot 1's belief about target position (x, y)
- Similarly for Robot 2 and Robot 3

1.1.3 Step 3: Define Restriction Maps (Consistency Constraints)

Question: What does it mean for two neighboring vertices to be "consistent"?

Action: For each edge $e = (u, v) \in E$, define restriction maps $r_{e,u} : \mathcal{F}(u) \to \mathcal{F}(e)$ and $r_{e,v} : \mathcal{F}(v) \to \mathcal{F}(e)$ that encode the consistency requirement.

Common Patterns:

Pattern A: Identity Consistency (Agreement) If u and v should hold identical beliefs:

$$r_{e,u} = r_{e,v} = I \quad \text{(identity map)} \tag{4}$$

Then consistency means $x_u = x_v$.

Pattern B: Linear Transformation Consistency If beliefs are related by a known transformation T:

$$r_{e,u} = I, \quad r_{e,v} = T \tag{5}$$

Then consistency means $x_u = T(x_v)$.

Example: Robots with different coordinate frames:

- Robot 1 uses global coordinates
- Robot 2 uses local coordinates (rotated by θ)
- Restriction map: $r_{e,\text{Robot 2}} = R_\theta$ (rotation matrix)
- Consistency: Robot 1's global position $= R_\theta \times$ Robot 2's local position

Pattern C: Projection or Aggregation If one vertex has more information than another:

$$r_{e,u} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_e}, \quad r_{e,v} : \mathbb{R}^{d_v} \to \mathbb{R}^{d_e} \tag{6}$$

where $d_e \leq \min(d_u, d_v)$.

Example: Sensor fusion:

- Sensor A measures (x, y, z) position
- Sensor B measures (x, y) position (no depth)
- Edge stalk: $\mathcal{F}(e) = \mathbb{R}^2$
- $r_{e,A}$: project $(x, y, z) \mapsto (x, y)$
- $r_{e,B}$: identity $(x, y) \mapsto (x, y)$
- Consistency: Sensor A's (x, y) projection matches Sensor B's (x, y)

1.1.4 Step 4: Construct the Connection Laplacian

Action: Build the matrix $\mathcal{L}$ of size $Nd \times Nd$ where $N = |V|$ and d is the stalk dimension.

Block Structure:

$$\mathcal{L} = D - A \tag{7}$$

where:

- D: Block-diagonal degree matrix

$$D_{vv} = \deg(v) \cdot I_d \tag{8}$$

- A: Block adjacency matrix with restriction maps

$$A_{uv} = r_{e,v}^T r_{e,u} \quad \text{if } e = (u, v) \in E \tag{9}$$

Intuition: The Laplacian measures "how much" local beliefs disagree when propagated through restriction maps.

1.1.5 Step 5: Validate the Sheaf

Checklist:

1. **Regularity:** Are restriction maps linear? (Required for spectral methods)
2. **Symmetry:** Is the graph undirected? (Ensures $\mathcal{L}$ is symmetric)
3. **Connectivity:** Is the graph connected? (Ensures $h^0 = 1$)
4. **Consistency Encoding:** Do the restriction maps truly capture the intended consistency relationships?

Test: Create a small example (3-5 vertices) and manually verify:

- A fully consistent global section (all constraints satisfied) should give $\mathcal{L}x = 0$
- An inconsistent configuration should give $\mathcal{L}x \neq 0$

1.2 Walk-Through 1: Logic Maze

1.2.1 Problem Description

An agent navigates a 5×5 grid maze. At each cell, the agent has a belief about its orientation (one of 4 directions: North, East, South, West). The agent receives local observations (e.g., "wall on left") that constrain relative orientations between adjacent cells. A contradiction occurs if the constraints form an inconsistent cycle.

1.2.2 Step 1: Belief Graph

- **Vertices:** 25 cells in the 5×5 grid
- **Edges:** Connect adjacent cells (up/down/left/right neighbors)
- **Graph structure:** 2D grid graph (40 edges for a 5×5 grid)

1.2.3 Step 2: Stalks

Representation of Orientation:
We use $SO(2)$ (2D rotation group) to represent orientations. Each orientation is a unit vector in $\mathbb{R}^2$:

- North: $(0, 1)$
- East: $(1, 0)$
- South: $(0, -1)$
- West: $(-1, 0)$

Stalk Definition:

$$\mathcal{F}(\text{cell}) = \mathbb{R}^2 \tag{10}$$

Each cell's stalk holds a 2D vector representing the agent's believed orientation at that cell.

1.2.4 Step 3: Restriction Maps

Consistency Requirement:
If the agent moves from cell u to cell v, its orientation should transform according to the observed turn:

- No turn: orientations should match
- Left turn: orientation rotates 90° counterclockwise
- Right turn: orientation rotates 90° clockwise

Restriction Map:
For edge $e = (u, v)$ with observed turn θ_e:

$$r_{e,u} = I, \quad r_{e,v} = R_{\theta_e} \tag{11}$$

where R_θ is the 2D rotation matrix:

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{12}$$

Example:

- Agent at cell u faces North: $x_u = (0, 1)$
- Agent moves to cell v with a left turn ($\theta = 90°$)
- Consistency: $x_u = R_{90°} x_v$
- If $x_v = (1, 0)$ (facing East), then $R_{90°} x_v = (0, 1)$ (consistent)

1.2.5 Step 4: Injecting a Contradiction

Scenario: At time $t = 50$, we inject a false observation at the center cell (cell 12):

- The agent suddenly believes it faces South instead of North
- This creates a cycle of inconsistent orientations around cell 12

Effect on Sheaf:

- Before injection: $h^1 = 0$ (all orientations consistent)
- After injection: $h^1 = 1$ (one independent cycle of contradiction)
- λ_1 decreases (frustration in the system)
- Φ drops sharply (detected as anomaly)

1.2.6 Implementation Code Snippet

```
def construct_logic_maze_sheaf(grid_size=5):
    # Step 1: Create grid graph
    G = nx.grid_2d_graph(grid_size, grid_size)
    G = nx.convert_node_labels_to_integers(G)

    # Step 2: Define stalks (SO(2) orientations)
    stalk_dim = 2
    stalks = {v: np.random.randn(stalk_dim) for v in G.nodes()}
    # Normalize to unit vectors
    for v in stalks:
        stalks[v] /= np.linalg.norm(stalks[v])

    # Step 3: Define restriction maps (rotation matrices)
    restriction_maps = {}
    for edge in G.edges():
        u, v = edge
        # Random turn angle (or from observations)
        theta = np.random.choice([0, np.pi/2, -np.pi/2])
        R = np.array([[np.cos(theta), -np.sin(theta)],
                      [np.sin(theta),  np.cos(theta)]])
        restriction_maps[edge] = R

    # Step 4: Construct Connection Laplacian
    L = construct_connection_laplacian(G, stalks, restriction_maps)

    return G, stalks, restriction_maps, L

def inject_contradiction(stalks, center_node=12):
    # Flip orientation at center node
    stalks[center_node] = -stalks[center_node]
```

1.3 Walk-Through 2: Safety Gym (Safe RL)

1.3.1 Problem Description

An RL agent learns to navigate a 2D environment (Safety Gym) while avoiding hazards. The agent maintains a belief graph where vertices represent states and edges represent state transitions. Each state has an associated Q-value vector (expected rewards for each action). Bellman consistency requires that Q-values satisfy the Bellman equation across transitions.

1.3.2 Step 1: Belief Graph

Discretization:

- Continuous state space $(x, y) \in [0, 10] \times [0, 10]$ discretized into a 10×10 grid
- **Vertices:** 100 grid cells (states)
- **Edges:** Connect states reachable by one action (4-connected grid)

Dynamic Graph:

- Edges are added as the agent explores and experiences transitions
- Initially sparse, becomes denser with training

1.3.3 Step 2: Stalks

Q-Values:

Each state has Q-values for 4 actions: {up, down, left, right}

Stalk Definition:

$$\mathcal{F}(\text{state}) = \mathbb{R}^4 \tag{13}$$

Element i of the stalk represents $Q(s, a_i)$, the expected return for taking action a_i in state s.

1.3.4 Step 3: Restriction Maps (Bellman Consistency)

Bellman Equation:

For a transition from state s to state s' via action a with reward r:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{14}$$

Consistency Requirement:

The Q-value at s for action a should equal the discounted max Q-value at s' plus the reward.

Restriction Map Design:

For edge $e = (s, s')$ corresponding to action a:

- Edge stalk: $\mathcal{F}(e) = \mathbb{R}$ (scalar, the Q-value for that transition)
- $r_{e,s} : \mathbb{R}^4 \to \mathbb{R}$: Extract $Q(s, a)$ (select component a)
- $r_{e,s'} : \mathbb{R}^4 \to \mathbb{R}$: Compute $r + \gamma \max_{a'} Q(s', a')$

Mathematically:

Let e_a be the unit vector selecting action a:

$$r_{e,s}(Q_s) = e_a^T Q_s \tag{15}$$

$$r_{e,s'}(Q_{s'}) = r + \gamma \max(Q_{s'}) \tag{16}$$

Consistency: $r_{e,s}(Q_s) = r_{e,s'}(Q_{s'})$ means the Bellman equation is satisfied.

1.3.5 Step 4: Detecting Safety Violations

Mechanism:

- During training, the agent updates Q-values based on experience
- If the agent encounters a hazard (safety violation), it receives a large negative reward
- This creates a sudden inconsistency: the Q-value for the action leading to the hazard should drop, but neighboring Q-values may not have updated yet
- This inconsistency manifests as increased h^1 (a cycle of Bellman violations)
- Φ drops, signaling danger

Using Φ as Auxiliary Reward:
We modify the reward function:

$$r'(s, a) = r(s, a) + \alpha \cdot \Phi(s) \tag{17}$$

where $\alpha = 0.1$ is a scaling factor.

Effect:

- High Φ (consistent beliefs): normal reward
- Low Φ (inconsistent beliefs): penalty, discouraging risky actions
- Agent learns to avoid states where its Q-value estimates are internally contradictory (often near hazards)

1.3.6 Implementation Code Snippet

```
def construct_safety_gym_sheaf(state_graph, q_values, gamma=0.99):
    G = state_graph
    stalk_dim = 4  # 4 actions

    # Step 2: Stalks are Q-value vectors
    stalks = {s: q_values[s] for s in G.nodes()}

    # Step 3: Restriction maps (Bellman consistency)
    restriction_maps = {}
    for edge in G.edges():
        s, s_prime = edge
        action = edge_to_action(edge)  # Determine which action
        reward = get_reward(s, action, s_prime)

        # r_{e,s}: select Q(s, action)
        r_e_s = np.zeros((1, 4))
        r_e_s[0, action] = 1.0

        # r_{e,s'}: compute r + gamma * max Q(s')
        r_e_s_prime = lambda Q: reward + gamma * np.max(Q)

        # For linear Laplacian, approximate max as weighted sum
        # (or use linearization around current Q-values)
```

```
        restriction_maps[edge] = (r_e_s, r_e_s_prime)

    L = construct_connection_laplacian(G, stalks, restriction_maps)
    return L

def compute_phronesis_reward(state, L, epsilon=1e-3):
    Phi, h1, lambda1 = compute_phronesis_index(L, epsilon)
    return Phi  # Higher Phi = more consistent = safer
```

1.4 Walk-Through 3: Multi-Robot Coordination

1.4.1 Problem Description

Three robots explore an environment and share observations about a target's location. Each robot has its own belief about the target position (x, y). Robots communicate over a network (triangle topology). A contradiction arises when Robot 1 believes the target is at $(2, 3)$, Robot 2 believes it's at $(7, 8)$, and Robot 3 has yet another belief, creating a cycle of incompatible observations.

1.4.2 Step 1: Belief Graph

Two-Level Structure:

- **Local Level:** Each robot has its own 10×10 grid belief graph (100 vertices per robot)
- **Global Level:** Robots are connected via a communication graph (3 vertices, 3 edges forming a triangle)

Combined Graph:

- Total vertices: $3 \times 100 = 300$ (local states) $+3$ (robot nodes) $= 303$
- Or, simplified: 3 robot nodes with aggregated beliefs

For our experiments, we use the simplified version:

- **Vertices:** Robot 1, Robot 2, Robot 3
- **Edges:** (R1, R2), (R2, R3), (R3, R1)

1.4.3 Step 2: Stalks

Target Position Belief:
Each robot's stalk represents its belief about the target's 2D position.
Stalk Definition:

$$\mathcal{F}(\text{Robot } i) = \mathbb{R}^2 \tag{18}$$

Element $(x, y) \in \mathbb{R}^2$ represents the robot's belief about where the target is located.

1.4.4 Step 3: Restriction Maps (Spatial Consistency)

Consistency Requirement:
If two robots communicate, their beliefs about the target location should be compatible (within sensor error).
Restriction Map:

For edge $e = (\text{Robot } i, \text{Robot } j)$:

$$r_{e,i} = r_{e,j} = I \quad \text{(identity)} \tag{19}$$

Interpretation: Robots should agree on the target position. Consistency means $x_i \approx x_j$.
Alternative (with coordinate transforms):
If robots use different coordinate frames:

$$r_{e,i} = I, \quad r_{e,j} = T_{ij} \tag{20}$$

where T_{ij} is the coordinate transformation from Robot j's frame to Robot i's frame.

1.4.5 Step 4: Injecting a Contradiction

Scenario:

- Robot 1 observes target at $(2, 3)$ in global coordinates
- Robot 2 observes target at $(7, 8)$ in global coordinates
- Robot 3 observes target at $(5, 5)$ (average, but still inconsistent with both)

Effect on Sheaf:

- The three beliefs form a triangle of disagreements
- No global section exists that satisfies all pairwise consistency constraints
- $h^1 = 1$ (one independent cycle of contradiction)
- Φ drops significantly

1.4.6 Step 5: Resolution via Negotiation

Strategy:
Robots detect low Φ and initiate a negotiation protocol:

1. Compute the average belief: $\bar{x} = \frac{1}{3}(x_1 + x_2 + x_3) = (4.67, 5.33)$
2. Each robot updates its belief to $\bar{x} +$ small noise
3. Re-compute Φ

Result:

- After averaging: $h^1 = 0$ (no contradiction)
- Φ increases (consistency restored)
- Coordination success improves

1.4.7 Implementation Code Snippet

```
def construct_multi_robot_sheaf(num_robots=3):
    # Step 1: Communication graph (triangle)
    G = nx.cycle_graph(num_robots)

    # Step 2: Stalks are 2D target position beliefs
    stalk_dim = 2
    stalks = {i: np.random.randn(stalk_dim) * 5 for i in G.nodes()}

    # Step 3: Restriction maps (identity for agreement)
    restriction_maps = {}
    for edge in G.edges():
        restriction_maps[edge] = np.eye(stalk_dim)

    L = construct_connection_laplacian(G, stalks, restriction_maps)
    return G, stalks, L

def inject_contradiction(stalks):
    stalks[0] = np.array([2.0, 3.0])  # Robot 1 belief
    stalks[1] = np.array([7.0, 8.0])  # Robot 2 belief
    stalks[2] = np.array([5.0, 5.0])  # Robot 3 belief

def resolve_via_negotiation(stalks):
    avg = np.mean([stalks[i] for i in stalks], axis=0)
    for i in stalks:
        stalks[i] = avg + np.random.randn(2) * 0.5  # Small noise
```

1.5 Summary: Decision Tree for Sheaf Construction

```
START: What kind of multi-agent system do you have?

+-- Logical/Symbolic Knowledge
|   +-- Stalks: Propositional assignments or logical states
|   +-- Restriction: Logical consistency (e.g., not(A and not A))
|   +-- Example: Logic Maze (orientation consistency)
|
+-- Numerical/Continuous Beliefs
|   +-- Stalks: Real-valued vectors (positions, Q-values, etc.)
|   +-- Restriction: Equality or linear transformation
|   +-- Example: Multi-Robot (spatial consistency)
|
+-- Dynamical Systems (RL/Control)
|   +-- Stalks: State-action values or policy parameters
|   +-- Restriction: Bellman equation or dynamics model
|   +-- Example: Safety Gym (Q-value consistency)
|
+-- Hierarchical/Nested Systems
    +-- Stalks: Multi-level representations
    +-- Restriction: Aggregation or projection maps
    +-- Example: Federated learning (local vs global models)
```

Key Takeaway: The sheaf construction is domain-specific but follows a systematic process.

Start with the graph structure, identify what "consistency" means in your domain, and encode it as restriction maps. Validate with small examples before scaling up.