**Informaticae Abstracta**. ISSN: 3061-8533
Año 2025, Vol. 3, No. 2, pp. 25-43
JUL | DIC 2025
https://doi.org/10.36677/ia.v3n2.27474

# De la Filosofía del Derecho a la Computación: Un Enfoque Computacional Multi-Agente Inspirado en la Teoría Pura de Kelsen

Vidal-López, R.

**Información del Artículo**

Recibido: 12 de septiembre, 2025
Aceptado: 15 de noviembre, 2025

**Palabras clave:** Sistemas multi-agente, Arquitectura BDI, Razonamiento jurídico, Teoría Pura del Derecho, Lógica híbrida intuicionista

**Resumen**

La resolución de conflictos normativos en el ámbito jurídico requiere un análisis riguroso de normas, hechos y valores, así como la capacidad de adaptarse a interpretaciones cambiantes. Este artículo presenta un modelo computacional multi-agente basado en la arquitectura BDI (*Beliefs*, *Desires*, *Intentions*), que integra la Teoría Pura del Derecho de Hans Kelsen y elementos de lógica híbrida intuicionista para simular procesos de razonamiento jurídico. Implementado en el lenguaje lógico-funcional Mercury, el modelo permite que los agentes generen y ajusten argumentos frente a ataques normativos y cambios en sus creencias, replicando la dinámica adaptativa del razonamiento judicial. Se emplea el caso clásico *Pierson v. Post* como estudio preliminar para ilustrar el comportamiento del sistema. Esta propuesta, de carácter descriptivo e interdisciplinario, contribuye al desarrollo de sistemas inteligentes aplicados al derecho y abre nuevas vías para el análisis computacional de decisiones normativas complejas.

## 1. Introducción

El avance de la inteligencia artificial y su creciente aplicación en el ámbito jurídico han abierto nuevas oportunidades para desarrollar sistemas que apoyen la toma de decisiones en escenarios legales complejos [1], [2]. Se han propuesto diversos modelos de razonamiento legal, como los sistemas basados en casos [2] y los enfoques basados en lógica deóntica para representar sistemas normativos [3]. Aunque estos modelos han demostrado eficacia en la representación de normas jurídicas, presentan limitaciones al momento de simular interacciones dinámicas entre actores jurídicos con intereses y perspectivas divergentes.

Una alternativa prometedora para superar estas limitaciones es el uso de sistemas multi-agente, que permiten modelar actores como jueces, abogados y partes litigantes en escenarios jurídicos simulados [4]. Al integrarse con modelos normativos basados en lógica y teoría del derecho, estos agentes pueden representar normas, valores y principios de justicia de forma más dinámica [5]. En particular, la arquitectura BDI (Beliefs, Desires, Intentions) ofrece una base sólida para simular cómo estos actores toman decisiones y generan argumentos.

En respuesta a este panorama, este trabajo propone un modelo computacional que combina la arquitectura BDI con la Teoría Pura del Derecho de Hans Kelsen, apoyado en la lógica híbrida intuicionista (*IHL*) para manejar incertidumbre y estados intermedios en el razonamiento jurídico [6].

Como estudio de caso se emplea *Pierson v. Post* [7], un caso clásico de derecho de propiedad que plantea un conflicto normativo entre persecución y posesión. La controversia —si la mera persecución de un animal salvaje confiere derechos de propiedad— ejemplifica tensiones entre reglas legales formales y principios de equidad. Esto lo convierte en un escenario ideal para probar el modelo.

La simulación se centra en el agente *Tomkins*, quien actúa como juez y es capaz de generar argumentos normativos en función de sus creencias y deseos. Aunque se planea incorporar otros agentes (*Livingston*, *Banks*, *Tropolle*), esta fase inicial se enfoca en validar el sistema para un solo agente, dejando para etapas futuras la automatización de ataques lógicos entre múltiples actores. Por ahora, tales interacciones se simulan manualmente.

El modelo se implementó en el lenguaje lógico-funcional Mercury [8], [9], elegido por su capacidad de manejar estructuras lógicas complejas con eficiencia y precisión.

Este artículo tiene como objetivo describir esta implementación inicial y explorar su potencial para simular juicios legales desde una perspectiva normativa. La estructura del artículo se organiza como sigue: primero se presenta el marco teórico; luego, la metodología; posteriormente, los resultados preliminares; y finalmente, las conclusiones y líneas futuras de investigación.

## 2. Marco Teórico

El presente trabajo propone un modelo computacional para la resolución de disputas legales que se fundamenta en una combinación de enfoques teóricos provenientes del derecho, la inteligencia artificial y las ciencias de la computación. Este modelo, se apoya en varias teorías clave, incluyendo la **Teoría Pura del Derecho** de Hans Kelsen, los agentes BDI (*Belief-Desire-Intention*), la lógica híbrida intuicionista (*IHL*), el uso de ontologías basadas en la teoría de topos y el lenguaje de comunicación entre agentes ACL (*Agent Communication Language*) de FIPA. Estos conceptos proporcionan una base sólida para el desarrollo de un sistema multiagente que simula la toma de decisiones en un entorno legal, como *Pierson v Post*. Aunque actualmente se ha implementado solo para el agente *Tomkins*, este marco es escalable a múltiples agentes en simulaciones futuras.

El fundamento legal del modelo se basa en la **Teoría Pura del Derecho** de Hans Kelsen, que describe el derecho como un sistema normativo autónomo y jerárquico, separado de influencias externas como la moral y la política [10]. Esta teoría se basa en dos principios centrales: la existencia de una *norma fundamental* (*Grundnorm*) y la jerarquía normativa, ambos esenciales para estructurar sistemas normativos tanto en contextos legales como computacionales. La *Grundnorm* es asumida como válida y no requiere justificación adicional, sirviendo como el punto de partida lógico para todo el sistema jurídico. Formalmente, esta jerarquía puede representarse como:

$$\textit{Grundnorm} = n_0$$

Las normas derivadas $(n_1, n_2, \ldots, n_k)$ están subordinadas jerárquicamente, con su validez dependiendo de una norma superior. Esto puede representarse mediante un grafo dirigido acíclico $G = (N, A)$, donde:

- $N$ es el conjunto de normas ($n_i \in N$).
- $A$ es el conjunto de relaciones de subordinación ($a_{ij} \in A \implies n_i \to n_j$).

La función de validación de una norma puede definirse como:

$$\textit{validarnorma}(n_j) = \begin{cases} \text{Verdadero}, & \text{si } n_j \text{ es derivada de } n_i \text{ y } \textit{validarnorma}(n_i), \\ \text{Falso}, & \text{en otro caso.} \end{cases}$$

Este enfoque asegura que la validez de cada norma dentro del sistema pueda ser trazada hacia la *Grundnorm*, garantizando coherencia lógica y normatividad en el sistema.

La perspectiva [11] de Kelsen, que define el análisis jurídico en términos de validez normativa sin implicar valores morales, es ideal para un modelo computacional, ya que permite formalizar las normas jurídicas en un marco lógico riguroso y preciso. En esta implementación inicial, el modelo propuesto utiliza la jerarquía normativa de Kelsen para representar normas como nodos en un grafo dirigido. Los agentes computacionales verifican la validez normativa a través de una función recursiva basada en la *validarnorma*. Por ejemplo, si el agente $A_i$ necesita decidir si una norma $n_j$ es válida, ejecuta:

$$\textit{validarnorma}(n_j) \implies \exists n_i \text{ tal que } (n_i, n_j) \in A$$

En términos computacionales, esto se implementa mediante recorridos en profundidad (*Depth-First Search*, **DFS**) sobre el grafo G, asegurando que todas las relaciones normativas sean verificadas [12]. Este formalismo permite representar no solo la jerarquía, sino también dinámicas de adaptación cuando se introducen nuevas normas o se invalidan existentes.

Por ejemplo, en el caso *Pierson v. Post*, el agente *Tomkins* utiliza la función *validarnorma* para determinar si las normas relacionadas con la propiedad sobre el zorro cumplen con los criterios establecidos por la *Grundnorm*. Esto asegura que sus decisiones sean consistentes y justificadas en términos normativos, manteniendo un enfoque objetivo dentro del marco jurídico predefinido.

La *Teoría Pura del Derecho* permite que las decisiones de los agentes sean trazables y explicables, aspectos cruciales en el contexto jurídico. Al eliminar ambigüedades y garantizar coherencia normativa, esta teoría se convierte en una herramienta clave para superar las limitaciones de modelos tradicionales que carecen de transparencia y justificabilidad, como los basados en aprendizaje automático [13]. En este trabajo, la *Teoría Pura del Derecho* de Kelsen proporciona los fundamentos normativos necesarios para el desarrollo de un modelo computacional adaptativo y preciso, ejemplificado en el agente *Tomkins*.

En cuanto a la arquitectura de los agentes, el modelo sigue el enfoque BDI (*Belief-Desire-Intention*), ampliamente utilizado en el desarrollo de agentes autónomos en sistemas multiagente [4]. Este marco conceptual permite modelar el comportamiento adaptativo de los agentes, simulando procesos de toma de decisiones racionales mediante tres componentes clave: *creencias* (B), que representan el conocimiento o percepción del agente sobre el estado del mundo; *deseos* (D), que son las metas que el agente aspira a alcanzar; y *intenciones* (I), que corresponden a los compromisos del agente para realizar acciones dirigidas al logro de esos deseos.

Formalmente, un agente BDI puede representarse como una tupla:

$$\text{Agente} = (B, D, I),$$

donde:

- B es el conjunto de **creencias**, que incluye información sobre el entorno y otros agentes.
- D es el conjunto de **deseos**, o metas deseadas, aunque no necesariamente factibles.
- I es el conjunto de **intenciones**, que representa metas seleccionadas para guiar las acciones del agente.

El comportamiento del agente se modela mediante la relación:

$$(B, D) \rightarrow I,$$

donde las creencias B y los deseos D influyen en la generación de intenciones I a través de reglas normativas o estratégicas. Este proceso es iterativo y dinámico, ya que las intenciones se adaptan continuamente en función de las observaciones del entorno y los resultados de acciones previas. Por ejemplo, el agente actualiza sus creencias mediante:

$$B' = B \cup \textit{observaciones},$$

donde B' representa las creencias actualizadas tras incorporar nueva información del entorno.

En este modelo, el agente *Tomkins* actúa como un actor jurídico en el caso *Pierson v. Post*, empleando sus creencias B sobre los hechos del caso (por ejemplo, *"Post perseguía al zorro"* o *"Pierson mató al zorro"*), sus deseos D (como *"resolver el caso"* o *"promover el bienestar público"*), y generando intenciones I como *"evaluar los derechos de propiedad"* o *"emitir un juicio basado en normas"*.

Esta arquitectura es particularmente adecuada para el contexto jurídico porque:

- **Racionalidad práctica:** Permite modelar cómo los agentes toman decisiones basadas en sus creencias y metas normativas, lo que refleja razonamientos humanos en escenarios legales [14].
- **Adaptabilidad:** Los agentes pueden ajustar sus decisiones dinámicamente a medida que se incorporan nuevas evidencias o cambian las condiciones del entorno.
- **Transparencia:** La estructura explícita de B, D e I facilita la trazabilidad y justificación de las decisiones, lo cual es crucial en sistemas normativos.

El uso de BDI en este trabajo permite generar argumentos adaptativos que simulan procesos de razonamiento jurídico. Por ejemplo, en el caso *Pierson v. Post*, *Tomkins* analiza las creencias iniciales sobre la posesión del zorro, evalúa escenarios posibles basados en sus deseos y produce decisiones normativas consistentes con los principios del derecho. Este enfoque demuestra la capacidad del modelo para manejar conflictos normativos en sistemas complejos y dinámicos.

Para manejar la incertidumbre en las creencias y los hechos disponibles, el modelo incorpora la **Lógica Híbrida Intuicionista (IHL)**, una extensión de la lógica modal que permite representar estados intermedios de conocimiento y creencias en un entorno dinámico. La lógica intuicionista se distingue de la lógica clásica al rechazar el principio del tercero excluido,

admitiendo que una proposición p no siempre es verdadera (p) o falsa (¬p), sino que puede encontrarse en un estado de conocimiento incompleto [6]. Además, la lógica modal, base de la IHL, introduce los operadores $\Diamond$ y $\Box$ para razonar sobre **posibilidades** y **necesidades**, respectivamente. El operador $\Diamond p$ indica que la proposición p es posible, mientras que $\Box p$ expresa que p es necesariamente verdadera. Este marco lógico es particularmente adecuado para el sistema legal, donde la información a menudo es incompleta, incierta o está sujeta a cambios.

Formalmente, un modelo de lógica híbrida intuicionista puede representarse como una tupla:

$$M = (W, R, V, @_w),$$

donde:

- $W$: Conjunto de **mundos posibles**.
- R: Relación de **accesibilidad** entre los mundos, que define cómo se conectan los distintos estados de conocimiento.
- $V$: Función de **valoración**, que asigna a cada proposición el conjunto de mundos donde es verdadera.
- $@_w$: Operador que permite referirse a proposiciones en mundos específicos, facilitando el razonamiento sobre escenarios pasados, alternativos o hipotéticos.

La semántica de sus principales operadores se describe como sigue:

- $\Box p$: p es verdadera en un mundo $w$ si lo es en todos los mundos accesibles desde $w$:

$$M, w \models \Box p \iff \forall w'(R(w, w') \Rightarrow M, w' \models p).$$

- $\Diamond p$: p es verdadera en un mundo $w$ si lo es en al menos un mundo accesible desde $w$:

$$M, w \models \Diamond p \iff \exists w'(R(w, w') \wedge M, w' \models p).$$

- $@_w p$: p es verdadera en un mundo específico $w$, lo que permite razonar sobre evidencia particular en ese mundo:

$$M, w \models @_v p \iff M, v \models p, \text{ donde } v \in W.$$

En el modelo propuesto, la IHL permite que los agentes, como *Tomkins*, gestionen estados intermedios de conocimiento y adapten sus creencias y acciones conforme surge nueva información. Por ejemplo, en el caso *Pierson v. Post*, el agente puede enfrentar un estado de conocimiento incompleto respecto a la proposición *"Post poseía al zorro"*, representado como:

$$\neg(\Box p \vee \neg\Box p),$$

indicando que la verdad de p no puede determinarse completamente en el estado actual.

La IHL también proporciona herramientas para razonar sobre escenarios futuros y evaluar la necesidad o posibilidad de ciertos hechos normativos. Por ejemplo:

- **Posibilidad futura:** El agente puede evaluar si es posible que un derecho de propiedad sea reconocido:
$$\Diamond(\textit{Pierson adquiere la propiedad del zorro}).$$
- **Necesidad normativa:** Puede determinar si es necesario un principio para la resolución del conflicto:
$$\Box(\textit{la captura física establece la propiedad}).$$

Este enfoque permite que los agentes adapten sus razonamientos legales de manera continua y justificada, manteniendo coherencia con los principios normativos subyacentes al sistema jurídico. La incorporación de la IHL no solo mejora la capacidad del modelo para manejar incertidumbre, sino que también enriquece el análisis de escenarios hipotéticos, fortaleciendo la trazabilidad y justificabilidad de las decisiones tomadas por los agentes.

El modelo también incorpora **ontologías basadas en la teoría de topos** para estructurar los conceptos legales como posesión, propiedad y normativa. La **teoría de topos**, desarrollada por Grothendieck y Lawvere [15], surge como una generalización de la teoría de conjuntos y ofrece un marco matemático que encapsula tanto relaciones lógicas como estructuras geométricas. Un *topos* (significa "lugar" o "posición") es una categoría $\mathcal{T}$ que posee propiedades que lo hacen un modelo unificado para representar conceptos abstractos y relaciones complejas.

Formalmente, un *topos* se define como una tupla:

$$\mathcal{T} = (\mathrm{Obj}(\mathcal{T}), \mathrm{Mor}(\mathcal{T}), \Omega),$$

donde:

- $\mathrm{Obj}(\mathcal{T})$: Conjunto de **objetos**, que representan las entidades abstractas en el dominio, como los conceptos legales de posesión y propiedad.
- $\mathrm{Mor}(\mathcal{T})$: Conjunto de **morfismos**, que denotan relaciones entre objetos, representadas como $f : A \to B$, donde $A, B \in \mathrm{Obj}(\mathcal{T})$.
- $\Omega$: El **clasificador de subobjetos**, que actúa como un dominio lógico dentro del topos, permitiendo interpretar proposiciones sobre los objetos.

En este marco, el clasificador de subobjetos $\Omega$ permite modelar proposiciones como morfismos. Por ejemplo, un subobjeto de $A$, que puede interpretarse como un subconjunto en la teoría de conjuntos, es un morfismo $f : A \to \Omega$, donde $\Omega$ codifica las verdades lógicas del sistema.

Para una proposición $P$, su evaluación en un objeto $A$ se define como:

$$P(A) = \{x \in A \mid P(x) \text{ es verdadero en } \Omega\}.$$

En el contexto de este trabajo, las **ontologías basadas en la teoría de topos** proporcionan un marco riguroso para estructurar relaciones entre actores legales (como demandantes o jueces) y conceptos jurídicos (como posesión y propiedad). Por ejemplo, las relaciones entre posesión y propiedad pueden representarse como morfismos en un topos:

$$\text{Posesión}(D, O) \rightarrow \text{Propiedad}(D, O),$$

donde:

- Posesión$(D, O)$: Representa la posesión efectiva del objeto $O$ por el actor $D$.
- Propiedad$(D, O)$: Denota la propiedad legal del objeto $O$ por $D$.
- Los morfismos codifican transiciones legales, como transferencias de derechos o disputas sobre la validez de la posesión.

El uso de esta teoría permite extender el modelo a entornos más complejos, donde múltiples agentes interactúan y las relaciones legales entre ellos pueden representarse de manera precisa y adaptable. Además, la estructura jerárquica y lógica de los topos refleja de forma natural la organización normativa descrita en la *Teoría Pura del Derecho* de Kelsen, lo que facilita la coherencia normativa en los sistemas multiagente.

Este marco extensible es particularmente útil en simulaciones jurídicas complejas, como el caso *Pierson v. Post*, donde conceptos como la posesión y la propiedad pueden evaluarse dinámicamente dentro de un sistema normativo. Por ejemplo, el clasificador $\Omega$ puede utilizarse para verificar si un subobjeto, como la posesión efectiva, cumple con los criterios normativos requeridos para derivar una propiedad legal. Esto proporciona no solo una representación formal de las relaciones legales, sino también una base lógica para razonar sobre ellas.

Finalmente, se utiliza el **Lenguaje de Comunicación entre Agentes (ACL)** de *FIPA* para facilitar la interacción y coordinación en sistemas multi-agente distribuidos. Aunque actualmente el agente *Tomkins* no interactúa con otros agentes, el ACL sería esencial en el modelo ampliado para permitir la comunicación entre agentes BDI que compartan o negocien creencias, deseos e intenciones, como en el caso de una simulación jurídica completa con múltiples actores [16]. Estos agentes podrían utilizar el ACL para realizar actos de lenguaje como informar, proponer o negociar, lo que resulta fundamental para simular procesos como la negociación y la deliberación entre las partes en conflicto [17]. Los conceptos de actos de habla fueron extensamente desarrollados por John R. Searle, quien en su obra *"Speech Acts: An Essay in the Philosophy of Language"* [18], exploró la capacidad de las palabras para realizar acciones en lugar de solo transmitir información [19].

La semántica de ACL se puede representar utilizando operadores modales de necesidad ($\Box$) y posibilidad ($\Diamond$), que permiten expresar propiedades clave de los actos comunicativos. Por ejemplo, el acto de *informar* implica que el agente emisor $a$:

- Cree necesariamente ($\Box$) que el contenido de su mensaje $\phi$ es verdadero.
- Considera posible ($\Diamond$) que el agente receptor $b$ pueda incorporar $\phi$ a su conjunto de conocimiento.

Formalmente, el acto de *informar* se puede modelar como:

$$\text{informar}(a, b, \phi) \iff \Box\phi \wedge \Diamond(\text{Sabe}(b, \phi)),$$

donde:

- $\alpha$ es el agente emisor.
- $b$ es el agente receptor.
- $\phi$ es la proposición transmitida.
- $\Box\phi$ representa que $\alpha$ necesariamente cree que $\phi$ es verdadera.
- $\Diamond(\text{Sabe}(b, \phi))$ indica que $\alpha$ considera posible que $b$ llegue a saber que $\phi$.

Este uso de los operadores modales proporciona una base lógica sencilla pero robusta para modelar la semántica de ACL, enfocándose en los aspectos de certeza y posibilidad en los actos de comunicación entre agentes. Además, los operadores modales pueden extenderse para capturar dinámicas más complejas de interacción en futuras implementaciones del modelo.

En este modelo, la comunicación entre agentes no es solo un intercambio de datos, sino un acto racional basado en sus estados internos (creencias, deseos, intenciones) y en la dinámica de su entorno.

Aunque actualmente el agente *Tomkins* no interactúa con otros agentes, el ACL sería fundamental en el modelo ampliado para habilitar una comunicación eficaz en simulaciones jurídicas más complejas con múltiples actores. Por ejemplo, agentes BDI podrían utilizar el ACL para negociar acuerdos, resolver conflictos o coordinar estrategias legales, realizando actos de lenguaje como *proponer*, *rechazar* o *aceptar*. Esto no solo facilitaría la colaboración entre agentes, sino que también permitiría modelar dinámicamente procesos como la deliberación y la negociación jurídica [17].

La implementación de ACL en el modelo proporciona un marco robusto y estandarizado para la interacción multi-agente, asegurando interoperabilidad y extensibilidad para futuras investigaciones.

Este conjunto de teorías y herramientas permite construir un modelo escalable y adaptable para simular decisiones jurídicas basadas en normas y principios, como se ha comenzado a implementar en el agente *Tomkins* para el caso *Pierson v Post*. El marco teórico descrito proporciona un soporte sólido para futuras expansiones que permitirían una simulación detallada de interacciones entre agentes en un juicio legal completo.

# 3. Metodología

Esta sección describe el proceso de implementación del modelo de agentes BDI utilizando el lenguaje de programación *Mercury*, un lenguaje lógico-funcional especialmente adecuado para modelar sistemas normativos. Su sistema de tipos fuerte, inferencia de modos y optimización en tiempo de compilación contribuyen a una implementación precisa y coherente, aspectos esenciales al simular el comportamiento de actores jurídicos en contextos normativos.

El modelo desarrollado emplea una arquitectura BDI (*Belief-Desire-Intention*) para representar de manera estructurada las creencias, deseos e intenciones de los agentes involucrados en el caso *Pierson v. Post*. En esta etapa inicial, la implementación se centró exclusivamente

en el agente *Tomkins*, quien desempeña el rol de juez. Aunque el objetivo final es simular el juicio completo con cuatro agentes (Tomkins, Livingston, Banks y Tropolle), esta fase se enfocó en construir una base sólida para su posterior expansión.

Una característica fundamental de *Mercury* que fortalece la robustez del modelo es su sistema de tipos estático, que permite definir estructuras de datos complejas y garantizar relaciones internas consistentes. Este control desde la fase de compilación previene errores de tipo y asegura integridad lógica en entornos sensibles como el jurídico. La definición del tipo *agent*, mostrado a continuación, encapsula los componentes esenciales de un agente BDI.

```
:- type agent --->
    agent(
        beliefs :: list(belief),
        desires :: list(desire),
        arguments :: list(argument)
    ).
```

Listing 1: Definición del tipo *agent*.

Esta estructura permite integrar de forma cohesionada los elementos internos del agente y gestionarlos con precisión. La capacidad de Mercury para trabajar con tipos algebraicos facilita una representación concisa pero detallada de cada componente del estado mental del agente, lo cual es indispensable en la simulación de razonamiento jurídico.

El agente *Tomkins* es inicializado con un conjunto de *beliefs*, *desires* e *intentions*, definidos como tipos algebraicos. En el siguiente fragmento, se muestran las proposiciones relevantes al caso y la forma en que se modelan las creencias.

```
% Definicion de proposiciones y creencias.
:- type proposition --->
    f1; % Post estaba persiguiendo al zorro.
    f2; % Post no habia capturado ni herido al zorro.
    f3; % Pierson mató al zorro para arruinar el deporte de Post.
    f4; % Los zorros matan ganado.
    f5; % Fomentar la caza reducirá el número de zorros.
    f6; % Reducir el número de zorros protege el ganado.
    f7. % Si la caza se desalienta, no se inflige sufrimiento innecesario.

% Definicion de las creencias del agente.
:- type belief ---> belief(proposition); disbelief(proposition); unknown(proposition).
```

Listing 2: Definición de tipos para representar proposiciones y creencias.

Este modelo permite que Tomkins adopte posturas afirmativas, negativas o neutrales sobre cada proposición, lo que refleja la ambigüedad y complejidad de los entornos legales. La fuerte tipificación impide que se representen estados inválidos, asegurando que la simulación se mantenga dentro de parámetros normativos coherentes.

Los deseos del agente modelan los objetivos que este aspira a lograr, y las intenciones representan los planes elegidos para cumplir dichos objetivos. Ambos se definen también como tipos algebraicos. A continuación se presentan los valores, condiciones y estructuras que permiten expresar estos elementos.

```
% Condiciones para los deseos.
:- type condition ---> ownership_plaintiff;
    no_ownership_defendant; malicious_intent_plaintiff.

% Valores que el agente busca maximizar.
```

```
:- type value ---> less_litigation;
    public_benefit; economic_benefit.

% Deseos que vinculan un nombre, un valor y una condición.
:- type desire ---> desire(name, value, condition).

% Argumento compuesto por creencias, acción, metas y valores.
:- type argument ---> argument(
    beliefs::list(belief), action::action,
    goals::list(proposition), values::list(value)
).
```

Listing 3: Definición de deseos y argumentos.

Esta representación permite al agente alinear sus objetivos con valores normativos como el beneficio público o la reducción de litigios. La semántica declarativa de Mercury favorece una implementación clara de estos componentes.

Uno de los aspectos clave del modelo es la generación automática de argumentos que respalden las intenciones del agente. Cada argumento propuesto es evaluado en función de su compatibilidad con las creencias y deseos del agente. El siguiente predicado decide si un argumento puede integrarse al conjunto de intenciones del agente:

```
:- pred should_add_argument(agent::in, argument::in) is semidet.
should_add_argument(Tomkins, Arg) :-
    tomkins_belief_in_argument(Tomkins^beliefs, Arg),
    tomkins_desire_value_in_argument(Tomkins^desires, Arg).
```

Listing 4: Predicado para evaluar un argumento propuesto.

Si el argumento es coherente con el estado mental del agente, se incorpora a su conjunto de intenciones, lo cual ilustra el proceso de razonamiento interno basado en metas y consistencia lógica [4].

La siguiente función muestra cómo se filtran los argumentos válidos y se actualiza el estado del agente:

```
:- func add_arguments_to_tomkins(agent) = agent.
add_arguments_to_tomkins(Tomkins) = TomkinsUpdated :-
    Arguments = [arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10],
    ValidArguments = list.filter(should_add_argument(Tomkins), Arguments),
    TomkinsUpdated = Tomkins^arguments := ValidArguments.
```

Listing 5: Función para actualizar el conjunto de argumentos del agente.

Este mecanismo permite al agente modificar dinámicamente su razonamiento conforme cambian sus creencias o prioridades, lo que simula con precisión la actividad judicial frente a evidencia o argumentos en disputa.

En esta etapa, los ataques a las creencias del agente se simulan manualmente, pero ya permiten observar cómo dichos desafíos alteran la validez de sus argumentos. Esta capacidad de adaptación es crucial para representar la dinámica del razonamiento en escenarios legales complejos.

En resumen, la implementación en *Mercury* del modelo BDI proporciona una estructura lógica y tipada que permite representar con precisión la toma de decisiones en contextos jurídicos. La metodología empleada garantiza coherencia, extensibilidad y claridad, estableciendo una base sólida para incorporar agentes adicionales y mecanismos automatizados de ataque y defensa argumentativa en futuras etapas del proyecto.

# 4. Resultados

El modelo propuesto, basado en una arquitectura BDI (Belief-Desire-Intention) e implementado en el lenguaje *Mercury*, demuestra cómo es posible representar procesos de decisión jurídica mediante un enfoque normativo y racional. Para validar su funcionamiento, se utilizó el caso clásico *Pierson v. Post*, simulando cómo un agente, en este caso *Tomkins* (juez), toma decisiones a partir de sus creencias, deseos e intenciones.

Este enfoque se inspira en la Teoría Pura del Derecho de Hans Kelsen, lo que implica que las decisiones del agente se guían por la validez formal de las normas, sin recurrir a factores morales o subjetivos externos. El agente opera dentro de un marco normativo bien definido, lo cual garantiza decisiones objetivas y legalmente coherentes.

El siguiente fragmento de código muestra la inicialización del agente *Tomkins* con un conjunto de creencias y deseos normativos:

```
Tomkins = agent(
    [belief(f1), belief(f2), belief(f3), belief(f4), unknown(f5), belief(f6)],
    [desire(clear_law, less_litigation, ownership_plaintiff),
     desire(unclear_law, less_litigation, ownership_no_possession),
     desire(trade_restricted, economic_benefit,
         ↪ malicious_intent_productive_activity_defendant)],
    []
).
```

Listing 6: Inicialización del agente Tomkins con creencias y deseos.

Las creencias reflejan proposiciones del caso, como la persecución del zorro por parte de Post (`f1`) o la amenaza que representan los zorros para el ganado (`f4`). Los deseos de Tomkins están vinculados a objetivos jurídicos como reducir el litigio y promover beneficios públicos o económicos. La lógica híbrida intuicionista (IHL) permite representar estados de creencia inciertos, como `unknown(f5)`, lo cual añade flexibilidad al razonamiento legal del agente.

Una vez definido su estado mental inicial, Tomkins evalúa argumentos construidos a partir de creencias, acciones, metas y valores. A continuación, se presentan dos argumentos con enfoques contrapuestos:

```
:- func arg1 = argument.
arg1 = argument(
    [belief(f4), belief(f5), belief(f6)],
    encourage_fox_hunting,
    [f6],
    [public_benefit]
).

:- func arg2 = argument.
arg2 = argument(
    [belief(f7)],
    discourage_fox_hunting,
    [f7],
    [humaneness]
).
```

Listing 7: Argumentos evaluados por Tomkins.

El argumento `arg1` defiende la caza del zorro como forma de proteger al ganado, en consonancia con el valor del beneficio público. En contraste, `arg2` se basa en la compasión hacia

los animales, abogando por desalentar la caza. Esta dualidad permite observar cómo Tomkins selecciona argumentos según su alineación con creencias y valores normativos.

El siguiente paso es evaluar cuáles de estos argumentos son coherentes con el estado mental del agente:

```
:- pred should_add_argument(agent::in, argument::in) is semidet.
should_add_argument(Tomkins, Arg) :-
    tomkins_belief_in_argument(Tomkins^beliefs, Arg),
    tomkins_desire_value_in_argument(Tomkins^desires, Arg).
```

Listing 8: Evaluación de argumentos por el agente Tomkins.

Solo los argumentos que satisfacen las creencias y valores deseados serán integrados al razonamiento del agente:

```
:- func add_arguments_to_tomkins(agent) = agent.
add_arguments_to_tomkins(Tomkins) = TomkinsUpdated :-
    Arguments = [arg1, arg2, ..., arg10],
    ValidArguments = list.filter(should_add_argument(Tomkins), Arguments),
    TomkinsUpdated = Tomkins^arguments := ValidArguments.
```

Listing 9: Actualización de intenciones del agente.

Este mecanismo permite que el agente actualice dinámicamente sus intenciones de acuerdo con su marco normativo, lo que resulta esencial en contextos jurídicos donde nuevas pruebas o argumentos pueden surgir.

Además, el modelo permite simular ataques entre agentes. Estos ataques desafían creencias fundamentales, obligando a reevaluar los argumentos asociados. El siguiente ejemplo presenta un ataque contra la creencia `f1`:

```
:- func attack1a = attack.
attack1a = attack(
    [belief(f1)],
    [ownership_plaintiff],
    [disbelief(f1)]
).
```

Listing 10: Ataque contra la creencia f1.

El impacto de un ataque se procesa mediante un mecanismo de actualización de creencias:

```
:- pred apply_attack(agent::in, attack::in, agent::out) is det.
apply_attack(!Agent, Attack) :-
    Attack = attack(QuestionedBeliefs, Conditions, ArgumentBeliefs),
    list.foldl((pred(Belief::in, A0::in, A::out) is det :-
        (if belief_in_list(Belief, A0^beliefs) then
            (if list.member(desire(_, _, Condition), A0^desires) and
                list.member(Condition, Conditions) then
                    list.delete_all(A0^beliefs, Belief, TempBeliefs),
                    list.append(TempBeliefs, ArgumentBeliefs, NewBeliefs),
                    A = A0^beliefs := NewBeliefs
            else
                A = A0)
        else
            A = A0)
    ), QuestionedBeliefs, !Agent).
```

Listing 11: Aplicación del ataque al agente.

Una vez modificado el sistema de creencias, el agente puede reevaluar los argumentos disponibles y actualizar sus intenciones:

```
TomkinsUpdated = add_arguments_to_tomkins(TomkinsUpdatedAttack),
TomkinsReevaluatedArguments = list.map(argument_name, TomkinsUpdated^arguments)
io.print("= Nuevas intenciones del agente Tomkins: =\n"),
io.print(TomkinsReevaluatedArguments, !IO).
```

Listing 12: Reevaluación de intenciones tras ataque.

Este flujo simula el razonamiento jurídico dinámico, donde un argumento previamente válido puede perder fuerza al cambiar el contexto normativo del agente. Por ejemplo, si Tomkins ya no cree en la posesión por persecución, dejará de apoyar la asignación de propiedad al demandante.

El modelo también ha sido probado con simulaciones adicionales, incorporando otros agentes como *Banks* y *Anthony*, quienes aportan distintas perspectivas jurídicas. Estas simulaciones mostraron cómo el sistema se adapta a diferentes valores normativos y contextos, ofreciendo decisiones coherentes pero variables, según la configuración interna de cada agente.

En conjunto, los resultados muestran que la arquitectura BDI implementada en *Mercury* es capaz de representar razonamientos legales complejos, incluyendo el análisis de argumentos, la adaptación frente a ataques y la toma de decisiones en entornos normativos cambiantes. El uso de lógica formal, tipos algebraicos y razonamiento simbólico proporciona un marco sólido y escalable para el estudio computacional del derecho.

Además, la capacidad de modificar dinámicamente las creencias permite simular contextos legales realistas, donde la información puede ser ambigua, contradictoria o incompleta. Esto no solo resulta útil desde una perspectiva académica, sino que abre caminos hacia el desarrollo de herramientas automatizadas para análisis jurídico, enseñanza del derecho y diseño de sistemas normativos artificiales.

# 5. Discusión

La resolución computacional de disputas jurídicas sigue siendo un desafío para la inteligencia artificial, debido a la naturaleza dinámica, argumentativa y normativamente compleja del derecho. En este contexto, el modelo propuesto basado en agentes BDI implementado en *Mercury* representa un avance significativo por su capacidad para simular razonamientos jurídicos transparentes, adaptables y teóricamente fundamentados.

A diferencia de enfoques basados en aprendizaje automático como Lex Machina, Kira Systems o CaseMine, que se centran en el análisis de grandes volúmenes de datos mediante técnicas estadísticas, nuestro modelo no opera como una "caja negra". Por el contrario, cada decisión del agente es rastreable y justificable, ya que se construye explícitamente sobre estructuras normativas y principios argumentativos definidos. Esta trazabilidad es crucial en entornos jurídicos, donde la legitimidad de una decisión depende tanto de su contenido como de su justificación.

En comparación con marcos argumentativos formales como Carneades o Reason-Based Logic (RBL), que han sido fundamentales en el análisis lógico de argumentos jurídicos, nuestro enfoque introduce un mayor grado de adaptabilidad. El agente Tomkins no solo genera

argumentos alineados con sus creencias y deseos, sino que puede reevaluarlos activamente ante ataques y cambios de contexto. Esto permite representar la evolución del razonamiento jurídico a lo largo de un juicio, algo que los modelos con creencias estáticas difícilmente logran simular con fidelidad.

Una de las contribuciones más destacadas del modelo es su capacidad para gestionar conflictos normativos y estados de incertidumbre mediante la integración de lógica híbrida intuicionista (IHL). Este enfoque, inspirado en trabajos como los de Shams [20] y Governatori [21], permite representar creencias intermedias o desconocidas sin comprometer la coherencia del sistema. Así, los agentes pueden operar en entornos donde las normas son ambiguas, contradictorias o en constante cambio, reflejando mejor la realidad del derecho contemporáneo.

Pese a sus ventajas, el modelo presenta desafíos en términos de escalabilidad. A medida que se incorporan más agentes, normas y creencias, la complejidad computacional crece. Aunque *Mercury* ofrece un rendimiento optimizado para programación lógica-funcional, serán necesarias estrategias de optimización para extender el modelo a escenarios más complejos. Además, la incorporación de normas dinámicas, como sugiere García [22], permitiría una representación más granular de los sistemas jurídicos donde las reglas evolucionan con el contexto.

La simulación del caso *Pierson v. Post* demuestra la aplicabilidad del modelo a un escenario clásico. Sin embargo, para ampliar su utilidad, sería recomendable desarrollar una biblioteca de casos con distintas estructuras normativas y valorativas. Esto permitiría explorar cómo los agentes ajustan sus estrategias frente a precedentes, doctrinas o valores diversos, abriendo posibilidades para una inteligencia artificial jurídica más contextual y versátil.

En definitiva, el valor del modelo no reside únicamente en su implementación técnica. La verdadera aportación está en su arquitectura conceptual: una síntesis entre la Teoría Pura del Derecho de Kelsen, la lógica formal y la teoría de agentes BDI. Esta combinación permite capturar la dimensión normativa del derecho, su estructura argumentativa y su capacidad de adaptación. Mientras que cualquier programador podría replicar el modelo en *Mercury*, su diseño refleja una comprensión profunda de los desafíos filosóficos y estructurales del razonamiento jurídico. Esta integración interdisciplinaria ofrece una vía prometedora para el desarrollo de sistemas jurídicos artificiales más transparentes, adaptables y normativamente coherentes.

# 6. Conclusiones

Este trabajo ha presentado un modelo computacional innovador para la resolución de disputas legales, integrando la Teoría Pura del Derecho de Kelsen, agentes *BDI*, lógica híbrida intuicionista (*IHL*) y una ontología basada en *topoi*. La implementación en *Mercury* ha permitido simular cómo un agente jurídico puede generar, revisar y adaptar argumentos normativos en función de sus creencias y deseos, dentro de un marco estructurado y justificable.

A través del estudio del caso *Pierson v. Post*, se demuestra que el modelo no solo es funcional, sino que ofrece una aproximación transparente y dinámica al razonamiento jurídico, capaz de responder a ataques y transformaciones en el entorno normativo. Esta capacidad

adaptativa representa una diferencia clave frente a enfoques más estáticos o automatizados que no explicitan la lógica de sus decisiones.

Más allá de su valor técnico, el modelo destaca por su aporte conceptual: propone una forma de representar computacionalmente el derecho sin sacrificar su estructura normativa ni su dimensión argumentativa. Esto lo convierte en una herramienta con potencial no solo en simulaciones jurídicas, sino también en educación legal, diseño institucional y análisis normativo en contextos complejos.

Aunque la implementación actual se limita a un solo agente, se sienta una base sólida para extender el modelo a escenarios multiagente y casos más diversos. La posibilidad de incorporar dinámicas complejas, contradicciones normativas y valores divergentes abre un camino fértil para futuras investigaciones interdisciplinarias.

En este sentido, los investigadores, desarrolladores y teóricos del derecho pueden considerar este enfoque como una plataforma para modelar, experimentar y entender los procesos normativos de forma más rica y formalizada. La IA aplicada al derecho no debe limitarse a predecir resultados, sino que puede ayudarnos a reflexionar sobre cómo y por qué se toman ciertas decisiones legales. Este trabajo es un paso en esa dirección.

# Referencias


[1] T. Bench-Capon y G. Sartor, «A model of legal reasoning with cases incorporating theories and values,» *Artificial Intelligence*, vol. 150, n.º 1, págs. 97-143, 2003, ISSN: 0004-3702.

[2] K. D. Ashley, *Tutorial on AI and Legal Reasoning*, 2005.

[3] J.-J. C. Meyer, R. J. Wieringa et al., «Deontic logic in computer science normative system specification,» *HeinOnline*, vol. 3, pág. 93, 1993.

[4] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[5] C. S. W. V. Andrés García-Camino Juan A. Rodríguez-Aguilar, «Constraint rule-based programming of norms for electronic institutions,» *Autonomous Agents and Multi-Agent Systems*, vol. 18, n.º 1, págs. 186-217, sep. de 2008, ISSN: 1573-7454.

[6] T. Braüner y V. de Paiva, «Intuitionistic hybrid logic,» *Journal of Applied Logic*, vol. 4, n.º 3, págs. 231-255, 2006, Methods for Modalities 3 (M4M-3), ISSN: 1570-8683. DOI: https://doi.org/10.1016/j.jal.2005.06.009 dirección: https://www.sciencedirect.com/science/article/pii/S1570868305000443

[7] N. Y. C. of Appeals, *Pierson v. Post, 3 Cai. R. 175 (N.Y. 1805)*, Disponible en https://www.nycourts.gov/reporter/archives/pierson_post.htm (última consulta: noviembre 2024), 1805.

[8] Z. Somogyi, F. Henderson y T. Conway, «The execution algorithm of mercury, an efficient purely declarative logic programming language,» *The Journal of Logic Programming*, vol. 29, n.º 1, págs. 17-64, 1996, High-Performance Implementations of Logic Programming Systems, ISSN: 0743-1066.

[9] F. Henderson et al., «The Mercury language reference manual,» 1996.

[10] M. S. Green, «Hans Kelsen and the Logic of Legal Systems,» *Alabama Law Review*, George Mason Law & Economics Research Paper No. 03-50, vol. 53, págs. 365-413, 2003.

[11] H. Alessa, «The role of Artificial Intelligence in Online Dispute Resolution: A brief and critical overview,» *Information & Communications Technology Law*, vol. 31, n.º 3, págs. 319-342, 2022. DOI: 10.1080/13600834.2022.2088060 eprint: https://doi.org/10.1080/13600834.2022.2088060. dirección: https://doi.org/10.1080/13600834.2022.2088060

[12] R. E. Tarjan y U. Zwick, «Finding strong components using depth-first search,» *European Journal of Combinatorics*, vol. 119, pág. 103 815, 2024, Dedicated to the memory of Pierre Rosenstiehl, ISSN: 0195-6698. DOI: https://doi.org/10.1016/j.ejc.2023.103815 dirección: https://www.sciencedirect.com/science/article/pii/S0195669823001324

[13] D. Restrepo Amariles y P. M. Baquero, «Promises and limits of law for a human-centric artificial intelligence,» *Computer Law & Security Review*, vol. 48, pág. 105 795, 2023, ISSN: 0267-3649.

[14] A. S. Rao y M. P. Georgeff, «BDI Agents: From Theory to Practice,» en *International Conference on Multiagent Systems*, 1995.

[15] G. E. Reyes, «A Topos-Theoretic Approach to Reference and Modality,» *Notre Dame Journal of Formal Logic*, vol. 32, n.º 3, págs. 359-391, 1991. DOI: 10.1305/ndjfl/1093635834

[16] IEEE Foundation for Intelligent Physical Agents (FIPA), *Design Process Documentation Template*, http://www.fipa.org/, IEEE FIPA DPDF Working Group, 2001.

[17] K. Greenwood, T. B. Capon y P. McBurney, «Towards a computational account of persuasion in law,» en *Proceedings of the 9th International Conference on Artificial Intelligence and Law*, ép. ICAIL '03, Scotland, United Kingdom: Association for Computing Machinery, 2003, págs. 22–31, ISBN: 1581137478.

[18] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press, 1969. dirección: https://www.cambridge.org/core/books/speech-acts/D2D7B03E472C8A390ED60B86E08640E7

[19] L. Villoro, *Creer, Saber y Conocer*, 2da. Edición, S. XXI, ed. Siglo XXI, 1989, ISBN: 9682316944.

[20] J. P. W. W. V. Zohreh Shams Marina De Vos, «Practical reasoning with norms for autonomous software agents,» *Engineering Applications of Artificial Intelligence*, vol. 65, págs. 388-399, 2017.

[21] G. Governatori y A. Rotolo, «BIO logical agents: Norms, beliefs, intentions in defeasible logic,» *Autonomous Agents and Multi-Agent Systems*, vol. 17, n.º 1, págs. 36-69, 2008.

[22] A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra y W. Vasconcelos, «Constraint rule-based programming of norms for electronic institutions,» *Autonomous agents and multi-agent systems*, vol. 18, n.º 1, págs. 186-217, 2009.

# 7. Aplicación del modelo del caso *Pierson v. Post* en Mercury

```
:- module test1.
:- interface.
:- import_module io.

% Prototipos de funciones y tipos.
:- pred main(io::di, io::uo) is det.

:- implementation.
:- import_module list, bool, string.
:- import_module pair.

% Definición de proposiciones utilizadas en el caso legal.
:- type proposition --->
    f1;  % Post estaba en persecución del zorro.
    f2;  % Post no había capturado ni herido al zorro.
    f3;  % Pierson mató al zorro para arruinar el deporte de Post.
    f4;  % Los zorros matan ganado.
    f5;  % Fomentar la caza reducirá el número de zorros.
    f6;  % Reducir el número de zorros protege al ganado.
    f7.  % Si se desalienta la caza, no se inflige sufrimiento innecesario a los animales
        ↪ .

% Las creencias pueden ser positivas, negativas o desconocidas respecto a las
    ↪ proposiciones.
:- type belief --->
    belief(proposition);
    disbelief(proposition);
    unknown(proposition).

% Condiciones bajo las cuales los deseos son válidos.
:- type condition --->
    ownership_plaintiff;
    no_ownership_defendant;
    no_ownership_no_possession;
    ownership_no_possession;
    no_ownership_plaintiff;
    no_ownership_possession;
    malicious_intent_productive_activity_defendant;
    malicious_intent_plaintiff;
    malicious_intent_defendant;
    fewer_foxes_farmers_protected;
    no_fewer_foxes_no_farmers_protected;
    reduced_animal_suffering;
    no_reduced_animal_suffering;
    ownership_pursuit;
    no_pursuit_no_ownership.

% Valores que un agente puede intentar maximizar o mantener.
:- type value --->
    less_litigation;
    economic_benefit;
    public_benefit;
    humaneness.

% Nombres de deseos que reflejan estrategias o metas legales.
:- type name --->
    clear_law ;
    unclear_law ;
```

```
    trade_restricted ;
    malice_condemned ;
    malice_condoned ;
    less_threat_to_others ;
    more_threat_to_others ;
    more_suffering ;
    less_suffering .

% Acciones que un agente puede tomar basadas en el escenario legal.
:- type action --->
    encourage_fox_hunting ;
    discourage_fox_hunting ;
    find_ownership_established ;
    find_no_ownership ;
    find_for_plaintiff ;
    find_for_defendant ;
    not_find_for_defendant .

% Los deseos combinan nombres, valores y condiciones.
:- type desire --->
    desire(name, value, condition).

% Los argumentos son combinaciones de creencias, acciones, metas y valores.
:- type argument --->
    argument(
        beliefs :: list(belief),
        action :: action,
        goals :: list(proposition),
        values :: list(value)
    ).

% Los ataques están definidos para desafiar las creencias de los agentes.
:- type attack --->
    attack(
        question :: list(belief),
        conditions :: list(condition),
        argument :: list(belief)
    ).

% Estructura del agente actualizada.
:- type agent --->
    agent(
        beliefs :: list(belief),
        desires :: list(desire),
        arguments :: list(argument)
    ).

% Función para añadir argumentos válidos a un agente basándose en creencias y deseos.
:- func add_arguments_to_\textit{Tomkins}(agent) = agent.
add_arguments_to_\textit{Tomkins}(\textit{Tomkins}) = \textit{Tomkins}Updated :-
    Arguments = [arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10],
    ValidArguments = list.filter(should_add_argument(\textit{Tomkins}), Arguments),
    \textit{Tomkins}Updated = \textit{Tomkins}^arguments := ValidArguments.

% Punto de entrada para el programa Mercury.
main(!IO) :-
    % Configuración inicial del agente con creencias y deseos.
    \textit{Tomkins} = agent(
        [belief(f1), belief(f2), belief(f3), belief(f4), unknown(f5), belief(f6), unknown
            ↪ (f7)],
        [desire(clear_law, less_litigation, ownership_plaintiff), desire(unclear_law,
            ↪ less_litigation, ownership_no_possession), desire(trade_restricted,
            ↪ economic_benefit, malicious_intent_productive_activity_defendant)],
```

```
        []
    ),
    % Actualiza el agente con argumentos válidos y muestra los resultados.
    \textit{Tomkins}Updated = add_arguments_to_\textit{Tomkins}(\textit{Tomkins}),

    % Imprime las creencias, deseos y argumentos de \textit{Tomkins} después de la
        ↪ actualización.
    io.print("= Creencias del agente \textit{Tomkins}: =\n", !IO),
    io.print(\textit{Tomkins}Updated^beliefs, !IO),
    io.print("\n\n", !IO),

    io.print("= Deseos del agente \textit{Tomkins}: =\n", !IO),
    io.print(\textit{Tomkins}Updated^desires, !IO),
    io.print("\n\n", !IO),

    io.print("= Intenciones (argumentos) del agente \textit{Tomkins}: =\n", !IO),
    \textit{Tomkins}Arguments = list.map(argument_name, \textit{Tomkins}Updated^arguments
        ↪ ),
    io.print(\textit{Tomkins}Arguments, !IO),
    io.print("\n\n", !IO),

    % Aplica un ataque y reevalúa las creencias del agente.
    Attack1 = attack1a,
    apply_attack(\textit{Tomkins}Updated, Attack1, \textit{Tomkins}UpdatedAttack),

    % Reevalúa los argumentos después del ataque.
    \textit{Tomkins}Reevaluated = add_arguments_to_\textit{Tomkins}(\textit{Tomkins}
        ↪ UpdatedAttack),
    \textit{Tomkins}ReevaluatedArguments = list.map(argument_name, \textit{Tomkins}
        ↪ Reevaluated^arguments),

    % Imprime el estado de \textit{Tomkins} después de reevaluar el ataque.
    io.print("= Creencias de \textit{Tomkins} después de la reevaluación: =\n", !IO),
    io.print(\textit{Tomkins}Reevaluated^beliefs, !IO),
    io.print("\n\n", !IO),

    io.print("= Deseos de \textit{Tomkins} después de la reevaluación: =\n", !IO),
    io.print(\textit{Tomkins}Reevaluated^desires, !IO),
    io.print("\n\n", !IO),

    io.print("= Intenciones (argumentos) de \textit{Tomkins} después de la reevaluación:
        ↪ =\n", !IO),
    io.print(\textit{Tomkins}ReevaluatedArguments, !IO),
    io.print("\n\n", !IO).
```

Listing 13: Aplicación del modelo del caso *Pierson v. Post* en Mercury