Aquí tienes una especificación técnica detallada en formato Markdown, diseñada para ser entregada directamente a un equipo de desarrollo o implementada en un proyecto de software serio.

Esta especificación se centra en la **Lógica Epistémica Dinámica (DEL)** y la **Verificación de Modelos (Model Checking)**, que son las formalizaciones técnicas de tu idea de "mover lógica a través de los universos".

Especificación Técnica: Módulos Avanzados para JUDIT (v2.0)

Versión del Documento: 1.0

Contexto: Extensión del motor de inferencia y visualización para Lógica Modal/Epistémica.

1. Resumen Ejecutivo

El objetivo es implementar capacidades dinámicas en el motor de JUDIT. Actualmente, el sistema es *estático* (evalúa una fórmula en un grafo fijo). La actualización permitirá que **las fórmulas transformen la estructura del grafo** (Mundos y Relaciones), simulando eventos, acciones y consecuencias contrafácticas.

---

2. Módulo de Lógica Epistémica Dinámica (DEL) - "Action Models"

Este módulo permite la "actualización del producto" (Product Update), que es el estándar matemático para modelar cambios de información.

2.1. Estructura de Datos: Modelo de Acción (M_a)

Se debe definir una nueva estructura de datos paralela al Modelo de Kripke (M).

Definición de Clase (Pseudocódigo/TypeScript interface):

```
TypeScript
```

```
interface ActionEvent {
  id: string;
  precondition: Formula; // Fórmula que debe ser verdadera en el mundo origen para que este evento ocurra
}

interface ActionModel {
  events: ActionEvent[];
  relations: Map<string, Array<{from: string, to: string}>>; // Relaciones de accesibilidad entre eventos (quién distingue qué evento)
```

```
  // Ejemplo: Si el Agente A no distingue el evento e1 del e2, hay una arista e1 -> e2 para A.
}
```

2.2. Algoritmo de Ejecución (Product Update)

La operación `ExecuteAction(CurrentModel M, ActionModel A) -> NewModel M'`

Esta función calcula el "producto cruzado" entre el grafo de mundos y el grafo de eventos.

Lógica del Algoritmo:

1. **Generación de Nuevos Mundos (W'):**
    - Iterar sobre cada par (w, e) donde $w \in M.W$ y $e \in A.Events$.
    - **Condición de Supervivencia:** El par (w, e) se convierte en un nuevo mundo w' SI Y SOLO SI $M, w \models e.precondition$ (la precondición del evento se cumple en el mundo original).
    - El nuevo mundo hereda la valuación (verdades p, q, r) del mundo original w.
2. **Generación de Nuevas Relaciones (R'):**
    - Para dos nuevos mundos (w, e) y (w', e'), y un agente ag:
    - Existe una relación R'_{ag} entre ellos SI Y SOLO SI:
        1. Existía relación en el modelo original: $(w, w') \in R_{ag}$
        2. Existe relación en el modelo de acción: $(e, e') \in R_{ag}^{action}$

Visualización en Frontend:

- **Transición:** Al ejecutar, animar la "explosión" de mundos. Los mundos originales que no cumplen ninguna precondición se desvanecen. Los que cumplen múltiples precondiciones se duplican.

---

3. Módulo de Lógica Condicional y Contrafácticos (Semántica de Lewis)

Permite evaluar "¿Qué pasaría si...?" buscando los mundos más "cercanos".

3.1. Métrica de Distancia (Similaridad)

Se necesita un algoritmo para cuantificar qué tan "lejos" está un mundo de otro.

Implementación sugerida (Distancia de Hamming ponderada):

- Dadas dos valuaciones $V(w_1)$ y $V(w_2)$.
- $Distance(w_1, w_2) = \sum |V(w_1, p_i) - V(w_2, p_i)|$
- *Optimización:* Permitir al usuario definir pesos para las variables (ej. cambiar la variable `nuclear_war` cuesta 100 puntos de distancia, cambiar `raining` cuesta 1).

3.2. Operador Contrafáctico ($>$)

Implementar el operador binario `Counterfactual(φ, ψ)` (Si ϕ fuera verdad, ψ sería verdad).

Algoritmo de Evaluación:

1. Sea w_{actual} el mundo seleccionado.
2. Identificar el conjunto $S_\phi = \{ w \in W \mid M, w \models \phi \}$ (todos los mundos donde la premisa es cierta).
3. Si S_ϕ está vacío, el contrafáctico es vacíamente verdadero (o falso, según la semántica elegida).
4. Seleccionar el subconjunto $Min_\phi \subseteq S_\phi$ tal que la distancia $Distance(w_{actual}, w)$ sea mínima.
5. **Resultado:** `TRUE` si $M, w \models \psi$ para *todo* $w \in Min_\phi$.

---

4. Módulo de Verificación de Modelos (Model Checking & Trace)

Herramientas para validar propiedades a lo largo del tiempo o caminos.

4.1. Verificador de Invariantes

Permitir definir una fórmula "Global" que debe cumplirse siempre.

Función `CheckInvariant(Formula φ)`:

- Ejecuta una búsqueda (BFS/DFS) desde los mundos iniciales.
- Verifica $M, w \models \phi$ en cada nodo visitado.
- **Salida:** Si falla, devolver la **Traza de Error**: una lista de mundos $(w_0 \rightarrow w_1 \rightarrow ... \rightarrow w_{error})$ que demuestra cómo se llega al estado ilegal.

4.2. Generador de Contraejemplos (SAT Solver Integration)

Integrar una librería externa (como Z3 compilado a WebAssembly o un solver JS ligero como `logic-solver`).

- **Input:** Una fórmula que el usuario afirma que es una Tautología (siempre verdadera).
- **Proceso:** Negar la fórmula ($\neg \phi$) y pasarla al SAT Solver.
- **Output:** Si el SAT Solver encuentra solución, significa que la fórmula original NO es una tautología.
- **Acción:** JUDIT debe leer la solución del SAT (asignación de variables) y **construir automáticamente un grafo** mínimo que represente esa solución para mostrárselo al usuario.

---

5. Especificaciones de UI/UX para Funcionalidades Nuevas

5.1. Panel de "Línea de Tiempo" (Time Travel)

- **Ubicación:** Parte inferior de la pantalla (dockable).
- **Componentes:**
    - Slider horizontal representando estados discretos ($t_0, t_1, t_2...$).
    - Cada vez que se aplica un "Update" (ver sección 2), se genera un nuevo estado t_{n+1}.
    - Permite hacer clic en t_{n-1} para revertir el grafo al estado anterior (Snapshot pattern).

5.2. Editor de Relaciones Multi-Agente

- **Interacción:**
    - Al hacer Shift + Drag para crear una relación, abrir un pequeño menú contextual flotante (pop-over).
    - **Checkboxes:** [ ] Agente a, [ ] Agente b, [ ] Agente c.
    - **Visualización:** Las líneas deben ser coloreadas o punteadas según el agente (ej. a=sólida roja, b=punteada azul).
    - Si múltiples agentes comparten la misma relación, usar líneas paralelas o multicolor.

---

6. Stack Tecnológico Recomendado

Dado que es una aplicación web (parece React o similar):

- **Estructuras de Grafos:** Graphology o Cytoscape.js (optimizados para análisis, no solo renderizado).
- **Renderizado:** D3.js (para control total de la física de nodos) o React Flow (si se prefiere componentes listos).
- **Motor Lógico:** Implementar un **Visitor Pattern** sobre el Árbol de Sintaxis Abstracta (AST) de las fórmulas. Esto permite añadir nuevos operadores (como el contrafáctico) sin reescribir todo el evaluador.

Ejemplo de flujo de trabajo con estas mejoras:

1. El usuario diseña un protocolo de seguridad (Grafo M_0).
2. Define una acción "Ataque Hacker" (Modelo de Acción M_a).
3. Ejecuta la acción. JUDIT genera M_1 (el estado post-ataque).
4. El usuario pregunta: "En este nuevo estado, ¿Sabe el sistema que ha sido hackeado?" ($K_{sys} hackeado$).
5. El sistema evalúa y muestra la ruta de acceso visual.