INTELIGENCIA ARTIFICIAL

UNIDAD 1: INTELIGENCIA ARTIFICIAL Y AGENTES INTELIGENTES

1.1. INTELIGENCIA ARTIFICIAL

Que es la IA

Las definiciones que aparecen en la parte superior se refieren a procesos mentales y al razonamiento, mientras que las de la parte inferior aluden a la conducta. Las definiciones de la izquierda miden el éxito en términos de la fidelidad en la forma de actuar de los humanos, mientras que las de la derecha toman como referencia un concepto ideal de inteligencia, que llamaremos racionalidad. Un sistema es racional si hace «lo correcto», en función de su conocimiento. El enfoque centrado en el comportamiento humano debe ser una ciencia empírica, que incluya hipótesis y confirmaciones mediante experimentos. El enfoque racional implica una combinación de matemáticas e ingeniería.

Sistemas que piensan como humanos	**Sistemas que piensan racionalmente**
«El nuevo y excitante esfuerzo de hacer que los computadores piensen… máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985)
«[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje…» (Bellman, 1978)	«El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)
Sistemas que actúan como humanos	**Sistemas que actúan racionalmente**
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990)	«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole *et al.*, 1998)
«El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	«IA… está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)

Figura 1.1 Algunas definiciones de inteligencia artificial, organizadas en cuatro categorías.

Comportamiento Humano: la prueba de Turing

La **Prueba de Turing**, propuesta por Alan Turing (1950), se diseñó para proporcionar una definición operacional y satisfactoria de inteligencia. En vez de proporcionar una lista de cualidades necesarias para obtener inteligencia artificialmente, él sugirió una prueba basada en la incapacidad de diferenciar entre entidades inteligentes indiscutibles y seres humanos. El computador supera la prueba si un evaluador humano no es capaz de distinguir si las respuestas, a una serie de preguntas planteadas, son de una persona o no. Programar un computador para que supere la prueba requiere un trabajo considerable porque el computador debería poseer las siguientes capacidades:

1. **Procesamiento de lenguaje natural** que le permita comunicarse satisfactoriamente en inglés.
2. **Representación del conocimiento** para almacenar lo que se conoce o siente.
3. **Razonamiento automático** para utilizar la información almacenada para responder a preguntas y extraer nuevas conclusiones.
4. **Aprendizaje automático** para adaptarse a nuevas circunstancias y para detectar y extrapolar patrones.

La Prueba de Turing evitó deliberadamente la interacción física directa entre el evaluador y el computador, dado que para medir la inteligencia es innecesario simular físicamente a una persona. La Prueba Global de Turing incluye una

Comentado [VCNC1]: Estos 4 titulitos son la explicacion de la clasificacion del cuadro de arriba, lo dejaría pero capaz se puede resumir mas

señal de vídeo que permite al evaluador valorar la capacidad de percepción del evaluado, y también le da la oportunidad al evaluador de pasar objetos físicos. Para superar la Prueba Global de Turing el computador debe estar dotado de

5. **Visión computacional** para percibir objetos.
6. **Robótica** para manipular y mover objetos.

Estas seis disciplinas abarcan la mayor parte de la IA.

Pensar como Humano: el enfoque cognitivo

Para poder decir que un programa dado piensa como un humano, es necesario contar con un mecanismo para determinar cómo piensan los humanos. Se puede hacer mediante introspección (intentando atrapar nuestros propios pensamientos conforme éstos van apareciendo) y mediante experimentos psicológicos. Una vez se cuente con una teoría lo suficientemente precisa sobre cómo trabaja la mente, se podrá expresar esa teoría en la forma de un programa de computador. Si los datos de entrada/salida del programa y los tiempos de reacción son similares a los de un humano, existe la evidencia de que algunos de los mecanismos del programa se pueden comparar con los que utilizan los seres humanos. En el campo interdisciplinario de la **ciencia cognitiva** convergen modelos computacionales de IA y técnicas experimentales de psicología intentando elaborar teorías precisas y verificables sobre el funcionamiento de la mente humana.

Pensamiento Racional: el enfoque de las Leyes del Pensamiento

Aristóteles intentó codificar la «manera correcta de pensar», es decir, un proceso de razonamiento irrefutable. Sus **silogismos** son esquemas de estructuras de argumentación mediante las que siempre se llega a conclusiones correctas si se parte de premisas correctas. Estas leyes de pensamiento supuestamente gobiernan la manera de operar de la mente; su estudio fue el inicio de un campo llamado **lógica**.

Estudiosos de la lógica desarrollaron una notación precisa para definir sentencias sobre todo tipo de elementos del mundo y especificar relaciones entre ellos. Ya en 1965 existían programas que, en principio, resolvían *cualquier* problema resoluble descrito en notación lógica. La llamada tradición **logista** dentro del campo de la inteligencia artificial trata de construir sistemas inteligentes a partir de estos programas.

Actuar de Forma Racional: el enfoque del agente racional

Comentado [VCNC2]: Esto se lo trata en la parte de agentes inteligentes me parecio medio demás

Un **agente** es algo que razona. Pero de los agentes informáticos se espera que tengan otros atributos que los distingan de los programas convencionales (controles autónomos, percepción del entorno, persistencia en el tiempo, adaptación a cambios, que sean capaces de alcanzar objetivos diferentes).

En el caso del enfoque de la IA según las «leyes del pensamiento», todo el énfasis se pone en hacer inferencias correctas. La obtención de estas inferencias correctas puede formar *parte* de lo que se considera un agente racional, ya que una manera racional de actuar es llegar a la conclusión lógica de que, si una acción dada permite alcanzar un objetivo, hay que llevar a cabo dicha acción. Sin embargo, el efectuar una inferencia correcta no depende siempre de la *racionalidad*, ya que existen situaciones para las que no hay nada correcto que hacer y en las que hay que tomar una decisión. Existen también formas de actuar racionalmente que no implican realizar inferencias.

Todas las habilidades que se necesitan en la Prueba de Turing deben permitir emprender acciones racionales. Por lo tanto, es necesario contar con la capacidad para representar el conocimiento y razonar basándonos en él, porque ello permitirá alcanzar decisiones correctas en una amplia gama de situaciones.

¿Por qué es interesante la IA?

Sentido de la Identidad

El hombre se ha aplicado a sí mismo el nombre científico de ***homo sapiens*** (hombre sabio o "capaz de conocer") como una valoración de la trascendencia de nuestras habilidades mentales tanto para la vida cotidiana como en nuestro propio sentido de la identidad.

Comprensión de Entidades Inteligentes

Los esfuerzos en el campo de la IA se enfocan en lograr la compresión de entidades inteligentes. Por eso, una de las razones de su estudio es aprender más sobre nosotros mismos. A diferencia de la filosofía y de la psicología, que también se ocupan de la inteligencia, los esfuerzos de la IA están encaminados tanto a la comprensión de entidades inteligentes como su construcción. Otra razón por la que se estudia la IA es que las entidades inteligentes así construidas son interesantes y útiles por derecho propio.

Es un problema Complejo pero

El problema que aborda la IA es uno de los más complejos, pero a diferencia de la investigación en torno al desplazamiento a mayor velocidad que la luz o de un dispositivo antigravitatorio, el investigador del campo de la IA cuenta con pruebas contundentes de que tal búsqueda es factible. Todo lo que este investigador tiene que hacer es mirarse en un espejo para tener ante sí un ejemplo de sistema inteligente.

El estudio de la inteligencia es una de las disciplinas más antiguas

Por más de 2000 años los filósofos se han preguntado cómo se ve, aprende, recuerda y razona. La llegada de las computadoras a principio de la década de los cincuenta permitió pasar de la especulación, a nivel de charlas de café, en torno a estas facultades mentales a su abordaje mediante una auténtica disciplina teórica y experimental.

Fundamentos (De otras ciencias)

Comentado [VCNC3]: Aca quizas se podría resumir/sacar alguna de las disciplinas

Si bien la IA es un campo joven, es heredera de diversas ideas, puntos de vista y técnicas de otras disciplinas.

Disciplina	Aporte
Filosofía	• **Modelos de razonamiento y lógica:** Los filósofos, como Aristóteles, sentaron las bases para la lógica formal y el razonamiento deductivo, principios que son fundamentales para el diseño de sistemas de razonamiento en la IA, como sistemas expertos y sistemas de inferencia. • **Dualismo y materialismo:** Influyeron en las concepciones de cómo la mente y la inteligencia podrían ser representadas y replicadas en máquinas. Dada una mente física que gestiona conocimiento, el siguiente problema es establecer las fuentes de este conocimiento. • **Movimiento empírico:** proporcionó enfoques y conceptos clave para abordar la adquisición de conocimiento, el razonamiento y la toma de decisiones en los sistemas de IA. • **Principio de inducción:** influyó en el desarrollo de sistemas de aprendizaje automático y la toma de decisiones basada en datos. La inducción es un proceso mediante el cual se infieren patrones generales o reglas a partir de ejemplos específicos. • **Positivismo Lógico:** esta doctrina sostiene que todo el conocimiento se puede caracterizar mediante teorías lógicas relacionadas, en última instancia, con **sentencias de observación** que corresponden a estímulos sensoriales. • **Teoría de la Confirmación:** intenta explicar cómo el conocimiento se • obtiene a partir de la experiencia.
Matemáticas	Para pasar a una ciencia formal se necesitaba una formulación matemática en: 1. Lógica: • Lógica proposicional o Booleana • Lógica de Primer Orden: extiende la lógica de Boole para incluir objetos y relaciones. Se utiliza hoy como el sistema más básico de representación de conocimiento. • Teoría de Referencia: enseña cómo relacionar objetos de una lógica con objetos del • mundo real. 2. Computación: • Avances en Algoritmos. • Teorema de Incompletitud: en cualquier lenguaje con capacidad suficiente para expresar las propiedades de los números naturales, existen aseveraciones verdaderas de las cuales no es posible decidir su validez mediante ningún algoritmo. • Intratabilidad: un problema es intratable si el tiempo necesario para la resolución de casos particulares de dicho problema crece exponencialmente con el tamaño de dichos casos. • NP-Completitud: es un método para reconocer problemas intratables. 3. Probabilidad
Economía	• Teoría de la decisión • Teoría de Juegos

	• Investigación Operativa • Satisfacción.
Neurociencia	Es el estudio del sistema neurológico, y en especial del cerebro. En la actualidad se dispone de información sobre la relación existente entre las áreas del cerebro y las partes del cuerpo humano que controlan o de las que reciben impulsos sensoriales.
Psicología	• **Conductismo:** estudio exclusivo de mediciones objetivas de percepciones (o *estímulos*) sobre animales y de las acciones resultantes (o *respuestas*). Construcciones mentales como conocimientos, creencias, objetivos y pasos en un razonamiento quedaron descartadas por ser consideradas «psicología popular» no científica. • **Psicología cognitiva:** su característica es la conceptualización del cerebro como un dispositivo de procesamiento de información. • **Ciencia Cognitiva:** surge del desarrollo del modelo computacional. Se presentaron artículos que mostraban cómo se podían utilizar los modelos informáticos para modelar la psicología de la memoria, el lenguaje y el pensamiento lógico, respectivamente.
Computación	Para que la inteligencia artificial pueda llegar a ser una realidad se necesitan inteligencia y un artefacto: el computador. Desde mediados del siglo pasado, cada generación de dispositivos *hardware* ha conllevado un aumento en la velocidad de proceso y en la capacidad de almacenamiento, asi como una reducción de precios.
Teoría de Control	Se visualiza el comportamiento determinista como algo emergente de un mecanismo regulador que intenta minimizar el error (la diferencia entre el estado presente y el estado objetivo). La teoría de control moderna tiene como objetivo el diseño de sistemas que maximizan una **función objetivo** en el tiempo. Lo cual se asemeja ligeramente a nuestra visión de lo que es la IA: diseño de sistemas que se comportan de forma óptima.
Lingüística	La lingüística moderna y la IA «nacieron», al mismo tiempo y maduraron juntas, solapándose en un campo híbrido llamado **lingüística computacional** o **procesamiento del lenguaje natural**. El entendimiento del lenguaje requiere la comprensión de la materia bajo estudio y de su contexto, y no solamente el entendimiento de la estructura de las sentencias.

Breve historia y algunos hitos

Año	Avance
1936	Alan Turing crea la “Máquina de Turing”, la base de las computadoras.
1943	Warren McCulloch y Walter Pitts son considerados pioneros en la IA, desarrollando un modelo de neuronas artificiales basado en conocimientos de fisiología cerebral, lógica proposicional y teoría de la computación de Turing. Crearon un modelo de neuronas que podían estar "activadas" o "desactivadas", con activación estimulada por otras neuronas. Vieron el estado de una neurona como equivalente a una proposición con estímulos adecuados. Demostraron que las funciones de cómputo y los conectores lógicos podían implementarse con redes de neuronas.
1949	Donald Hebb propuso la regla de aprendizaje Hebbiano para ajustar conexiones neuronales según la estimulación, idea que sigue vigente.
1950	Presentó la prueba de Turing (una prueba para determinar si una máquina es inteligente como un humano), el aprendizaje automático, los algoritmos genéricos y el aprendizaje por refuerzo.
1951	Marvin Minsky y Dean Edmonds construyeron el primer computador basado en red neuronal en 1951, llamado SNARC.
1952	McCarthy organiza un taller en Darmouth a la que asistieron científicos que trabajaban en temas relacionados con "máquinas pensantes". Allen Newell y Herbert Simon captaron la atención con su programa de razonamiento llamado Teórico Lógico (TL), que demostró teoremas matemáticos. Acá se acuñó el nombre de “Inteligencia Artificial”.
1956-1969	Optimismo exagerado. Se resolvieron problemas como la demostración de teoremas y se logró que las máquinas superaran a los humanos en algunos juegos simples.
1966-1973	Se intentaron atacar problemas del dominio del sentido común o aquellos que necesitan de conocimiento general, como los relacionados con el lenguaje, los esfuerzos no fueron suficientes. Esta situación llevó a un desanimo generalizado y a la pérdida de inversiones.
1969-1979	Se centró en el desarrollo de métodos de búsqueda generales y en el uso de conocimiento específico del dominio para resolver problemas más complejos.
1980 – Presente	La IA resurgió de la mano de los Sistemas Expertos. Desde ese momento el área no deja de crecer. Los sistemas de visión artificial y los de reconocimiento de patrones han provisto innumerables soluciones en los últimos años. La cantidad de datos disponibles, los recursos computacionales y los

	avances como el de las redes neuronales profundas (*deep learning*) han formado un círculo virtuoso donde tanto las empresas como la comunidad científica invierten cada vez más recursos.

Corrientes

Desde los inicios de la IA se plantearon dos enfoques diferentes, uno por parte de los investigadores del MIT y otro por los del Carnegie-Mellon. Estos caminos se fueron alejando hasta llegar a plantearse un distanciamiento que caracterizó las investigaciones en IA de los veinte años siguientes y que fue sin lugar a dudas muy perjudicial por el atraso que provocó en algunas líneas de investigación.

Corriente conexionista - subsimbólica - ascendente.

La escuela liderada por los investigadores de la Universidad de Carnegie-Mellon proponía desarrollar modelos del comportamiento humano a partir de elementos que simularan en todo lo posible al cerebro con un enfoque claramente cognitivo. Así iniciaron una corriente denominada "de aproximaciones subsimbólicas" que sigue un diseño de modelo "ascendente", comenzando en el nivel más bajo y operando hacia niveles superiores. Según esta corriente, para concebir máquinas inteligentes hay que seguir muchos de los pasos evolutivos que acompañaron el desarrollo de la inteligencia en los seres vivos. De esta forma hay que comenzar por replicar la capacidad de procesamiento de señales y subir por la cadena evolutiva en pasos sucesivos, construyendo progresivos niveles superiores de inteligencia que constituirán las bases sólidas para los pasos siguientes. Al amparo de esta propuestase desarrollaron los modelos conexionistas, redes neuronales inspiradas en los modelos biológicos.

Corriente tradicional - simbólica - descendente.

La segunda escuela, liderada por Minsky y McCarthy del MIT orientó su actividad a obtener comportamientos "inteligentes" sin procurar que la estructura de los componentes tuviese semejanza con sus equivalentes biológicos y llegó a su mayor auge en los primeros 20 años de la IA. Esta escuela iniciada en el MIT fue denominada "tradicional", "de procesamiento de símbolos" o "de diseño descendente" y se apoyó en representar el conocimiento mediante sentencias declarativas y en la deducción de sus consecuencias a partir de la aplicación de reglas de inferencia. Sin embargo, poco a poco se fue comprobando que muchas de las soluciones propuestas eran apropiadas para modelos pequeños, pero completamente ineficientes o inaplicables cuando se pretendía resolver problemas de tamaño real. Los investigadores se encontraron con que sus sistemas sucumbían ante la creciente longitud y complejidad de su programación.

1.2. AGENTES INTELIGENTES

Agentes y su entorno

Un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores. Se trabajará con la hipótesis general de que cada agente puede percibir sus propias acciones (pero no siempre sus efectos).

El término **percepción** se utiliza para indicar que el agente puede recibir entradas en cualquier instante. La secuencia de percepciones refleja el historial completo de lo que el agente ha recibido. Este tomará una decisión en un momento dado dependiendo de la secuencia completa de percepciones hasta ese instante.

En términos matemáticos se puede decir que el **comportamiento** del agente viene dado por la **función del agente** que proyecta una percepción dada en una acción. Esta función se puede presentar en forma de tabla.

Inicialmente, la **función del agente** para un agente artificial se implementará mediante el **programa del agente**. La primera es una **descripción matemática abstracta**, mientras que la segunda es una **implementación completa**, que se ejecuta sobre la arquitectura del agente.

Concepto de racionalidad

Un agente racional es aquel que hace lo correcto. Se puede decir que lo correcto es aquello que permite al agente obtener un resultado mejor. Por tanto, se necesita determinar una forma de medir el éxito, junto a la descripción del entorno y de los sensores y actuadores del agente.

Medidas de rendimiento

Las medidas de rendimiento incluyen los **criterios** que determinan el **éxito** en el comportamiento del agente. Cuando se sitúa un agente en un medio, éste genera una **secuencia de acciones** de acuerdo con las percepciones que recibe, esta secuencia hace que su hábitat pase por una **secuencia de estados**. Si la secuencia es la deseada, entonces el agente habrá actuado correctamente.

Como regla general, es mejor diseñar medidas de utilidad de acuerdo con lo que se quiere para el entorno, más que de acuerdo con cómo se cree que el agente debe comportarse.

Racionalidad

La **racionalidad** en un momento determinado depende de **cuatro factores**:

- La medida de rendimiento que define el criterio de éxito.
- El conocimiento del medio en el que habita acumulado por el agente.
- Las acciones que el agente puede llevar a cabo.
- La secuencia de percepciones del agente hasta este momento

Esto nos lleva a la definición de **agente racional**: En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.

Omnisciencia, aprendizaje y autonomía

Es necesario distinguir entre racionalidad y omnisciencia. Un **agente omnisciente** conoce el resultado de su acción y actúa de acuerdo con él, pero esto no es posible. Un ejemplo es si quiero cruzar la calle y cae una puerta de un avión cuando estoy cruzando, cruzar no fue irracional. La racionalidad no es lo mismo que la perfección. La **racionalidad** maximiza el rendimiento esperado, mientras la perfección maximiza el resultado real.

El asunto es que resulta imposible diseñar un agente que siempre lleve a cabo, de forma sucesiva, las mejores acciones después de un acontecimiento. La definición propuesta de racionalidad no requiere omnisciencia, ya que la elección racional depende sólo de la secuencia de percepción hasta la fecha.

Llevar a cabo acciones con la intención de modificar percepciones futuras, proceso denominado **recopilación de información**, es una parte importante de la racionalidad. Por ejemplo, <mirar> ambos lados antes de cruzar la calle. Un segundo ejemplo de recopilación de información lo proporciona la exploración que debe llevar a cabo el agente aspiradora en un medio inicialmente desconocido. Esto implica que el agente racional no sólo recopile información, sino que **aprenda** lo máximo posible de lo que está percibiendo.

La configuración inicial del agente puede reflejar un **conocimiento** preliminar del entorno, pero a medida que el agente adquiere experiencia éste puede modificarse y aumentar. Hay casos excepcionales en los que se conoce totalmente el entorno a priori, en estos casos no percibe ni aprende, simplemente actúa de forma correcta, pero son frágiles a cambios en su entorno.

Los agentes con éxito dividen las tareas de **calcular la función del agente** en tres períodos:

- Cuando se está **diseñando** el agente, los diseñadores están encargados de realizar algunos de estos cálculos
- Cuando está **pensando** en la siguiente operación, el agente realiza más cálculos
- Cuando está **aprendiendo** de la experiencia, el agente lleva a cabo más cálculos para decidir cómo modificar su forma de comportarse

Un agente carece de **autonomía** cuando se apoya más en el conocimiento inicial que en sus propias percepciones. Un agente racional debe ser autónomo, debe saber aprender a determinar cómo tiene que compensar el conocimiento incompleto o parcial inicial. Por ejemplo, que la aspiradora aprenda donde y cuando aparecerá suciedad adicional.

Pocas veces se necesita autonomía completa desde el comienzo, pero es razonable proporcionar a los agentes que disponen de inteligencia artificial un **conocimiento inicial**, así como de la **capacidad de aprendizaje**. Después de las suficientes experiencias interaccionando con el entorno, el comportamiento del agente racional será efectivamente independiente del conocimiento que poseía inicialmente.

La naturaleza del entorno

Los entornos de trabajo son los «**problemas**» para los que los agentes racionales son las «**soluciones**»

Especificación del entorno de trabajo

En el diseño de un agente, el primer paso debe ser siempre especificar el entorno de trabajo de la forma más completa posible.

Propiedades de los entornos de trabajo

Se puede identificar un pequeño número de dimensiones en las que categorizar estos entornos. Estas dimensiones determinan, hasta cierto punto, el diseño más adecuado para el agente y la utilización de cada una de las familias principales de técnicas en la implementación del agente.

- **Totalmente observable vs Parcialmente observable**
  Si los sensores del agente le proporcionan acceso al estado completo del medio en cada momento es **totalmente observable**
- **Determinista vs Estocástico**
  Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente es **determinista**
- **Episódico vs Secuencial**
  La experiencia del agente se divide en episodios que consisten en la percepción del agente y la realización de una única acción posterior. Si la elección de la acción en cada episodio depende sólo del episodio en sí mismo es **episódico**
- **Dinámico vs Estático**
  Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es **dinámico**
- **Discreto vs Continuo**
  La distinción entre discreto y continuo se puede aplicar al estado del medio, a la forma en la que se maneja el tiempo y a las percepciones y acciones del agente. Por ejemplo, un medio con estados **discretos** como el del juego del ajedrez tiene un número finito de estados distintos. Por otro lado, el taxista define un estado **continuo**.
- **Agente individual vs Multiagente**
  Un agente resolviendo un crucigrama por sí mismo está claramente en un entorno de agente **individual**, mientras que un agente que juega al ajedrez está en un entorno con dos agentes. Hay que analizar que entidades deben considerarse como agentes, la clave está en identificar si el comportamiento de B está mejor descrito por la maximización de una medida de rendimiento cuyo valor depende del comportamiento de A. En el ajedrez, la entidad oponente B intenta maximizar su medida de rendimiento, la cual, minimiza la medida de rendimiento del agente A, por tanto, es un entorno **multiagente competitivo**. En el medio del taxista, el evitar colisiones maximiza la medida de rendimiento de todos los agentes siendo un entorno **multiagente parcialmente cooperativo**.

Estructura de los agentes

El trabajo de la IA es diseñar el **programa del agente** que implemente la **función del agente** que proyecta las percepciones en las acciones, éste se ejecutará en algún tipo de computador con sensores físicos y actuadores, lo cual se conoce como arquitectura:

Agente = arquitectura + programa

La arquitectura hace que las percepciones de los sensores estén disponibles para el programa, ejecuta los programas, y se encarga de que los actuadores pongan en marcha las acciones generadas.

Programas de los agentes

Los programas de los agentes reciben las **percepciones** actuales como entradas de los **sensores** y devuelven una **acción** a los **actuadores**. Hay que tener en cuenta la diferencia entre los **programas** de los agentes, que toman la **percepción actual** como entrada, y la **función** del agente, que recibe la **percepción histórica** completa.

Agente dirigido mediante tabla: almacena la secuencia de percepciones y después las compara con las secuencias almacenadas en la tabla de acciones para decidir qué hacer. La tabla representa explícitamente la función que define

el programa del agente. Para construir un agente racional de esta forma, los diseñadores deben realizar una tabla que contenga las acciones apropiadas para cada secuencia posible de percepciones.

Cuatro **tipos básicos de programas** para agentes:

- Agentes reactivos simples
- Agentes reactivos basados en modelos
- Agentes basados en objetivos
- Agentes basados en utilidad

Agentes reactivos simples

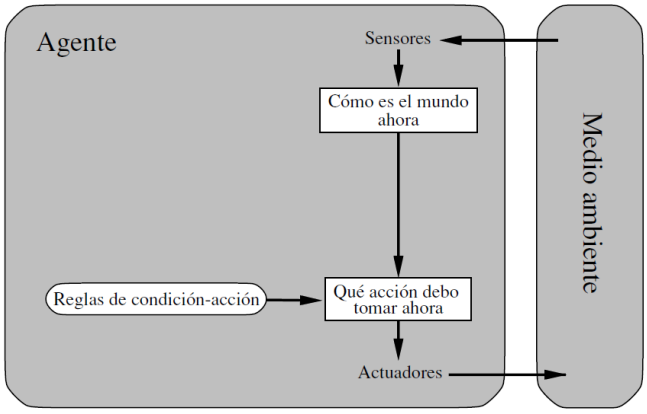
El tipo de agente más sencillo es el agente reactivo simple. Estos agentes seleccionan las **acciones** sobre la base de las **percepciones actuales**, ignorando el resto de las percepciones históricas. Se basa en conexiones establecidas en el programa del agente para que se ejecute una acción. Esta conexión se denomina regla de condición-acción, y se representa por:

sí <percepción> entonces <acción>

función AGENTE-ASPIRADORA-REACTIVO([*localización*, *estado*]) **devuelve** una acción

si *estado* = *Sucio* **entonces devolver** *Aspirar*
de otra forma, si *localización* = *A* **entonces devolver** *Derecha*
de otra forma, si *localización* = *B* **entonces devolver** *Izquierda*

Una aproximación más general y flexible es la de construir primero un intérprete de propósito general para reglas de condición-acción y después crear conjuntos de reglas para entornos de trabajo específicos



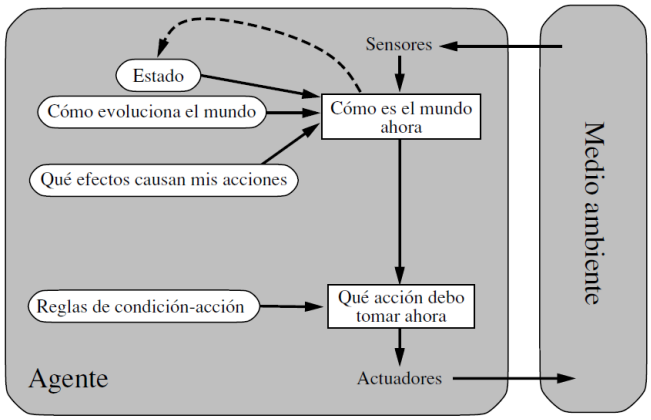
La función INTERPRETAR-ENTRADA genera una descripción abstracta del estado actual a partir de la percepción, y la función REGLA-COINCIDENCIA devuelve la primera regla del conjunto de reglas que coincide con la descripción del estado dada. Hay que tener en cuenta que la descripción en términos de «reglas» y «coincidencias» es puramente conceptual, las implementaciones reales pueden otras.

Los **bucles infinitos** son a menudo inevitables para los agentes reactivos simples que operan en algunos **entornos parcialmente observables**. Salir de los bucles infinitos es posible si los agentes pueden seleccionar sus acciones aleatoriamente. Por ejemplo, si un agente aspiradora percibe [Limpio], puede lanzar una moneda y elegir entre Izquierda y Derecha. Este comportamiento puede resultar racional en algunos entornos multiagente pero no en entornos de agentes individuales.

Agentes reactivos basados en modelos

Para manejar la visibilidad parcial el agente debe mantener algún tipo de **estado interno** que dependa de la historia percibida y que de ese modo refleje por lo menos alguno de los **aspectos no observables** del estado actual. Primero, se necesita alguna información acerca de cómo **evoluciona el mundo** independientemente del agente. Segundo, se necesita más información sobre cómo **afectan al mundo** las acciones del agente.

Este conocimiento acerca de «cómo funciona el mundo», tanto si está implementado con un circuito booleano simple o con teorías científicas completas, se denomina modelo del mundo. Un agente que utilice este modelo es un agente basado en modelos.



función AGENTE-REACTIVO-CON-ESTADO(percepción) **devuelve** una acción
estático: *estado*, una descripción actual del estado del mundo
reglas, un conjunto de reglas condición-acción
acción, la acción más reciente, inicialmente ninguna

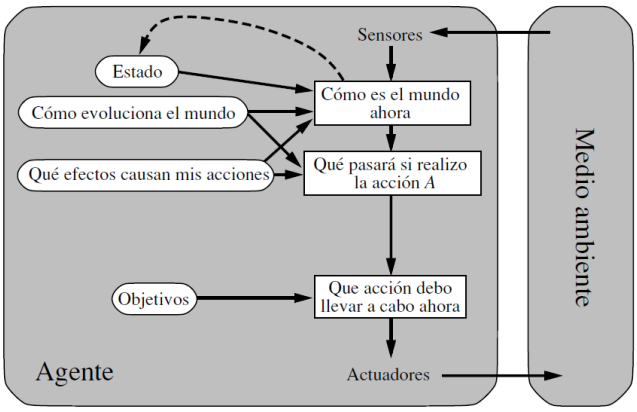
estado ← ACTUALIZAR-ESTADO(*estado*, *acción*, *percepción*)
regla ← REGLA-COINCIDENCIA(*estado*, *reglas*)
acción ← REGLA-ACCIÓN[*regla*]
devolver *acción*

Un agente reactivo basado en modelos, que almacena información sobre el estado actual del mundo utilizando un modelo interno. Después selecciona una acción de la misma forma que el agente reactivo.

Agentes basados en objetivos

El conocimiento sobre el estado actual del mundo no es siempre suficiente, por ejemplo, en un cruce, el taxista puede girar a la izquierda, derecha o seguir, pero esto depende de a donde quiere ir. Además de la descripción del **estado actual**, el agente necesita algún tipo de información sobre su **meta** que describa las situaciones que son deseables.

En algunas ocasiones, la selección de acciones basadas en objetivos es directa, cuando alcanzar los objetivos es el resultado inmediato de una acción individual. En otras ocasiones, puede ser más complicado, cuando el agente tiene que considerar secuencias complejas para encontrar el camino que le permita alcanzar el objetivo



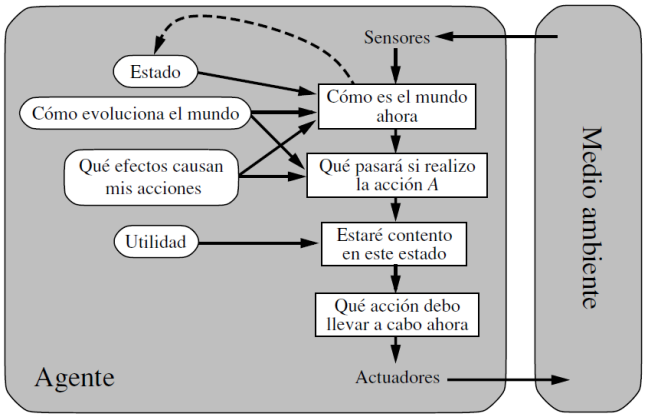
La toma de decisiones de este tipo es diferente de las reglas de condición–acción descritas anteriormente ya que hay que tener en cuenta consideraciones sobre el **futuro**.

Aunque el agente basado en objetivos pueda parecer menos eficiente, es más flexible ya que el conocimiento que soporta su decisión está representado explícitamente y puede modificarse.

Agentes basados en utilidad

Las metas por sí solas no son realmente suficientes para generar comportamiento de gran calidad en la mayoría de los entornos. El taxi puede llegar a destino por varias rutas, pero algunas pueden ser más rápidas, seguras, fiables o baratas que otras. Para indicar que se prefiere un estado del mundo a otro es que un **estado** tiene más **utilidad** que otro para el agente.

Una función de utilidad proyecta un estado (o una secuencia de estados) en un número real, que representa un nivel de felicidad. Cuando hay **objetivos en conflicto**, por ejemplo, velocidad y seguridad, la utilidad determina un **equilibrio** para cumplir ambos. Cuando hay varios objetivos y el cumplimiento de ellos es incierto, la utilidad pondera la **probabilidad de éxito** en función de la importancia de los objetivos.

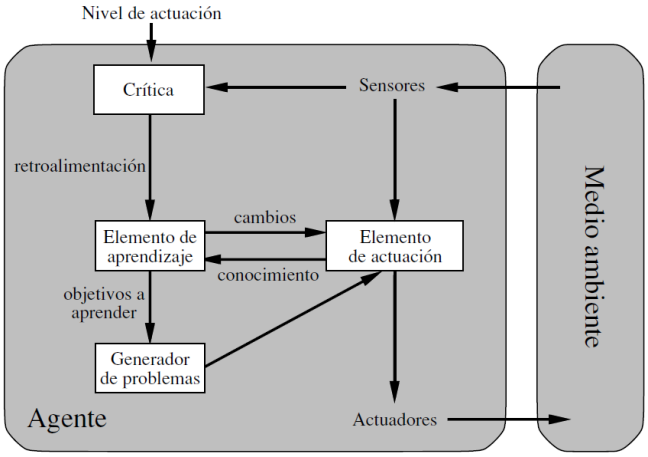


Un agente basado en utilidad y basado en modelos. Utiliza un modelo del mundo, junto con una función de utilidad que calcula sus preferencias entre los estados del mundo. Después selecciona la acción que le lleve a alcanzar la mayor utilidad esperada, que se calcula haciendo la media de todos los estados resultantes posibles, ponderado con la probabilidad del resultado.

Agentes que aprenden

Turing (1950) consideró la idea de programar sus máquinas inteligentes a mano. El método que propone es construir máquinas que aprendan y después enseñarlas. El aprendizaje tiene otras ventajas, como se ha explicado anteriormente: permite que el agente opere en medios inicialmente desconocidos y que sea más competente que si sólo utilizase un conocimiento inicial.

Un agente que aprende se puede dividir en cuatro componentes conceptuales, tal y como se muestra en la figura.



La distinción más importante entre el **elemento de aprendizaje** y el **elemento de actuación** es que el primero está responsabilizado de hacer **mejoras** y el segundo se responsabiliza de la **selección de acciones** externas.

El elemento de aprendizaje se realimenta con las **críticas** sobre la actuación del agente con respecto a un **nivel de actuación fijo** y determina cómo se debe **modificar el elemento de actuación** para proporcionar mejores resultados en el futuro. Dado un diseño para un agente, se pueden construir los mecanismos de aprendizaje necesarios para mejorar cada una de las partes del agente.

El último componente del agente con capacidad de aprendizaje es el **generador de problemas**. El agente puede llevar a cabo nuevas acciones que no sean totalmente óptimas a corto plazo, pero puede **descubrir acciones** mejores a largo plazo. El generador de problemas es responsable de sugerir estas acciones exploratorias.

Unidad 2: Razonamiento en ambientes deterministas I

2.1. Agentes resolventes-problemas

El ambiente más simple donde se puede desempeñar un agente inteligente es Totalmente observable, Determinista, Episódico, Estático, Discreto, Agente individual y Conocido.

Se supone que los agentes inteligentes deben maximizar su medida de rendimiento. Esto puede simplificarse algunas veces si el agente puede elegir **un objetivo** y trata de satisfacerlo.

Los objetivos ayudan a organizar su comportamiento limitando las metas que intenta alcanzar el agente. El primer paso para solucionar un problema es la **formulación del objetivo**, basado en la situación actual y la medida de rendimiento del agente.

Consideraremos un **objetivo** como un conjunto de **estados del mundo** (exactamente aquellos estados que satisfacen el objetivo). La tarea del agente es encontrar qué **secuencia de acciones** permite obtener un **estado objetivo**. Para esto, necesitamos decidir qué acciones y estados considerar. Dado un objetivo, la **formulación del problema** es el proceso de decidir qué acciones y estados tenemos que considerar.

Una vez definido el objetivo, el agente no sabrá cuál de las posibles acciones es mejor, porque no conoce lo suficiente los estados que resultan al tomar cada acción. En general, un agente con distintas opciones inmediatas de valores desconocidos puede decidir qué hacer, examinando las diferentes **secuencias posibles** de acciones que le conduzcan a **estados** de valores **conocidos**, y entonces escoger la mejor secuencia.

Este proceso de hallar esta secuencia se llama **búsqueda**. Un algoritmo de búsqueda toma como entrada un **problema** y devuelve una **solución** de la forma secuencia de acciones. Una vez que encontramos una solución, se procede a ejecutar las acciones que ésta recomienda. Esta es la llamada fase de **ejecución**.

Problemas y Soluciones bien definidos

Un **problema** puede definirse formalmente por **cuatro componentes**:

1. El **estado inicial** en el que comienza el agente.
2. Una **descripción** de las posibles **acciones** disponibles por el agente. La formulación más común utiliza una **función sucesor**. Dado un estado particular *x*, SUCESOR-FN(*x*) devuelve un conjunto de **pares ordenados** *_acción, sucesor_*, donde cada acción es una de las **acciones en el estado *x*** y cada sucesor es un **estado alcanzable desde *x***, aplicando la acción.

   Implícitamente el estado inicial y la función sucesor definen el **espacio de estados** o espacio de búsqueda del problema (el conjunto de todos los estados que pueden ser alcanzados aplicando acciones a partir del estado inicial). El espacio de estados forma un **grafo** en el cual los nodos son estados y los arcos entre los nodos son acciones. Es importante entender que el espacio de estados existe solo como una posibilidad, no está disponible ni se lo conoce. Un camino en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones

   Un **camino** en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones.
3. El **test objetivo**, el cual determina si un estado es un estado objetivo. Existen dos posibilidades:
   a. Existe un conjunto explícito de posibles estados objetivo, y el test simplemente comprueba si el estado es uno de ellos.
   b. El objetivo se especifica como una propiedad abstracta más que como un conjunto de estados enumerados explícitamente.
4. Una función **costo del camino** que asigna un costo numérico a cada camino. El agente resolvente de problemas elige una función costo que refleje nuestra medida de rendimiento. Suponemos que:

Comentado [VCNC4]: Hay ejemplo de esto pero no los entendí

a. El costo del camino puede describirse como la suma de los costos de las acciones individuales a lo largo del camino, siendo el **costo individual** de una acción *a* que va desde un estado *x* al estado *y* denotado por *c*(*x*,*a*,*y*).

b. Los costos son no negativos.

Los elementos anteriores definen un problema y pueden unirse en una estructura de datos simple que se dará como entrada al algoritmo resolvente del problema. Una solución de un problema es un camino (**camino de la solución**) que contiene una sucesión de estados desde el estado inicial a un estado objetivo, donde cada estado puede ser generado aplicando una operación sobre el estado anterior. La calidad de la solución se mide por la función costo del camino, y una solución óptima tiene el costo más pequeño del camino entre todas las soluciones.

El **camino de búsqueda** es la sucesión de estados explorados por el método hasta encontrar la solución. No están conectados por acciones.

Generalmente, la salida esperada de un método de búsqueda es el camino entre el estado inicial y el estado objetivo (camino de la solución).

En algunos casos el estado mismo objetivo es la salida esperada (cinemática inversa, 8 reinas)

Formular los problemas

El proceso de **eliminar detalles** de una representación se le llama abstracción. Para una **descripción** de un estado puede ser En(lugar) en vez de considerar el tiempo, el estado de la carretera, etc. Además de abstraer la descripción del estado, debemos abstraer sus **acciones**. Por ejemplo, conducir solo cambia la localización en vez de considerar el gasto de combustible o la contaminación.

Búsqueda de Soluciones

Para implementar la búsqueda es necesario encontrar o crear:

- Una forma de representar el/los estado/s inicial, final e intermedios.
- Un conjunto de reglas compuestas por una condición de aplicación y una operación.
- Una estrategia de control para decidir el orden de aplicación de las reglas.

Resolvemos los problemas formulados mediante búsqueda a través del espacio de estados, con técnicas de búsqueda que utilizan un **árbol de búsqueda** explícito generado por el estado inicial y la función sucesor, definiendo así el espacio de estados. En general, podemos tener un grafo de búsqueda más que un árbol, cuando el mismo estado puede alcanzarse desde varios caminos.

La **raíz** del árbol de búsqueda es el **nodo de búsqueda** que corresponde al **estado inicial**. El primer paso es comprobar si éste es un estado **objetivo**. Si no estamos en un estado objetivo, tenemos que considerar otros estados. Esto se hace **expandiendo** el estado actual; es decir aplicando la **función sucesor al estado actual** y **generar** así un nuevo conjunto de estados. Ahora debemos escoger cuál de las posibilidades consideramos o si volvemos al paso anterior.

Esto es la esencia de la búsqueda, llevamos a cabo una opción y dejamos de lado las demás para más tarde, en caso de que la primera opción no conduzca a una solución. Continuamos escogiendo, comprobando y expandiendo hasta que se encuentra una solución o no existen más estados para expandir. El estado a expandir está determinado por la **estrategia de búsqueda**.

Hay muchas formas de representar los nodos, pero vamos a suponer que un **nodo** es una estructura de datos con **cinco componentes**:

- **Estado:** estado, del espacio de estados, que corresponde con el nodo.
- **Nodo Padre:** el nodo en el árbol de búsqueda que ha generado este nodo.
- **Acción:** la acción que se aplicará al padre para generar el nodo.
- **Costo del camino:** el costo, tradicionalmente denotado por *g*(*n*), de un camino desde el estado inicial al nodo, indicado por los punteros a los padres.

- **Profundidad:** el número de pasos a lo largo del camino desde el estado inicial.

Es importante distinguir:

- **Espacio de estados y árbol de búsqueda**: para el problema de búsqueda de un ruta, hay solamente 20 estados en el espacio de estados, uno por cada ciudad. Pero hay un número infinito de caminos en este espacio de estados, así que el árbol de búsqueda tiene un número infinito de nodos. Es importante recordar la distinción entre nodos y estados.
- **Nodo y Estado**: un nodo es una estructura de datos usada para representar el árbol de búsqueda. Un estado corresponde a una configuración del mundo. Así, los nodos están en caminos particulares, según lo definido por los punteros del nodo padre, mientras que los estados no lo están.

También necesitamos representar la colección de nodos que se han generado pero todavía no se han expandido – a esta colección se le llama **frontera**. Cada elemento de la frontera es un **nodo hoja**, es decir, un nodo sin sucesores en el árbol. La representación más simple de la frontera sería como un conjunto de nodos. La estrategia de búsqueda será una función que seleccione de este conjunto el siguiente nodo a expandir. Aunque esto sea conceptualmente sencillo, podría ser computacionalmente costoso, porque la función estrategia quizá tenga que mirar cada elemento del conjunto para escoger el mejor. Por lo tanto, nosotros asumiremos que la colección de nodos se implementa como una **cola**.

Las operaciones en una cola son como siguen:

Comentado [ML5]: No se que tan relevante sea pero siento que puede ser una pregunta general

— HACER-COLA(elemento, …) crea una cola con el(los) elemento(s) dado(s).

— VACIA?(cola) devuelve verdadero si no hay ningún elemento en la cola.

— PRIMERO(cola) devuelve el primer elemento de la cola.

— BORRAR-PRIMERO(cola) devuelve PRIMERO(cola) y lo borra de la cola.

— INSERTA(elemento,cola) inserta un elemento en la cola y devuelve la cola resultante. (Veremos que tipos diferentes de colas insertan los elementos en órdenes diferentes.)

— INSERTAR-TODO(elementos, cola) inserta un conjunto de elementos en la cola y devuelve la cola resultante.

Algoritmo general de búsqueda en árboles

```
función BÚSQUEDA-ÁRBOLES(problema,frontera) devuelve una solución o fallo
  frontera ← INSERTA(HACER-NODO(ESTADO-INICIAL[problema]), frontera)
  hacer bucle
    si VACIA?(frontera) entonces devolver fallo.
    nodo ← BORRAR-PRIMERO(frontera)
    si TEST-OBJETIVO[problema] aplicado al ESTADO[nodo] es cierto
      entonces devolver SOLUCIÓN(nodo)
    frontera ← INSERTAR-TODO(EXPANDIR(nodo,problema),frontera)
```

```
función EXPANDIR(nodo,problema) devuelve un conjunto de nodos
  sucesores ← conjunto vacío
  para cada (acción,resultado) en SUCESOR-FN[problema](ESTADO[nodo]) hacer
    s ← un nuevo NODO
    ESTADO[s] ← resultado
    NODO-PADRE[s] ← nodo
    ACCIÓN[s] ← acción
    COSTO-CAMINO[s] ← COSTO-CAMINO[nodo] + COSTO-INDIVIDUAL(nodo,acción,s)
    PROFUNDIDAD[s] ← PROFUNDIDAD[nodo] + 1
    añadir s a sucesores
  devolver sucesores
```

Medir el rendimiento de la resolución del problema

La salida del algoritmo de resolución de problemas es fallo o una solución.

El **rendimiento** de un algoritmo se puede evaluar de cuatro formas:

- **Completitud**: ¿está garantizado que el algoritmo encuentre una solución cuando esta exista?
- **Optimización**: ¿encuentra la estrategia la solución óptima?
- **Complejidad en tiempo**: ¿cuánto tarda en encontrar una solución?
- **Complejidad en espacio**: ¿cuánta memoria se necesita para el funcionamiento de la búsqueda?

La complejidad en tiempo y espacio siempre se considera con respecto a alguna medida de la dificultad del problema. En informática teórica, la medida es el tamaño del grafo del espacio de estados. En IA, donde el grafo está representado de forma implícita por el estado inicial y la función sucesor y frecuentemente es infinito, la complejidad se expresa en términos de tres cantidades:

- **b**, el factor de ramificación o el máximo número de sucesores de cualquier nodo
- **d**, la profundidad del nodo objetivo más superficial
- **m**, la longitud máxima de cualquier camino en el espacio de estados.

El **tiempo** a menudo se mide en términos de número de nodos generados durante la búsqueda, y el **espacio** en términos de máximo número de nodos que se almacena en memoria.

Para valorar la **eficacia** de un algoritmo de búsqueda, podemos considerar el **costo de la búsqueda** (que depende de la complejidad en tiempo, pero puede incluir también un término para el uso de la memoria) o podemos utilizar el **coste total**, que combina el **costo de la búsqueda** y el **costo del camino** solución encontrado.

Para el problema de encontrar una ruta desde Arad hasta Bucarest, el costo de la búsqueda es la cantidad de tiempo que ha necesitado la búsqueda y el costo de la solución es la longitud total en kilómetros del camino. Así, para el cálculo del coste total, tenemos que sumar kilómetros y milisegundos. No hay ninguna conversión entre los dos, pero quizá sea razonable, en este caso, convertir kilómetros en milisegundos utilizando una estimación de la velocidad media de un coche (debido a que el tiempo es lo que cuida el agente.) Esto permite al agente encontrar un punto óptimo de intercambio en el cual el cálculo adicional para encontrar que un camino más corto llegue a ser contraproducente.

Comentado [ML6]: No se si dejarlo o no

2.2. Búsqueda

Comentado [VC7]: Agregar los ejemplos de cada metodo que estan en la diapo

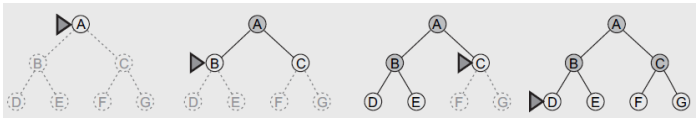
Estrategias de Búsqueda No Informada

El término significa que ellas no tienen información adicional acerca de los estados más allá de la que proporciona la definición del problema. Todo lo que ellas pueden hacer es generar los **sucesores** y distinguir entre un **estado objetivo** de uno que no lo es. Las estrategias que saben si un estado **no objetivo** es «**más prometedor**» que otro se llaman **búsqueda informada** o **búsqueda heurística**. Todas las estrategias se distinguen por el **orden de expansión** de los nodos.

Búsqueda primero en anchura/amplitud (FIFO)

Es una estrategia en la que se expande primero el nodo raíz. Se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo del próximo nivel.

Se puede implementar llamando a la BÚSQUEDA-ÁRBOLES(problema, COLA-FIFO()) con una frontera vacía que sea una cola FIFO asegurando que los nodos primeros visitados serán los primeros expandidos. La cola FIFO pone todos los nuevos sucesores generados al final de la cola, lo que significa que los nodos más superficiales se expanden antes que los nodos más profundos como se ve en la figura.



Evaluando el rendimiento de este algoritmo en base a los 4 aspectos mencionados anteriormente se puede decir:

- es **completa** (si el nodo objetivo más superficial está en una cierta profundidad finita d, la búsqueda primero en anchura lo encontrará después de expandir todos los nodos más superficiales, con tal que el factor de ramificación b sea finito)
- El nodo objetivo más superficial no es necesariamente el **óptimo**, lo es si el costo del camino es una función no decreciente de la profundidad del nodo (por ejemplo, cuando todas las acciones tienen el mismo coste)
- Tenemos que considerar la **cantidad de tiempo y memoria** que utiliza para completar una búsqueda. Considerando un espacio de estados hipotético donde cada estado tiene b sucesores. La raíz del árbol de búsqueda genera b nodos en el primer nivel, cada uno de ellos genera b nodos más y así sucesivamente. Ahora supongamos que la solución está a una profundidad d. En el peor caso, expandiremos todos excepto el último nodo en el nivel d (ya que el objetivo no se expande), generando $b^{d+1} - b$ nodos en el nivel d + 1. Entonces el número total de nodos generados es:

  $$b + b^2 + b^3 + \cdots + b^d + \left(b^{d+1} - b\right) = O\left(b^{d+1}\right)$$

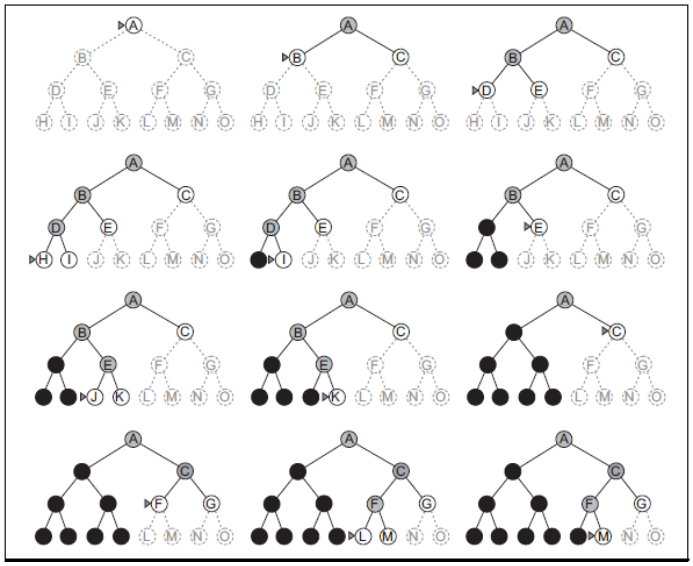
  Cada nodo generado debe permanecer en la memoria, porque o es parte de la frontera o es un antepasado de un nodo de la frontera. La **complejidad en espacio** es, por lo tanto, la misma que la **complejidad en tiempo** (más un nodo para la raíz)

  Son un problema más grande los requisitos de memoria para la búsqueda primero en anchura que el tiempo de ejecución. No obstante, los requisitos de tiempo siguen siendo un factor importante, una solución a profundidad 12 llevaría 35 años. En general, los problemas de búsqueda de complejidad exponencial no pueden resolverse por métodos sin información, salvo casos pequeños.

Primero en profundidad (LIFO)

La búsqueda primero en profundidad siempre expande el nodo más profundo en la frontera actual del árbol de búsqueda. La búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda, donde los nodos no tienen ningún sucesor. Cuando esos nodos se expanden, son quitados de la frontera, así entonces la búsqueda «retrocede» al siguiente nodo más superficial que todavía tenga sucesores inexplorados.

Esta estrategia puede implementarse con una cola último en entrar primero en salir (LIFO), también conocida como una pila.



La **ventaja** es que tiene bajo requisito de memoria ya que necesita almacenar sólo un camino desde la raíz a un nodo hoja, junto con los nodos hermanos restantes no expandidos para cada nodo del camino. Una vez que un nodo se ha

expandido, se puede quitar de la memoria tan pronto como todos sus descendientes han sido explorados. Para un espacio de estados con factor de ramificación b y máxima profundidad m, la búsqueda primero en profundidad requiere almacenar sólo bm+1 nodos.

El **inconveniente** de la búsqueda primero en profundidad es que puede hacer una elección equivocada y obtener un camino muy largo (o infinito) aun cuando una elección diferente llevaría a una solución cerca de la raíz del árbol de búsqueda. La búsqueda primero en profundidad:

- **No es óptima**: explorará el subárbol izquierdo entero incluso si el nodo C es un nodo objetivo. Si el nodo J fuera también un nodo objetivo, entonces la búsqueda primero en profundidad lo devolvería como una solución.
- **No es completo**: si el subárbol izquierdo fuera de profundidad ilimitada y no contuviera ninguna solución, la búsqueda primero en profundidad nunca terminaría.

En el caso peor, la búsqueda primero en profundidad generará todos los nodos O(bm) del árbol de búsqueda, donde m es la profundidad máxima de cualquier nodo. Nótese que m puede ser mucho más grande que d (la profundidad de la solución más superficial), y es infinito si el árbol es ilimitado.

Una **variante** de la búsqueda primero en profundidad, llamada búsqueda hacia atrás utiliza menos memoria, ya que sólo se genera un sucesor a la vez. Cada nodo parcialmente expandido recuerda qué sucesor se expande a continuación, de esta manera sólo se necesita O(m) memoria más que el O(bm) anterior. La idea de generar un sucesor modificando directamente la descripción actual del estado más que copiarlo, reduce los requerimientos de memoria a solamente una descripción del estado y O(m) acciones.

Comparación

Criterio	Primero en anchura	Costo uniforme	Primero en profundidad	Profundidad limitada	Profundidad iterativa	Bidireccional (si aplicable)
¿Completa?	Sí[a]	Sí[a,b]	No	No	Sí[a]	Sí[a,d]
Tiempo	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Espacio	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
¿Optimal?	Sí[c]	Sí	No	No	Sí[c]	Sí[c,d]

Estados Repetidos (Árbol vs Grafo)

Una de las complicaciones más importantes al proceso de búsqueda es la posibilidad de perder tiempo expandiendo estados que ya han sido visitados y expandidos. Para algunos problemas, esta posibilidad nunca aparece, el espacio de estados es un árbol y hay sólo un camino a cada estado. Pero para otros, la repetición de estados es inevitable. Esto incluye todos los problemas donde las acciones son reversibles, como son los problemas de búsqueda de rutas y los puzles que deslizan sus piezas. Los árboles de la búsqueda para estos problemas son infinitos, pero si podamos parte de los estados repetidos, podemos cortar el árbol de búsqueda en un tamaño finito, generando sólo la parte del árbol que atraviesa el grafo del espacio de estados. Considerando solamente el árbol de búsqueda hasta una profundidad fija, es fácil encontrar casos donde la eliminación de estados repetidos produce una reducción exponencial del coste de la búsqueda. En el caso extremo, un espacio de estados de tamaño d+1 se convierte en un árbol con 2d hojas.

Si el algoritmo no detecta los estados repetidos, éstos pueden provocar que un problema resoluble llegue a ser irresoluble. La detección por lo general significa la comparación del nodo a expandir con aquellos que han sido ya expandidos; si se encuentra un emparejamiento, entonces el algoritmo ha descubierto dos caminos al mismo estado y puede desechar uno de ellos.

Para la búsqueda primero en profundidad, los únicos nodos en memoria son aquellos del camino desde la raíz hasta el nodo actual. La comparación de estos nodos permite al algoritmo descubrir los caminos que forman ciclos y que pueden eliminarse inmediatamente. Asegura que espacios de estados finitos no hagan árboles de búsqueda infinitos debido a los ciclos, pero no evita la proliferación exponencial de caminos que no forman ciclos. El único modo de evitar éstos es guardar más nodos en la memoria. Hay una compensación fundamental entre el espacio y el tiempo.

Comentado [Ui8]: Revisar esto

Si un algoritmo recuerda cada estado que ha visitado, entonces puede verse como la exploración directamente del grafo de espacio de estados. Podemos modificar el algoritmo general para incluir una estructura de datos llamada lista cerrada, que almacene cada nodo expandido. (A veces se llama a la frontera de nodos no expandidos lista abierta.) Si el nodo actual se empareja con un nodo de la lista cerrada, se elimina en vez de expandirlo. Sobre problemas con muchos estados repetidos, la búsqueda en grafo es mucho más eficiente que la búsqueda en árboles. Los requerimientos en tiempo y espacio, en el caso peor, son proporcionales al tamaño del espacio de estados. Esto puede ser mucho más pequeño que O(b^d).

La optimización para la búsqueda en grafos es una cuestión difícil. Cuando se detecta un estado repetido, el algoritmo ha encontrado dos caminos al mismo estado. El algoritmo siempre desecha el camino recién descubierto. Si el camino recién descubierto es más corto que el original, podría omitir una solución óptima.

- Esto no puede pasar cuando utilizamos la *búsqueda de coste uniforme* o la *búsqueda primero en anchura* con costos constantes, por lo que estas dos estrategias óptimas de búsqueda en árboles son también estrategias óptimas de búsqueda en grafos.
- La *búsqueda con profundidad iterativa*, por otra parte, utiliza la expansión primero en profundidad y puede seguir un camino subóptimo a un nodo antes de encontrar el óptimo. La búsqueda en grafos de profundidad iterativa tiene que comprobar si un camino recién descubierto a un nodo es mejor que el original, y si es así, podría tener que revisar las profundidades y los costos del camino de los descendientes de ese nodo.
- Notemos que el uso de una lista cerrada significa que la búsqueda primero en profundidad y la búsqueda en profundidad iterativa tienen unos requerimientos lineales en espacio. Como el algoritmo de BÚSQUEDA-GRAFOS mantiene cada nodo en memoria, algunas búsquedas son irrealizables debido a limitaciones de memoria.

Estrategia de Búsqueda Informada

Una estrategia de búsqueda informada (la que utiliza el conocimiento específico del problema más allá de la definición del problema en sí mismo) puede encontrar soluciones de una manera más eficiente que una estrategia no informada. El conocimiento del dominio del problema puede ayudar a dirigir el proceso de búsqueda de manera que sean exploradas en primer lugar aquellas trayectorias más prometedoras.

Heurística

- Una de las definiciones de la RAE: “En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.”
- Para los métodos de búsqueda, es una técnica que aumenta la eficiencia del proceso, posiblemente sacrificando demandas de completitud.
- Es conocimiento (muchas veces intuitivo) que puede guiar el proceso de búsqueda.
- Las heurísticas no garantizan la solución óptima, pero generalmente mejoran la calidad del proceso.

El conocimiento heurístico generalmente se incorpora al proceso de búsqueda a través de una función heurística. La función heurística h(n) determina un grado de “bondad” para cada estado evaluado.

Comentado [VCNC9]: Tambien hay ejemplo en la ppt

h(n): coste estimado del camino más barato desde el nodo n a un nodo objetivo.

Las funciones heurísticas son la forma más común de transmitir el conocimiento adicional del problema al algoritmo de búsqueda. Las consideraremos funciones arbitrarias específicas del problema, con una restricción: si n es un nodo objetivo, entonces h(n) = 0.

Método: Primero el mejor

La búsqueda primero el mejor trata de expandir el nodo más cercano al objetivo, alegando que probablemente conduzca rápidamente a una solución. Así, evalúa los nodos utilizando solamente la función heurística: f(n)=h(n). La heurística h’ es la distancia estimada al estado objetivo.

Utiliza una lista (ABIERTA) para guardar los estados a recorrer. La lista ABIERTA se ordena según el valor de la heurística (primero el mejor). Se utiliza otra lista (CERRADA) para guardar los estados recorridos.

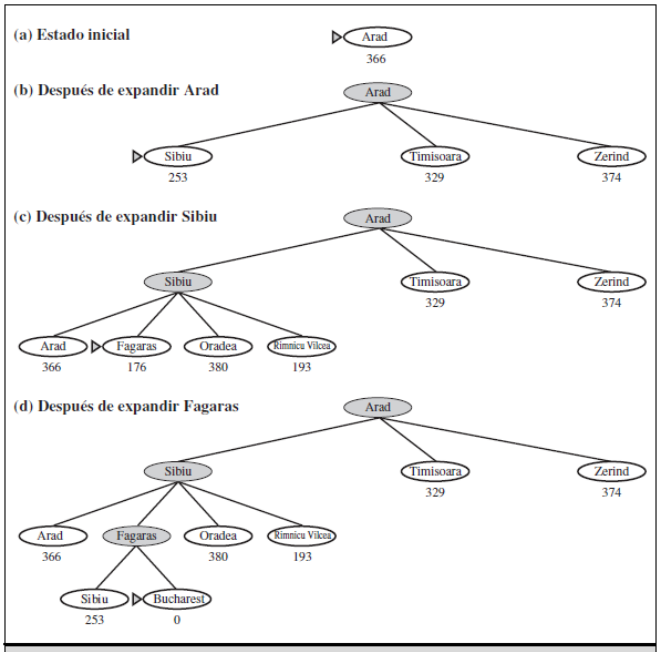


Figura 4.2 Etapas en una búsqueda primero el mejor avara para Bucarest utilizando la heurística distancia en línea recta h_{DLR}. Etiquetamos lo nodos con sus h-valores.

El ejemplo muestra el proceso para encontrar un camino desde Arad a Bucarest. El primer nodo a expandir desde Arad será Sibiu, porque está más cerca de Bucarest que Zerind o que Timisoara. El siguiente nodo a expandir será Fagaras, porque es la más cercana. Fagaras en su turno genera Bucarest, que es el objetivo. Para este problema particular, la búsqueda encuentra una solución sin expandir un nodo que no esté sobre el camino solución, de ahí, que su coste de búsqueda es mínimo. Sin embargo, no es óptimo: el camino vía Sibiu y Fagaras a Bucarest es 32 kilómetros más largo que el camino por Rimnicu Vilcea y Pitesti. Esto muestra por qué se llama algoritmo «avaro» (en cada paso trata de ponerse tan cerca del objetivo como pueda).

La minimización de h(n) es susceptible de ventajas falsas. Por ejemplo, para ir de Iasi a Fagaras la heurística sugiere que Neamt sea expandido primero, porque es la más cercana a Fagaras, pero esto es un callejón sin salida. La solución es ir primero a Vaslui (un paso que en realidad está más lejano del objetivo según la heurística) y luego seguir a Urziceni, Bucarest y Fagaras. En este caso, la heurística provoca nodos innecesarios para expandir. Además, si no somos cuidadosos en descubrir estados repetidos, la solución nunca se encontrará, la búsqueda oscilará entre Neamt e Iasi.

La búsqueda voraz primero el mejor se parece a la búsqueda primero en profundidad, prefiere seguir un camino hacia el objetivo, pero volverá atrás cuando llegue a un callejón sin salida. Sufre los mismos defectos que la búsqueda primero en profundidad:

- **No es óptima**
- **No es completa**, porque puede ir hacia abajo en un camino infinito y nunca volver para intentar otras posibilidades.
- La **complejidad en tiempo y espacio**, del caso peor, es O(b^m), donde m es la profundidad máxima del espacio de búsqueda. Con una buena función se puede reducir la complejidad considerablemente dependiendo del problema particular y de la calidad de la heurística.

Método: A*

A la forma más ampliamente conocida de la búsqueda primero el mejor se le llama búsqueda A*. Evalúa los nodos combinando g(n), el coste para alcanzar el nodo, y h(n), el coste de ir al nodo objetivo: **f (n) = g(n) + h(n).**

Comentado [VC10]: No se si esta bien, recurrir al libro para terminar de entender con ejemplos

Ya que la g(n) nos da el coste del camino desde el nodo inicio al nodo n, y la h(n) el coste estimado del camino más barato desde n al objetivo, tenemos: **f (n) = coste más barato estimado de la solución a través de n.**

- Utiliza una heurística f'=h'+g, donde g es la cantidad de estados recorridos hasta llegar al estado actual. g es una penalización a los caminos largos.
- La ventaja de A* es que encuentra caminos más cortos.
- Si la heurística es optimista, no se permiten estados repetidos y se tiene en cuenta un *detalle* garantiza la solución óptima.

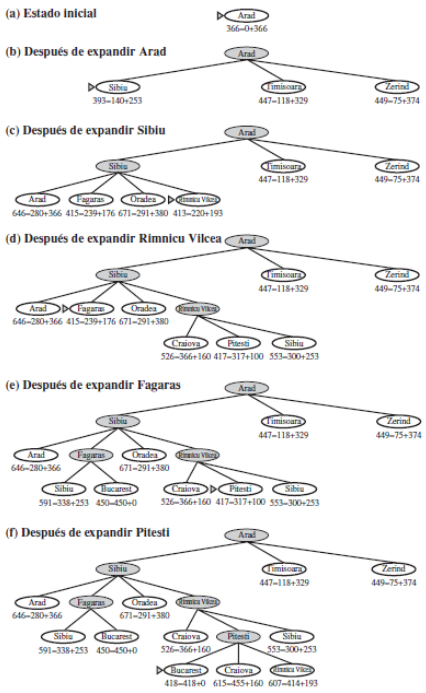
Si tratamos de encontrar la solución más barata, es razonable intentar primero el nodo con el valor más bajo de g(n) + h(n). Con esta estrategia, con tal de que la función heurística h(n) satisfaga ciertas condiciones, la búsqueda A* es tanto completa como óptima.

La optimalidad de A* es sencilla de analizar si se usa con la BÚSQUEDA-ÁRBOLES. En este caso, A* es óptima si h(n) es una **heurística admisible**, es decir, con tal de que la h(n) nunca sobrestime el coste de alcanzar el objetivo. Las heurísticas admisibles son por naturaleza optimistas, porque piensan que el coste de resolver el problema es menor que el que es en realidad. Ya que g(n) es el coste exacto para alcanzar n, tenemos como consecuencia inmediata que la f (n) nunca sobrestima el coste verdadero de una solución a través de n. Un ejemplo obvio de una heurística admisible es la distancia en línea recta hDLR que usamos para ir a Bucarest. La distancia en línea recta es admisible porque el camino más corto entre dos puntos cualquiera es una línea recta, entonces la línea recta no puede ser una sobrestimación.

De este ejemplo, podemos extraer una demostración general de que A*, utilizando la BÚSQUEDA-ÁRBOLES, es óptimo si la h(n) es admisible. Supongamos que aparece en la frontera un nodo objetivo subóptimo G2, y que el coste de la solución óptima es C*. Entonces, como G2 es subóptimo y h(G2) = 0 (cierto para cualquier nodo objetivo), sabemos que f (G2) = g(G2) + h(G2) = g(G2) > C*

Ahora considerando un nodo n de la frontera que esté sobre un camino solución óptimo. (Siempre debe de haber ese nodo si existe una solución.) Si la h(n) no sobrestima el coste de completar el camino solución, entonces sabemos que f (n) = g(n) + h(n) <= C*

Hemos demostrado que f (n) <= C* < f (G2), así que G2 no será expandido y A* debe devolver una solución óptima.



Unidad 3: Razonamiento en ambientes deterministas II

3.1 Agentes lógicos

Los **agentes basados en conocimiento** se pueden aprovechar del conocimiento expresado en formas genéricas, combinando y recombinando la información para adaptarse a diversos propósitos.

El **conocimiento** y el **razonamiento** juegan un papel importante cuando se trata con **entornos parcialmente observables**. Un agente basado en conocimiento puede combinar el **conocimiento** general con las **percepciones** reales para **inferir** aspectos ocultos del estado del mundo, antes de seleccionar cualquier acción.

El entendimiento del **lenguaje natural** también necesita inferir estados ocultos, en concreto, la **intención** del que habla.

Agentes basados en conocimiento

El componente principal es su **base de conocimiento** o BC. Esta es un **conjunto de sentencias**, cada una de las cuales se expresa en un lenguaje denominado lenguaje de representación del conocimiento y representa alguna **aserción acerca del mundo**.

Debe haber un **mecanismo** para **añadir** sentencias nuevas a la base de conocimiento, y uno para **preguntar** qué se sabe en la base de conocimiento. Los nombres estándar para estas dos tareas son DECIR y PREGUNTAR, respectivamente. Ambas tareas requieren realizar **inferencia**, es decir, derivar nuevas sentencias de las antiguas. En los agentes lógicos, la inferencia debe cumplir con el requisito esencial de que cuando se PREGUNTA a la base de conocimiento, la respuesta debe seguirse de lo que se HA DICHO a la base de conocimiento previamente.

Al igual que todos nuestros agentes, éste recibe una percepción como entrada y devuelve una acción. Inicialmente, la BC contiene algún **conocimiento de antecedentes**.

Cada vez que el programa del agente es invocado, realiza dos cosas. **Primero, DICE** a la base de conocimiento lo que ha **percibido**. **Segundo, PREGUNTA** a la base de conocimiento qué **acción** debe ejecutar. En este segundo proceso de

responder a la pregunta, se debe realizar un razonamiento extensivo acerca del estado actual del mundo, de los efectos de las posibles acciones, etcétera. Una vez se ha escogido la acción, el agente graba su elección mediante un DECIR y ejecuta la acción. Este segundo DECIR es necesario para permitirle a la base de conocimiento saber que la acción hipotética realmente se ha ejecutado.

```
función AGENTE-BC(percepción) devuelve una acción
  variables estáticas: BC, una base de conocimiento
                       t, un contador, inicializado a 0, que indica el tiempo

  DECIR(BC, CONSTRUIR-SENTENCIA-DE-PERCEPCIÓN(percepción, t))
  acción ← PREGUNTAR(BC, PEDIR-ACCIÓN(t))
  DECIR(BC, CONSTRUIR-SENTENCIA-DE-ACCIÓN(acción, t))
  t ← t + 1
  devolver acción
```

Los detalles del lenguaje de representación están ocultos en las dos funciones que implementan la interfaz entre los sensores, los accionadores, el núcleo de representación y el sistema de razonamiento. **CONSTRUIR-SENTENCIA-DE-PERCEPCIÓN** toma una percepción y un instante de tiempo y devuelve una sentencia afirmando lo que el agente ha percibido en ese instante de tiempo. **PEDIR-ACCIÓN** toma un instante de tiempo como entrada y devuelve una sentencia para preguntarle a la base de conocimiento qué acción se debe realizar en ese instante de tiempo. Los detalles de los mecanismos de inferencia están ocultos en **DECIR y PREGUNTAR**.

Gracias a las definiciones de DECIR y PREGUNTAR, el agente basado en conocimiento no obtiene las acciones mediante un proceso arbitrario. Es compatible con una descripción al nivel de conocimiento, en el que sólo necesitamos especificar lo que el agente sabe y los objetivos que tiene para establecer su **comportamiento**.

El **programa del agente**, inicialmente, antes de que empiece a recibir percepciones, se construye mediante la adición, una a una, de las **sentencias** que representan el **conocimiento del entorno** que tiene el diseñador. El diseño del lenguaje de representación que permita, de forma más fácil, expresar este conocimiento mediante sentencias simplifica el problema de la construcción del agente. Este enfoque en la construcción de sistemas se denomina enfoque **declarativo**. Por el contrario, el enfoque **procedural** codifica los comportamientos que se desean obtener directamente en código de programación. Para que un agente tenga éxito su diseño debe combinar elementos declarativos y procedurales.

A parte de DECIRLE al agente lo que necesita saber, podemos proveer a un agente basado en conocimiento de los mecanismos que le permitan **aprender** por sí mismo. De esta manera, el agente puede ser totalmente **autónomo**.

Lógica

Las bases de conocimiento se componen de **sentencias**. Estas sentencias se expresan de acuerdo a la **sintaxis** del lenguaje de representación, que especifica todas las sentencias que están **bien formadas**. Las sentencias de la base de conocimiento del agente son configuraciones físicas reales (de las partes) del agente. El razonamiento implica generar y manipular estas configuraciones.

Una lógica también debe definir la **semántica** del lenguaje, que trata el «**significado**» de las sentencias. En lógica, esta definición es bastante más precisa. La semántica del lenguaje define el **valor de verdad** de cada sentencia respecto a cada **mundo posible**. Por ejemplo, la sentencia «x + y = 4» es verdadera en un mundo en el que x sea 2 e y sea 2, pero falsa en uno en el que x sea 1 e y sea 1.

Cuando necesitemos ser más precisos, utilizaremos el término **modelo** en lugar del de «mundo posible». (También utilizaremos la frase «m es un modelo de α» para indicar que la sentencia α es verdadera en el modelo m.)

El razonamiento lógico requiere de la relación de **implicación** lógica entre las sentencias. Su notación matemática es

$$\alpha \models \beta$$

para significar que la sentencia α implica la sentencia β. Si y sólo si en cada modelo en el que es verdadera, también lo es.

El concepto de **implicación** se puede aplicar para derivar conclusiones, es decir, llevar a cabo la **inferencia lógica**. Para entender la implicación y la inferencia nos puede ayudar pensar en el conjunto de todas las consecuencias de la BC como en un pajar, y en α como en una aguja. La implicación es como la aguja que se encuentra en el pajar, y la inferencia consiste en encontrarla. Si el algoritmo de inferencia i puede derivar de la BC, entonces escribimos:

$$BC \vdash_i \alpha,$$

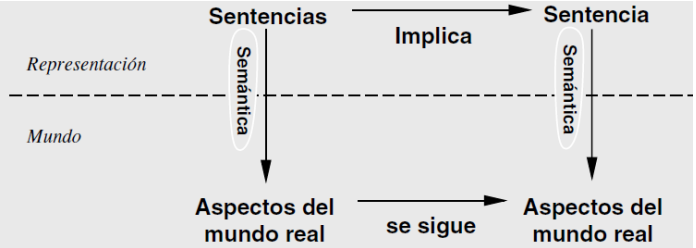
«α se deriva de la BC mediante i» o «i deriva α de la BC»

Un algoritmo de inferencia que deriva sólo sentencias implicadas es **sólido** o **mantiene la verdad**. También es muy deseable la propiedad de **completitud**: un algoritmo de inferencia es completo si puede derivar cualquier sentencia que está implicada.

Hemos descrito un proceso de razonamiento en el que se garantiza que las **conclusiones** sean **verdaderas** en cualquier mundo en el que las **premisas** lo sean; en concreto, sí una **BC** es verdadera en el mundo real, entonces cualquier **sentencia α** que se derive de la BC mediante un procedimiento de **inferencia sólido** también será verdadera en el mundo real.

La **denotación** es la **conexión**, si la hay, entre los procesos de **razonamiento lógico** y el **entorno real** en el que se encuentra el agente. En concreto, ¿cómo sabemos que la BC es verdadera en el mundo real? (Después de todo, la BC sólo es «sintaxis» dentro de la cabeza del agente.)

El **programa del agente** crea una **sentencia** adecuada a partir de los **sensores**. Entonces, siempre que esa sentencia esté en la base de conocimiento será verdadera en el mundo real. Así, el significado y el valor de verdad de las sentencias de las percepciones se definen mediante el proceso de los sensores y el de la construcción de las sentencias, activada por el proceso previo.



Las sentencias son **configuraciones físicas** del agente, y el **razonamiento** es el proceso de construcción de nuevas configuraciones físicas a partir de las antiguas. El razonamiento lógico debería asegurar que las nuevas configuraciones representen **aspectos del mundo** que realmente se siguen de los aspectos que las antiguas configuraciones representan.

Lógica proposicional

Sintaxis

La **sintaxis** de la lógica proposicional nos define las **sentencias** que se pueden construir. Las **sentencias atómicas** (es decir, los elementos sintácticos indivisibles) se componen de un único **símbolo proposicional**. Cada uno de estos símbolos representa una proposición que puede ser verdadera o falsa. Utilizaremos letras mayúsculas para estos símbolos: P, Q, R

Hay dos símbolos proposicionales con significado fijado: Verdadero, que es la proposición que siempre es verdadera; y Falso, que es la proposición que siempre es falsa.

Las **sentencias complejas** se construyen a partir de sentencias más simples mediante el uso de las **conectivas lógicas**, que son las siguientes cinco:

1) **¬ (no)**. Se denomina negación.
2) **∧ (y)**. Se denomina conjunción; sus componentes son los conjuntores.

3) **V (o)**. Es una disyunción de los disyuntores.
4) **⇒ (implica)**. Una sentencia como (X ∧ Y) ⇒¬W se denomina implicación (o condicional). Su premisa o antecedente es (X ∧ Y) y su conclusión o consecuente es ¬W. Las implicaciones también se conocen como reglas o afirmaciones si-entonces.
5) **⇔ (si y sólo sí)**. La sentencia X⇔¬W es una bicondicional.

Gramática formal:

$$
\begin{aligned}
\textit{Sentencia} &\rightarrow \textit{Sentencia Atómica} \mid \textit{Sentencia Compleja} \\
\textit{Sentencia Atómica} &\rightarrow \textbf{Verdadero} \mid \textbf{Falso} \mid \textit{Símbolo Proposicional} \\
\textit{Símbolo Proposicional} &\rightarrow \textbf{P} \mid \textbf{Q} \mid \textbf{R} \mid \ldots \\
\textit{Sentencia Compleja} &\rightarrow \neg\, \textit{Sentencia} \\
&\mid (\textit{Sentencia} \wedge \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \vee \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \Rightarrow \textit{Sentencia}) \\
&\mid (\textit{Sentencia} \Leftrightarrow \textit{Sentencia})
\end{aligned}
$$

El orden de precedencia en la lógica proposicional (de mayor a menor) es: **¬, Λ, V, ⇒** y **⇔**.

La precedencia entre las conectivas no resuelve la ambigüedad en algunas sentencias.

Semántica

Una vez especificada la sintaxis de la lógica proposicional, vamos a definir su semántica. La semántica define las **reglas** para determinar el **valor de verdad** de una sentencia respecto a un **modelo** en concreto. En la lógica proposicional un modelo define el valor de verdad (verdadero o falso).

La semántica en **lógica proposicional** debe especificar cómo obtener el valor de verdad de cualquier sentencia, dado un modelo. Este proceso se realiza de forma **recursiva**. Todas las sentencias se construyen a partir de las sentencias atómicas y las cinco conectivas lógicas; entonces necesitamos establecer cómo definir el valor de verdad de las sentencias atómicas y cómo calcular el valor de verdad de las sentencias construidas con las cinco conectivas lógicas. Para las **sentencias atómicas** es sencillo:

- Verdadero es verdadero en todos los modelos y Falso es falso en todos los modelos.
- El valor de verdad de cada símbolo proposicional se debe especificar directamente para cada modelo.

Para las **sentencias complejas**, tenemos reglas como la siguiente

- Para toda sentencia s y todo modelo m, la sentencia ¬s es verdadera en m si y sólo sí s es falsa en m.

Este tipo de reglas reducen el cálculo del valor de verdad de una sentencia compleja al valor de verdad de las sentencias más simples. Las reglas para las conectivas se pueden resumir en una **tabla de verdad** que especifica el valor de verdad de cada sentencia compleja según la posible asignación de valores de verdad realizada a sus componentes.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
falso	*falso*	*verdadero*	*falso*	*falso*	*verdadero*	*verdadero*
falso	*verdadero*	*verdadero*	*falso*	*verdadero*	*verdadero*	*falso*
verdadero	*falso*	*falso*	*falso*	*verdadero*	*falso*	*falso*
verdadero	*verdadero*	*falso*	*verdadero*	*verdadero*	*verdadero*	*verdadero*

Ya hemos comentado que una base de conocimiento está compuesta por sentencias. Ahora podemos observar que esa base de conocimiento lógica es una conjunción de dichas sentencias. Es decir, si comenzamos con una BC vacía y ejecutamos DECIR(BC, S1)... DECIR(BC, Sn) entonces tenemos BC = S1 ∧ … ∧ Sn. Esto significa que podemos manejar bases de conocimiento y sentencias de manera intercambiable.

Hay una conectiva diferente denominada «o exclusiva» («xor» para abreviar) que es falsa cuando los dos disyuntores son verdaderos. No hay consenso respecto al símbolo que representa la o exclusiva, siendo las dos alternativas:

$\dot{\vee}$ y $\oplus$

La tabla de verdad de la bicondicional P⇔Q muestra que la sentencia es verdadera siempre que P ⇒ Q y Q ⇒ P lo son.

Inferencia

Nuestro primer algoritmo para la inferencia será una implementación directa del concepto de implicación: enumerar los modelos, y averiguar si α es verdadera en cada modelo en el que la BC es verdadera.

algoritmo general para averiguar la implicación en lógica proposicional:

```
función ¿IMPLICACIÓN-EN-TV?(BC, α) devuelve verdadero o falso
  entradas: BC, la base de conocimiento, una sentencia en lógica proposicional
            α, la sentencia implicada, una sentencia en lógica proposicional
  símbolos ← una lista de símbolos proposicionales de la BC y α
  devuelve COMPROBAR-TV(BC, α, símbolos, [ ])

función COMPROBAR-TV(BC, α, símbolos, modelo) devuelve verdadero o falso
  si ¿VACÍA?(símbolos) entonces
    si ¿VERDADERO-LP?(BC, modelo) entonces devuelve ¿VERDADERO-LP?(α, modelo)
    sino devuelve verdadero
  sino hacer
    P ← PRIMERO(símbolos); resto ← RESTO(símbolos)
    devuelve CHEQUEAR-TV(BC, α, resto, EXTENDER(P, verdadero, modelo)) y
             COMPROBAR-TV(BC, α, resto, EXTENDER(P, falso, modelo))
```

¿IMPLICACIÓN-EN-TV? Realiza una enumeración recursiva de un espacio finito de asignaciones a variables. El algoritmo es **sólido** porque implementa de forma directa la definición de implicación, y es **completo** porque trabaja para cualquier BC y sentencia α, y siempre finaliza. La **complejidad temporal** del algoritmo es O(2n). (La **complejidad espacial** sólo es O(n) porque la enumeración es en primero en profundidad.)

Equivalencia, validez y satisfacibilidad

Al igual que la implicación, estos conceptos se aplican a todos los tipos de lógica.

El primer concepto es la **equivalencia lógica**: dos sentencias α y β son equivalentes lógicamente si tienen los mismos valores de verdad en el mismo conjunto de modelos. Este concepto lo representamos con α **⇔** β.

Equivalencias lógicas:

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{Conmutatividad de } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{Conmutatividad de } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{Asociatividad de } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{Asociatividad de } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{Eliminación de la doble negación} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{Contraposición} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{Eliminación de la implicación} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{Eliminación de la bicondicional} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{Ley de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{Ley de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{Distribución de } \wedge \text{ respecto a } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{Distribución de } \vee \text{ respecto a } \wedge
\end{aligned}
$$

El segundo concepto es el de **validez**. Una sentencia es válida si es verdadera en todos los modelos. Por ejemplo, la sentencia P ∨ ¬P es una sentencia válida. Las sentencias válidas también se conocen como **tautologías**, son necesariamente verdaderas y por lo tanto vacías de significado. ¿Qué utilidad tienen las sentencias válidas? De nuestra definición de implicación podemos derivar el **teorema de la deducción**:

Para cualquier sentencia α y β, α ⊨ β sí y sólo si la sentencia (α ⇒ β) es válida.

El último concepto es el de **satisfacibilidad**. Una sentencia es satisfactoria sí es verdadera para algún modelo. Si una sentencia α es verdadera en un modelo m, entonces decimos que m satisface α, o que m es un modelo de α.

Con algunas transformaciones adecuadas, los problemas de búsqueda también se pueden resolver mediante satisfacibilidad. La validez y la satisfacible están íntimamente relacionadas: α es válida si y sólo si ¬ α es insatisfacible; en contraposición, α es satisfacible si y sólo si ¬ α no es válida.

$$\alpha \vDash \beta \text{ sí y sólo si la sentencia } (\alpha \wedge \neg\beta) \text{ es insatisfactoria.}$$

Patrones de razonamiento en lógica proposicional

Los **patrones estándar de inferencia** se pueden aplicar para derivar cadenas de conclusiones que nos llevan al objetivo deseado. Estos patrones se denominan **reglas de inferencia**. La regla más conocida es la llamada **Modus Ponens** que se escribe como sigue:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

La notación nos dice que, cada vez que encontramos dos sentencias en la forma α ⇒ β y α, entonces la sentencia β puede ser inferida.

Otra regla de inferencia útil es la **Eliminación-Λ**, que expresa que, de una conjunción se puede inferir cualquiera de sus conjuntores:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Estas reglas se pueden utilizar sobre cualquier instancia en la que es aplicable, generando **inferencias sólidas**, sin la necesidad de enumerar todos los modelos. Todas las **equivalencias lógicas** vistas anteriormente se pueden utilizar como **reglas de inferencia**.

Pero no todas las reglas de inferencia se pueden usar en **ambas direcciones**. Por ejemplo, no podemos utilizar el Modus Ponens en la dirección opuesta para obtener α ⇒ β y α a partir de β.

A la **derivación** (una secuencia de aplicaciones de reglas de inferencia) se le denomina una **prueba** (o **demostración**). Obtener una prueba es muy semejante a encontrar una **solución** en un problema de búsqueda.

La **búsqueda de pruebas** es una alternativa a tener que enumerar los modelos. La búsqueda se puede realizar hacia delante a partir de la base de conocimiento inicial, aplicando las reglas de inferencia para derivar la sentencia objetivo, o hacia atrás, desde la sentencia objetivo, intentando encontrar una cadena de reglas de inferencia que nos lleven a la base de conocimiento inicial.

El hecho de que la **inferencia** en lógica proposicional sea un **problema NP-completo** nos hace pensar que, en el peor de los casos, la **búsqueda de pruebas** va a ser no mucho más eficiente que la **enumeración de modelos**. Sin embargo, en muchos casos prácticos, encontrar una prueba puede ser altamente eficiente simplemente porque el proceso puede **ignorar las proposiciones irrelevantes**, sin importar cuántas de éstas haya.

Esta propiedad de los sistemas lógicos en realidad proviene de una característica mucho más fundamental, denominada **monótono**. La característica de **monotonismo** nos dice que el conjunto de sentencias implicadas sólo puede aumentar (pero no cambiar) al añadirse información a la base de conocimiento.

El **monotonismo** permite que las reglas de inferencia se puedan aplicar siempre que se hallen premisas aplicables en la base de conocimiento; la conclusión de la regla debe permanecer sin hacer caso de qué más hay en la base de conocimiento.

Resolución

Hemos argumentado que las **reglas de inferencia** vistas hasta aquí son **sólidas**, pero no hemos visto la cuestión acerca de lo **completo** de los algoritmos de inferencia que las utilizan. Los algoritmos de búsqueda como el de búsqueda en profundidad iterativa son completos en el sentido de que éstos encontrarán cualquier **objetivo alcanzable**, pero si las reglas de inferencia no son adecuadas, entonces el objetivo no es alcanzable; no existe una prueba que utilice sólo esas reglas de inferencia.

Se introduce una **regla de inferencia** sencilla, la **resolución**, que nos lleva a un **algoritmo de inferencia completo** cuando se empareja a un **algoritmo de búsqueda completo**.

Obtiene **demostraciones por refutación**. Para probar la veracidad de una cláusula se intenta demostrar que su negación lleva a una contradicción con las proposiciones conocidas.

Es un proceso iterativo simple. En cada paso, se comparan (resuelven) dos cláusulas llamadas "cláusulas padre", produciendo una nueva cláusula. La nueva cláusula representa la forma en la que las cláusulas padre interaccionan entre ellas.

Forma normal conjuntiva

La regla de resolución sólo se puede aplicar a disyunciones de literales, por lo tanto, sería muy importante que la base de conocimiento y las preguntas a ésta estén formadas por disyunciones.

Toda **sentencia** en lógica proposicional es equivalente lógicamente a una **conjunción de disyunciones de literales**. Una sentencia representada mediante una conjunción de disyunciones de literales se dice que está en forma normal conjuntiva o FNC.

Procedimiento de conversión:

1) Eliminar ⇔, sustituyendo α⇔β por (α⇒β) ∧ (β⇒α)
2) Eliminar ⇒, sustituyendo α⇒β por ¬α ∨ β
3) Una FNC requiere que la ¬ se aplique sólo a los literales, por lo tanto, debemos «anidar las ¬» mediante la aplicación reiterada de las siguientes equivalencias
    a. ¬(¬α) ≡ α (eliminación de la doble negación)
    b. ¬(α ∧ β) ≡ (¬α ∨ ¬β) (de Morgan)
    c. ¬(α ∨ β) ≡ (¬α ∧ ¬β) (de Morgan)
4) Si nos queda de esta forma:

    $$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge ((\neg H_{1,2} \wedge \neg H_{2,1}) \vee B_{1,1})$$

    Donde tenemos una sentencia que tiene una ∧ con operadores de ∨ anidados, aplicados a literales y a una ∧ anidada.

    Aplicamos la ley de distributividad, distribuyendo la ∨ sobre la ∧ cuando nos es posible:

    $$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge ((\neg H_{1,2} \wedge \neg H_{2,1}) \vee B_{1,1})$$

    Quedando del siguiente modo:

    $$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge (\neg H_{1,2} \vee B_{1,1}) \wedge (\neg H_{2,1} \vee B_{1,1})$$

La sentencia inicial ahora está en FNC, una conjunción con tres cláusulas. Es más difícil de leer, pero se puede utilizar como entrada en el procedimiento de resolución.

Un algoritmo de resolución

Los procedimientos de inferencia basados en la resolución trabajan utilizando el principio de **prueba mediante contradicción**. Es decir, para demostrar que BC ⊨ α, demostramos que (BC ∧ ¬α) es insatisfacible. Lo hacemos demostrando una contradicción.

Primero se convierte (BC ∧ ¬α) a FNC. Entonces, se aplica la regla de resolución a las cláusulas obtenidas. Cada par que contiene **literales complementarios** se resuelve para generar una nueva cláusula, que se añade al conjunto de cláusulas si no estaba ya presente. El proceso continúa hasta que sucede una de estas dos cosas:

- No hay nuevas cláusulas que se puedan añadir, en cuyo caso α no implica β
- Se deriva la cláusula vacía de una aplicación de la regla de resolución, en cuyo caso α implica β.

La **cláusula vacía** (una disyunción sin disyuntores) es equivalente a Falso porque una disyunción es verdadera sólo si al menos uno de sus disyuntores es verdadero. Otra forma de ver que la cláusula vacía representa una contradicción es observar que se presenta sólo si se resuelven dos cláusulas unitarias complementarias, tales como P y ¬P.

función RESOLUCIÓN-LP(BC, α) **devuelve** *verdadero* o *falso*
  entradas: BC, la base de conocimiento, una sentencia en lógica proposicional
    α, la petición, una sentencia en lógica proposicional

  cláusulas ← el conjunto de cláusulas de $BC \wedge \neg\alpha$ en representación FNC
  nueva ← { }
  bucle hacer
    para cada C_i, C_j **en** cláusulas **hacer**
      resolventes ← RESUELVE-LP(C_i, C_j)
      si *resolventes* contiene la cláusula vacía **entonces devolver** *verdadero*
      nueva ← *nueva* ∪ *resolventes*
    si *nueva* ⊆ *cláusulas* **entonces devolver** *falso*
    cláusulas ← *cláusulas* ∪ *nueva*

La función RESUELVE-LP devuelve el conjunto de todas las cláusulas posibles que se obtienen de resolver las dos entradas.

Completitud de la resolución

Ahora vamos a demostrar por qué es completo el procedimiento RESOLUCIÓN-LP. Para hacerlo nos vendrá bien introducir el concepto de cierre de la resolución CR(S) del conjunto de cláusulas S, que es el conjunto de todas las cláusulas derivables, obtenidas mediante la aplicación repetida de la regla de resolución a las cláusulas de S o a las derivadas de éstas. El cierre de la resolución es lo que calcula el procedimiento RESOLUCIÓN-LP y asigna como valor final a la variable cláusulas. Es fácil ver que CR(S) debe ser finito, porque sólo hay un conjunto finito de las diferentes cláusulas que se pueden generar a partir del conjunto de símbolos P1…, Pk que aparecen en S. Por eso, el procedimiento RESOLUCIÓN-LP siempre termina.

El teorema de la completitud para la resolución en lógica proposicional se denomina **teorema fundamental de la resolución**:

Si un conjunto de cláusulas es insatisfacible, entonces el cierre de la resolución de esas cláusulas contiene la cláusula vacía.

Encadenamiento hacia delante y hacia atrás

La **completitud** de la **resolución** hace que ésta sea un método de inferencia muy importante. Sin embargo, en muchos casos prácticos no se necesita todo el poder de la resolución. Las **bases de conocimiento** en el mundo real a menudo contienen sólo cláusulas, de un tipo restringido, denominadas **cláusulas de Horn**. Una cláusula de Horn es una **disyunción de literales** de los cuales, como mucho uno es positivo.

La restricción de que haya sólo un literal positivo puede parecer algo arbitraria y sin interés, pero realmente es muy importante, debido a tres razones:

1) Cada **cláusula de Horn** se puede escribir como una **implicación** cuya premisa sea una conjunción de literales positivos y cuya conclusión sea un **único literal positivo**.

   Las cláusulas de Horn como ésta, con exactamente un literal positivo, se denominan **cláusulas positivas**. El literal positivo se denomina **cabeza**, y la disyunción de literales negativos **cuerpo** de la cláusula.

Una cláusula de Horn sin literales positivos se puede escribir como una implicación cuya conclusión es el literal Falso.

Una cláusula positiva que no tiene literales negativos simplemente afirma una proposición dada que se denomina hecho. Las cláusulas positivas forman la base de la programación lógica.

2) La inferencia con cláusulas de Horn se puede realizar mediante los algoritmos de **encadenamiento hacia delante** y de **encadenamiento hacia atrás**.
3) Averiguar si hay o no implicación con las cláusulas de Horn se puede realizar en un tiempo que es lineal respecto al tamaño de la base de conocimiento

El **encadenamiento hacia adelante** es un ejemplo del concepto general de razonamiento dirigido por los datos, es decir, un razonamiento en el que el foco de atención parte de los datos conocidos. Este razonamiento se puede utilizar en un agente para derivar conclusiones a partir de percepciones recibidas, a menudo, sin una petición concreta.

El **encadenamiento hacia atrás** trabaja a partir de una petición. Si se sabe que la petición q es verdadera, entonces no requiere ningún trabajo. De lo contrario el algoritmo encuentra aquellas implicaciones de la base de conocimiento de las que se concluye q. Si se puede probar que todas las premisas de una de esas implicaciones son verdaderas entonces q es verdadero.

Unidad 4: Razonamiento bajo incertidumbre

Lógica difusa

Cuando hablamos de incertidumbre, generalmente nos referimos a situaciones donde el agente no tiene acceso a toda la verdad sobre su ambiente. Entonces, se puede hablar de que los eventos son verdaderos o falsos con cierta probabilidad. En el caso de lógica difusa (fuzzy logic) particularmente esto no es así, un evento puede ser cierto en distinto grado.

La lógica difusa fue creada por Lotfi Zadeh en 1964. Cuando hablamos de lógica difusa, el adjetivo “difuso” se debe a que, en esta lógica, los valores de verdad son no-deterministas y tienen, por lo general, una connotación de incertidumbre.

La lógica difusa tiene como base los denominados conjuntos difusos y posee un sistema de inferencia basado en reglas de producción de la forma “si **antecedente** entonces **consecuente**”, donde los valores lingüísticos del antecedente y el consecuente están definidos por conjuntos difusos.

La LD se adapta al razonamiento humano, el cual busca un balance entre la precisión y el significado. Es un tipo de lógica multivaluada. Aunque los valores utilizados varían entre 0 y 1, en el caso más general no se pueden tomar como probabilidades. Se utilizan en campos como control numérico/automático, soporte de decisión, sistemas expertos y visión computarizada.

Ejemplo:

El problema consiste en determinar el valor “correcto”, para una propina que se debe abonar en función de algunas características del servicio recibido.

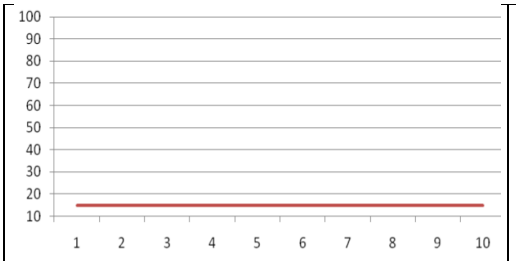
La esencia del problema es que:

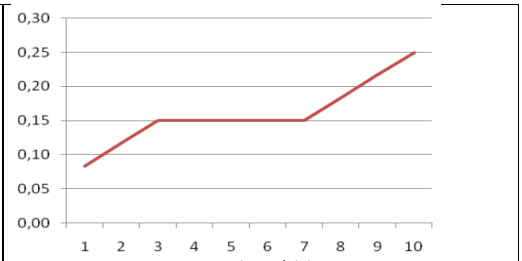
- Si el servicio es pobre luego la propina es poca
- Si el servicio es bueno luego la propina es media
- Si el servicio es excelente luego la propina es generosa

Si quisiéramos incluir la incidencia de la calidad de la comida solo agregamos algunas reglas:

- Si la comida es rancia luego la propina es poca
- Si la comida es deliciosa luego la propina es generosa

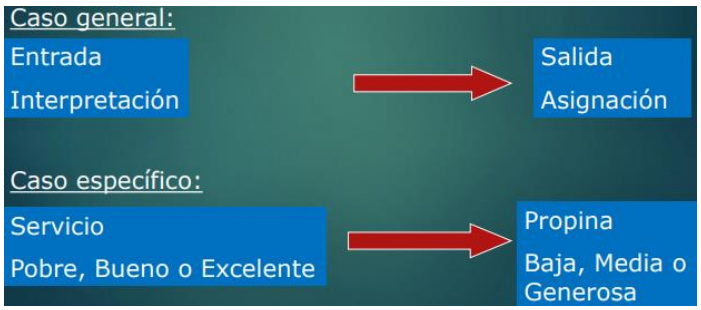
Caso de lógica tradicional	Ensayo de una función diferente





Si servicio es <= 3, propina = (0.10/3) * servicio + 0.05;
else si servicio <= 7, propina = 0.15;
else si servicio <= 10, propina = (0.10/3) * (servicio – 7) + 0.15;

Un sistema de inferencia difusa interpreta los valores de un vector de entrada y basado en un conjunto de reglas, asigna valores al vector de salida. Estas reglas se evalúan en paralelo y se refieren a variables y adjetivos que afectan a las variables. Antes de construir las reglas se definen los rangos de las variables y la forma en la que las afectan los adjetivos.



Se definen conjuntos (difusos) para cada variable. Los conjuntos se definen mediante funciones de membresía. Cada función indica el grado de pertenencia de la variable a dicho conjunto. Cada elemento puede pertenecer, en cierto grado, a más de un conjunto definido sobre la misma variable. Las funciones más comunes son:

- Triangular
- Trapezoidal
- Gaussiana
- Singleton

Reglas

La primera parte de la regla es el antecedente o premisa y la segunda es el consecuente o conclusión.

SI antecedente ENTONCES consecuente.

El antecedente es una interpretación que retorna un valor entre 0 y 1, mientras que el consecuente asigna un conjunto difuso a la variable de salida. Ejemplo: “si el servicio es bueno, la propina es media”

En lógica binaria, si el antecedente es verdadero, el consecuente es verdadero, pero en lógica difusa, si el antecedente es verdadero en cierto grado, el consecuente es verdadero en el mismo grado.

0,3 A → 0,3 B

Los antecedentes pueden estar compuestos por múltiples partes, por ejemplo: “si el cielo está gris y el viento es fuerte y la presión es baja, entonces…”. En este caso, las partes del antecedente se calculan simultáneamente y se resuelven como un escalar.

El consecuente especifica un conjunto difuso para la salida. La función de implicación modifica el conjunto en el grado de cumplimiento del antecedente.

Pasos de aplicación de las reglas

1. **Fuzzificación**

Se determina el grado en el que cada entrada pertenece a cada uno de los conjuntos difusos. Se evalúan las funciones de membresía.

2. **Aplicación de operadores difusos**

Si alguna regla tiene un antecedente compuesto se deben aplicar los operadores para obtener un único número que representa el antecedente. Cada operador difuso recibe uno o más valores y devuelve un único valor de verdad.

Operador	Método	Expresión
P y Q	Mínimo	mín(P, Q)
"	Producto	P * Q
"	Truncamiento	máx((P + Q - 1), 0)
P o Q	Máximo	máx(P, Q)
"	Amplificación	P + Q * (1 - P)
"	Adición	mín(P + Q, 1)
no P	Complemento	1 – P

3. **Aplicación del método de implicación**

Conforma el consecuente (conjunto difuso) para cada regla. El resultado es proporcional al valor del antecedente.

4. **Agregación**

Se unifican las salidas uniendo los procesos paralelos. Se combinan las salidas en un único conjunto difuso. La entrada del proceso es la lista de las salidas truncadas según el resultado de la implicación. La salida es un único conjunto difuso para cada variable de salida. Algunos métodos son máx, prob OR y sum.

5. **Defuzzificación**

Devuelve la salida final del modelo, un único número por variable. La entrada es el conjunto logrado en el paso de agregación. El método más popular es el cálculo del centroide:

$$CG = \frac{\sum_{x=a}^{b} \mu_A(x)x}{\sum_{x=a}^{b} \mu_A(x)}$$