uc3m | Universidad Carlos III de Madrid

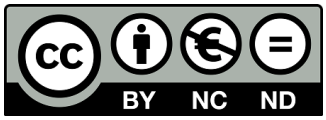
Grado en Ingeniería Informática

2021-2022

Apuntes

Ingeniería del Conocimiento

Jorge Rodríguez Fraile[1]



[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INFORMACIÓN

1.1. Profesores

Teoría y coordinadora: Susana Fernández Arregui, sfarregu@inf.uc3m.es, enviar mail con tiempo para pedir tutorías.

Prácticas: Alba Gragera Álvarez, agragera@pa.uc3m.es

1.2. Presentación

Asistencia no obligatoria.

Taller: tareas/ejercicios autoevaluados (nosotros hacemos el ejercicio y después otro compañero lo evalúa) semanales, para llevar la asignatura al día. Seguir los plazos, para que esta dinámica se pueda seguir. Wooclap que se realizaran en clase, para seguir el temario.

1.2.1. Objetivos

- Aprender a diseñar e implementar Sistemas basados en el Conocimiento.
- Aprender a identificar problemas, no se nos dará el problema que debemos resolver sino que nos los encontraremos y solucionaremos (antigua mentalidad). Aprender para crear empleo no para trabajar en un empleo.
- Analizaremos como obtener el conocimiento de los expertos, lograr conceptualizarlo y formalizarlo para hacerlo accesible.
- Aprender a diseñar e implementar ontologías.
- Analizar las técnicas más utilizadas para implementar SBC.
- Estudiaremos CLIP, una versión reducida.
- Aplicar los conocimientos teóricos a problemas complejos.

Lograremos adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación. También obtendremos soft-skill que nos permitirán destacar entre el resto de los graduados, como son la comunicación escrita y oral, flexibilidad, responsabilidad, trabajo en equipo.

1.2.2. Evaluación

- Habrá un examen de CLIP, día de asistencia obligatoria, un martes.
- 30 % Examen final.
  - Consistirán en formalizar y resolver un problema dado. También habrá preguntas relacionadas con las prácticas obligatorias.
  - Nota mínima de 4.
- 70 % Teoría.
  - Actividad individual relacionada con magistrales 15 %, son la que se realizan peer-to-peer.
    - Preguntas cortas de manera semanal.
    - Se reciben puntos por realizarlo 70 % y un 30 % por evaluar a un compañero (anónimamente).
    - Las notas se medirán en estrellas: 0, 2'5, 5, 7'5 o 10.
  - Practica obligatoria en parejas 30 %, en las clases prácticas.
  - Actividad parcial 25 %.
    - Examen CLIPS y antologías 1 punto.
    - Actividades foro 1 punto, 0.25 entregas y 0.25 por cada foro.
    - Entregas dominios de planificación 0.5 puntos.
    - Practica 1: Implementación en CLIPS 1.5 puntos.
    - Practica 2: Implementación en PDDL 1.5 puntos.

Prácticas dirigidas a conceptualizar (recoger todo el conocimiento, independientemente mediante el sistema que lo vayamos a realizar) e implementar el conocimiento, realizado mediante 2 prácticas.

2. CLIPS

2.1. Introducción

CLIPS: C Languaje Integrated Production System. Es la unión del lenguaje C y LIPS, de ahí su nombre.

Desarrollado por Software Technology Branch (STB) NASA desde 1986.

Combina los paradigmas de lenguajes declarativos, funcionales y orientados a objetos.

No tiene interfaz se utiliza a través del terminal, aparece *CLIPS >*.

Distingue entre mayúsculas y minúsculas, todos los comandos se presentan entre paréntesis (todo va con paréntesis).

2.1.1. Arquitectura de un sistema de producción

- **Base de Hechos o Memoria de Trabajo:** Representa conocimientos del dominio sobre el problema (hechos establecidos y metas a alcanzar). Según se ejecutan las reglas se va modificando, parte de esta es permanente (la inicial, no la del propio proceso).

  Representa el estado inicial, el conocimiento de manera declarativa (hechos e instancias).

  Los hechos pueden ser:

  - **Ordered:** Con uno o varios campos, donde el primer campo marca la relación entre los restantes, por lo que el orden es importante. Sin nombre.
  - **Non-ordered:** Abstraer la estructura de un hecho asignando un nombre, el orden de los slots no afecta dado que van con nombre.

- **Base de Reglas:** Representa conocimiento sobre cómo conseguir la solución del problema en forma de reglas de producción. Las reglas dan información sobre cómo resolverlo.

- **Motor de Inferencia (MI):** Razona sobre el conocimiento del problema y sobre su solución. Determina como se aplican las reglas.

Fases

- **Fase de selección**
  - **Restricción:** Acota el contenido de la Base de Reglas y la Base de Hechos según las características del programa.
  - **Equiparación:** Selección del conjunto de reglas candidatas para disparse.
    Equiparación de constantes, son aquellas secuencias de caracteres no precedidas por ?. Deben ser exactamente iguales.
    Equiparación de variables, este proceso equipara las variables de la regla con los hechos en busca de patrones de hechos que ocupen las posicione especificadas. Es decir, ocupe la misma posición en ordered o el mismo nombre en non-ordered.
    El mismo hecho puede ser utilizado varias veces en una regla, a menos que se especifique con un test neq de ambas.
  - **Resolución del Conjunto Conflicto:** Selección de la instancia de regla que va a ejecutar. Conjunto conflicto: Conjunto de instancias de las posibles reglas ejecutables.
    Estrategias de selección
    - **Orden de escritura:** Primera regla que cuyo antecedente es cierto según el orden de escritura en al BR.
    - **Prioridad** (la regla más prioritaria)
    - **Especificidad** (la regla más específica)
    - **Novedad** (la regla con elementos de la BH más recientemente añadidos o modificado)
    - Utilizar el **principio de refracción:** No se pueden ejecutar instancias de reglas ya disparadas
    - **Arbitrariedad**

    Estrategias de inferencia
    - **Encadenamiento hacia delante** (dirigido por el antecedente), parte de unos hechos particulares (el contenido de la BH).
      Busca reglas cuyo antecedente esté satisfecho por esos hechos.
    - **Encadenamiento hacia atrás** (dirigido por el consecuente), parte de un conjunto de metas u objetivos.
      Se buscan reglas cuya instanciación implica la consecución de una meta. La equiparación se realiza sobre el consecuente
      Si el antecedente equipara con la BH la meta se puede concluir

- **Fase de acción o ejecución**: La ejecución de una instancia de una regla modifica la Base de Hechos actual dando lugar a una Base de Hechos nueva al añadir, borrar o modificar. La nueva BH es el punto de partida para el siguiente ciclo.

  Por el principio de refracción nunca se ejecuta 2 veces la misma regla con los mismos hechos, entraría en bucle.

2.1.2. Expresiones

Se emplea una notación prefija, es decir primero la operación y después los operandos. Algunas funciones y operaciones matemáticas son $+ - / * mod\ div\ sqrt\ round\ integer$.

2.1.3. Tipos de datos

- NUMBER, INTEGER o FLOAT
- STRING
- SYMBOL, casos en los que tiene unos valores determinados
- EXTERNAL-ADDRESS
- FACT-ADDRESS
- INSTANCE-NAME
- INSTANCE-ADDRESS

2.2. Hechos

Son los elementos básicos del conocimiento y a cada uno CLIPS le asigna un identificador f-XX (fact-index).

El primer campo suele representar una relación entre las restantes, como (edad 14) o (padre-de Jose Alex).

Los campos sin valor se indican con *nil*.

2.2.1. Comandos sobre hechos

(facts) Base de hechos.

(assert hecho) Añadir hechos a la base de hechos. Si se trata de introducir dos hechos iguales solo deja el último.

(retract índice-hecho) Eliminar hechos de la base de hechos.

(modify dirección-hecho (nombre-atributo nuevo-valor)) Modifica el valor de un atributo. La dirección la podemos obtener buscando el hecho por algún atributo ?f <- (car (doors 2)).

(reset) Eliminar todos los hechos de la memoria de trabajo, añade initial-fact e initial-object. Cuando comenzamos lo utilizamos para añadir los hechos y que se tengan en cuenta.

(clear) Elimina todos los hechos y constructores.

(deffacts nombre comentarioOpcional hechos) Definir los hechos iniciales, se añaden al hacer (reset).

2.2.2. Plantillas o templates

(deftemplate nombre [comentario] slots-o-multislots)

Permite asignar un nombre a cada campo.

Puede almacenar un único valor con slot o varios con multislot, para que un mismo nombre tenga varios valores.

(slot/multislot nombre (type tipo) (allowed-values valoresPermitidos)/(range desde hasta) (default valor))

Para obligar al usuario a que de valor a un slot al crear un hecho en default se pone el valor *?NONE* y si queremos que CLIPS lo determine se pone *?DERIVE*, aunque este último no se usa mucho.

2.3. Reglas

(defrule nombre [comentario] [(declare (salience valorPrioridad))] premisa-o-elemento-de-condición =>acción)

La premisa puede ser:

- Un patrón que se encarga de consultar la existencia de un hecho en la Memoria de trabajo.
- Un test para comprobar el cumplimiento de una condición.
- Una negación para negar un patrón o test.

La acción puede ser:

- Crear un hecho o instancia.
- Eliminar un hecho o instancia.
- Modificar un hecho o instancia.
- Parar el sistema de producción, halt.
- Asignar un valor a variable.
- Operaciones de entrada/salida.
- Llamar a función.

Para indicar la variable se pone ? delante del nombre de la variable y ?$ si la variable es una lista.

En aquellos casos que queramos buscar por un atributo, pero no queramos almacenar el valor en variable se puede poner ? o ?$ solo.

Al definir reglas ya existentes, la primera regla se borra.

2.3.1. Elementos de condición (LHS)

- **Direcciones de hechos:** Almacenar la dirección de hechos en variables.

  ?P <- (persona (nombre ?N) (amigos $?A) (edad 30))

- **Test:** Comprobar el cumplimiento de alguna condición.

  (test (>= (abs (- ?y ?x)) 3))

- **Not:** No existencia de un determinado hecho.

  (not (dato rojo ?x ?x))

2.3.2. Acciones

- **Parada:** Para la ejecución del sistema.
  (halt)
- **bind:** Asignar valor a variables.
  (bind ?X (* ?Y 2))

2.3.3. Comandos para reglas

(list-defrules) Ver las reglas definidas.

(ppdefrule nombre-de-regla) Ver la definición de la regla.

(undefrule nombre-de-regla) Eliminar una regla.

(undefrule *) Eliminar todas las reglas.

2.3.4. Ejecución de reglas

La colocación de una regla en la agenda se realiza según su prioridad y la estrategia de resolución de conflictos definida

(agenda) Para ver las reglas ejecutables.

(run) Ejecutar las reglas seguidas.

(run veces) Ejecutar <veces>reglas.

(matches nombre-regla) Para ver que hechos se equiparan con los patrones de una regla.

(watch <item>) Observación de la evolución de los componentes del sistema.

(unwatch <item>) Desactivar el modo de observación.

set-break y remove-break Para poner y quitar un breakpoint en la regla que se les pasa como argumento.

Los item son: all compilations statistic messages deffunctions rules facts activations

2.3.5. Estrategias de resolución de conflictos

Se seleccionan con (set-strategy <estrategia>). La estrategia por omisión es depth.

Cuando no se determina tratara de usar el hecho más nuevo, es decir el último generado o modificado.

- **depth (Profundidad):** Nuevas activaciones por encima de las de igual prioridad.
- **breadth (Amplitud):** Nuevas activaciones por debajo de las de igual prioridad.
- **simplicity:** Nuevas activaciones por encima de las activaciones con igual o mayor especificidad
- **complexity:** Nuevas activaciones por encima de las activaciones con igual o menor especificidad
- **random:** Aleatoria. A cada activación se le asigna un número aleatorio para determinar su orden en la agenda.

2.3.6. Consideraciones

No se trata de una estructura if-then-else, sino if-then. Si se quiere else se crean 2 reglas.

No puede haber OR en el consecuente, para suplirlo se crean 2 reglas.

No DEBEN aparecer OR en el antecedente, se crean 2 reglas con una parte del OR en cada una.

No se pueden anidar elementos de decisión (ifs), para esto se pueden usar señalizadores que activen reglas, no olvidar borrarlos.

Desde una regla no se puede lanzar otra.

2.4. Funciones

- **open:** Abrir un fichero, en nombre de canal se puede poner una variable.

  (open nombre-fichero nombre-canal-asignado modo-acceso)

- **close:** Cerrar un fichero.

  (close nombre-canal)

- **read:** Leer de un canal.

  (read <nombre-canal>)

- **printout**

  Por pantalla: (printout t “Hello there!” crlf)

  En fichero: (printout file-name “red green”)

- **eq:** TRUE si los argumentos son iguales en valor y tipo, si no FALSE.
- **neq:** TRUE si alguno de los valores de las expresiones argumento es distinto, si no FALSE.
- **=, <>, <, >, <=, >=:** Funciones para argumentos numéricos.
- **and:** TRUE si los argumentos son TRUE.
- **or:** TRUE si alguno de los argumentos es TRUE.
- **not:** TRUE si el valor del argumento es FALSE.

Modos de acceso:

- “r” solo lectura.
- “w” solo escritura.
- “r+” lectura y escritura.
- “a” escritura al final de fichero (append).

Canales predefinidos: stdin, stdout, werror.

2.5. Marcos

Se utilizan en aquellos casos de estructuras más complejas que puedan requerir herencia, para los sencillos se usan las plantillas.

Declaración: (defclass nombre (is-a nombre-padre) propiedades slots-o-multislots)

Propiedades de la clase: (role abstract/concrete) para que no se puedan crear instancias o si y (pattern-match non-reactive/reactive).

Los slots de las clases son los mismos que las plantillas, pero además están (create-accesor read-write) para modificarlo y (source composite) heredar todas las facetas del mismo slot que el padre para redefinir slots del padre.

El nombre de la clase padre inicial es INITIAL-OBJECT.

2.5.1. Comandos para instancias

- **make-instance:** Añade una instancia.
  (make-instance of INTRODUCCIÓN (primitiva ESTAR))
- **unmake-instance:** Quita la instancia.
  (unmake-instance ?ins)
- **modify-instance:** Borra y crea instancias de la base de hechos.
  (modify-instance ?ins (orden 7))
- **instances:** Muestra las instancias de la base de hechos.
  [gen1] of INTRODUCCIÓN
- **(send [gen1] print):** Para ver valor de los slots.

3. TEMA 1: INTRODUCCIÓN

3.1. Definición de Ingeniería del Conocimiento

Ingeniería de conocimiento: Es la actividad de construir Sistemas Basados en el Conocimiento (SBC). Es el proceso para adquirir, estructurar, conceptualizar y después implementar un conjunto de conocimientos.

El conocimiento sobrevive a implementaciones y valorado en sí mismo.

Debe considerar la escalabilidad del sistema y el mantenimiento de este.

3.1.1. Pirámide de Datos/Información/Conocimiento

El pico es el metaconocimiento, en esta asignatura trataremos el conocimiento.

Datos: Gran volumen, pero de bajo valor, carecen de significado o contexto que los hagan más valiosos.

Información: Menor volumen, tienen valor al tener un contexto o significado asociado.

Conocimiento: Entender el dominio, pudiendo aplicarlo para resolver problemas. Se basan en la información que sale de los datos, pero este no se trata de meros datos sino de lo que subyace sobre estos.

Metaconocimiento: Es el conocimiento del conocimiento.

3.2. Sistemas Basados en el Conocimiento y Sistemas Expertos

Sistema Basados en el Conocimiento (SBC): Es un tipo de sistemas software, que tratan problemas cuyo método de resolución es más heurístico (guía hacia la solución de una manera más o menos buena, mediante el conocimiento previo del problema, en forma de una serie de reglas) que algorítmico y que contiene conocimientos públicos de un dominio. Lo clave es que es heurística y contiene conocimiento.

Es un sistema que usa conocimiento específico de un problema e intenta codificar el razonamiento que hacemos los humanos, como si fuera un experto, es decir juntando los datos e ir sacando conclusiones hasta llegar a la solución.

El conocimiento se presenta de manera declarativa (explicita). Los SBC son más heurístico que algorítmico.

El primer sistema que se puede llamar SBC es el General Problem Solver (GPS): H. Simon y A. Newell, 1957

3.2.1. Actores participantes

- **Experto:** Es el que posee el conocimiento, como podría ser un libro.
- **Usuario:** Es el que usara el conocimiento.
- **Ingeniero de conocimiento:** Es el que se encarga de formalizar el conocimiento para que sea utilizado
- **Desarrollador**
- **Gestor del proyecto**

3.2.2. Diferencias entre un SBC y un SST

SST: Sistema de Software Tradicional.

En arquitectura: Los SBC siempre separan la parte de datos y la parte de resolución, sin embargo en SST están mezcladas.

En tipo de problemas: Los SBC resuelven problemas heurísticos y los SST dan soluciones algorítmicas.

En técnicas de programación: SBC emplea programación declarativa y los SST programación imperativa.

Lo más importante es que los SBC son capaces de explicar cuál es el proceso de razonamiento que ha empleado para llegar a la solución. Es posible al ser declarativo y la forma en la que se implementan permite ver qué proceso es el que ha seguido.

3.2.3. Ingeniería del Software vs. Ingeniería del Conocimiento

	Ingeniería del Software	**Ingeniería del Conocimiento**
Dominios	Tratamiento eficaz de los datos	Conocimientos especializados
Tipo de problemas	Sistemáticos y procedimentales	Heurísticos y declarativos
Entrada	Datos, procesos y algoritmos	Conocimiento humano
Técnica de programación	Procedimentales sofisticados secuenciales y rígidos	Declarativos, elementales paralelos y flexibles
Modelo de solución de problemas	Implícitamente en el código del programa	Explícitamente en la Base de Conocimiento
Niveles organización conocimientos	1. Datos 2. Programas	1. Datos o hechos 2. Reglas operativas o heurísticas 3. Inferencia y control

Tabla 3.1: IS vs. IC

3.2.4. Elementos de un SBC

Base de conocimiento: contiene el conocimiento que hemos recibido de los expertos en un dominio determinado.

Base de hechos: Cómo está la base de conocimiento, pensando como el estado, es la memoria de trabajo que almacena los datos iniciales del problema y los resultados intermedio durante el proceso.

Motor de inferencia: Es el que se encarga de ir pasando de estados aplicando reglas, imitando el procedimiento humano de los expertos, poco a poco se va llegando a la solución. Aprenderemos cómo funciona y tendremos que definir las reglas.

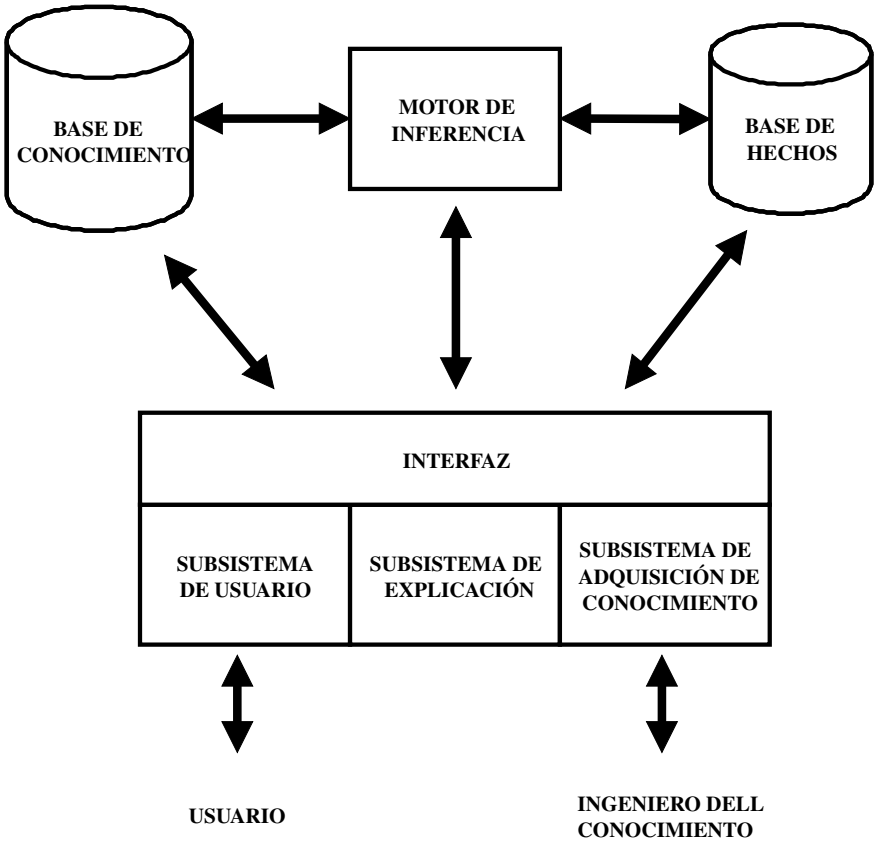


Fig. 3.1: Elementos de un SBC

3.2.5. Fases para construir un SBC

Iteración:

1. **Identificar** el problema (con usuario y expertos)
2. Adquirir conocimiento (con expertos y estudio de la documentación)
3. **Conceptualizar**, saber estructurar el conocimiento, separar los datos y el problema.
4. **Formalizar** conocimiento.
5. **Implementar** formalización (con desarrollador).
6. **Validar** funcionamientos esperados (con usuario y experto).

Es un proceso que no se realiza a la primera, se retrocede continuamente, además se deben hacer varias iteraciones en todos los pasos, es difícil realizarlos a la primera al ser fácil pasarse algo del problema.

3.2.6. Ventajas de los SBC

- Distribución y disponibilidad del conocimiento.
- Coherente y estabilidad.
- Almacenamiento del conocimiento
- Resuelven el problema con incertidumbre.
- Explicación de como adoptará la solución.
- Aprendizaje.

3.2.7. Desventajas de los SBC

- No asegura que se vaya a encontrar la solución óptima, pero al menos alcanzara una.
- El conocimiento es limitado, no se puede poseer todo el conocimiento, hay que asumir esta limitación y trabajar con ella.
- Falta de sentido común.
- La adquisición del conocimiento y mantenimiento es difícil.
- Degradación del conocimiento, requiere un mantenimiento constante para mantenerse al día.

3.3. Ejemplos Actuales Relacionados con la Ingeniería del Conocimiento

Sistema Experto en Medicina: en el que con todo el conocimiento médico y recibiendo una serie de variables ser capaz de dar un diagnóstico. En este punto entra en juego la ética y el querer saber en qué se basa este diagnóstico, como lo obtuvo.

- ATHENA: Assessment and Treatment of Hypertensión
- GIDEON: Global Infectious Disease and Epidemiology Network

Sistema de Gestión de Reglas de Negocio: Dado el conocimiento adquirido por profesionales e históricos (como en el sistema financiero) ser capaz de decidir cuándo invertir o realizar alguna operación.

- IBM ILOG JRules BRMS
- Drools
- Red Hat JBoss BRMS

Ontologías:

- Web semánticas: Web Ontology Languaje
- Protégé: Editor de ontologías
- Información de seguimiento de una persona

Planificación Automática y Robótica: Como por ejemplo es el proyecto Nao Therapist un robot para ayudar a los niños en rehabilitación por parálisis cerebral.

4. TEMA 2: IDENTIFICACIÓN Y ADQUISICIÓN

Se tratará la identificación de los problemas y la adquisición del conocimiento para los SBC que se componen de:

- **Conocimiento:** Formado por la Base hechos (define el estado del sistema) y la Base de reglas (como razona el sistema experto).
- **Motor de inferencia** que toma las decisiones como lo haría el experto humano, como si razonase.

Una parte muy importante del proceso es limpiar los datos para generar conocimiento.

4.1. Identificación del problema

Es la primera fase para el desarrollo de un SBC, saber cuál es el problema y si es posible aplicarle esta técnica. También se espera que se defina porque resolver este problema es útil.

Etapas de la identificación

1. **Objetivos del problema:** que se quiere resolver y el plan de requisitos. Los 3 primeros son los pedidos para las prácticas.
    - **Metas específicas y generales**, a que se quiere llegar y partir el problema.
    - **Funcionalidad y rendimiento esperado**, como el tiempo de respuesta esperado.
    - **Fiabilidad y calidad**, que serie de problemas debe ser capaz de superar para considerarlos de calidad.
    - Limitaciones de coste/tiempo.
    - Requisitos de fabricación/implementación.
    - Tecnología/medios disponibles.
    - Ampliaciones futuras.

2. **Evaluar el problema:** ver si de verdad es necesario y posible crear el SBC (tampoco matar moscas a cañonazos). Para la práctica se nos piden las 2 primeras:

   - **Adecuación**, se ajusta al tipo de problema. Nos tenemos que fijar en:

     Naturaleza del problema

     - Requiere experiencia, si lo hace es adecuado.
     - Implica el uso de diferentes tácticas, según los hechos irá por una táctica u otra.
     - Es una actividad práctica, se aprende por observación, no solo conociendo el manual.
     - Satisface necesidades a largo plazo, es útil para más de 1 vez o caso.

     Tipo:

     - La tarea no requiere investigación, si la requiere no habrá expertos que consultar.
     - Solo manipulación simbólica
     - Hace uso de heurísticas, no se conoce el procedimiento exacto, son aproximados y sabrá recuperarse.

     Complejidad:

     - Dominio acotado y problema puede descomponerse (los juegos suelen serlo)
     - Problema no demasiado fácil (no algorítmica, solución necesita justificación)
     - Transferencia de experiencia entre humanos es factible. Que será posible la transferencia de conocimiento.
     - No requiere tiempo real, porque suelen ser lentos.

   - **Plausibilidad**, ver que es posible.
     - Existen verdaderos expertos y son accesibles
     - Los expertos pueden explicar sus métodos de resolución del problema
     - La tarea solo depende de los conocimientos
     - La definición del problema está clara y estructurada
     - Hay suficientes y relevantes casos de prueba
   - Justificación. Normalmente si cumple las 2 anteriores es suficiente.
     - Solución más sencilla
     - Entorno peligroso o poco gratificante
     - Escasez de experiencia humana
     - Perdida de experiencia humana
     - Necesidad de omnipresencia simultánea en lugares distantes

        - Rentabilidad económica
        - Utilidad y necesidad
        - Inexistencia de alternativas.
    - Posibilidades de Éxito.
        - Expertos autosuficientes (conocimiento repartido de calidad, activos, cooperativos y coherentes)
        - Gestores del proyecto convencidos (prioridad proyecto, resultados aceptables, no perfectos . . .)
        - Ausencia de rechazo en los usuarios
        - Suficientes recursos (humanos y materiales)
        - Dominio estable, objetivos claros y evaluables
        - Precedentes de uso de IC en problemas semejantes
        - Las soluciones son explicables

3. **Definir el problema y concebir una solución**, como debería ser la solución. Ayuda a pensar en las entradas y sus salidas.
    - **Definición y descomposición** del problema. Descripción de problemas similares.
    - Descripción de cada subproblema.
    - Identificación de las **entradas y salidas** de cada subproblema.
    - Características de datos de entrada: incompletos, imprecisos, inciertos. Permiten tratar la incertidumbre.
    - Enumeración de casos y ejemplos con los que se cuenta, poner casos y ejemplos para visualizarlo.
    - Describir y cuantificar la complejidad prevista del problema.
    - Concepción global de la solución: esbozar a priori posibles soluciones.

4.2. Adquisición del Conocimiento

Esta fase se repite durante todo el proceso en paralelo para ir completando el sistema, ya que nos daremos cuenta de que hay cosas que nos hemos saltado.

Fuentes de conocimiento, pueden ser orales o escritas:

- **Documentación formal:** libros y manuales, revistas especializadas, etc.
- **Documentación informal:** presentaciones, apuntes, registros internos
- **Bases de datos:** EXTRACCIÓN de conocimiento

- **Expertos humanos:** EDUCCIÓN de conocimiento

Es importante saber separar el conocimiento factual de las estrategias (suele estar en la BH), el qué, y el procedimiento de resolución (suele estar en la BR), el cómo. Esta separación nosotros la realizaremos manualmente, pero hay métodos que lo hacen automática o semiautomáticamente.

4.2.1. Extracción de Conocimiento desde la Documentación

Estudio inicial de documentación: Consiste en la búsqueda de terminología, identificar **conceptos básicos**, y comprender la naturaleza de la **tarea**.

Análisis estructural de textos: Buscar términos dependientes del dominio y estructuras textuales mediante la detección de patrones.

Estructuras Textuales

Están compuestas por 4 elementos:

- **Definiciones:** Introducción de un concepto que puede estar:
  - Definido por su uso: se extrae el concepto y una característica.
  - Definido por las partes que lo componen: se extrae el concepto principal, los componentes y la relación formar-parte-de.

  (se usa para, es un, está compuesto por)
- **Afirmaciones:** Establece una verdad.

  (es causa de, se relaciona con, es la finalidad de)
- **Leyes:** Principios básicos.
- **Procedimientos:** Pasos para la resolución.

4.2.2. Educción de Conocimiento

Psicológicamente un experto:

- Analiza lo que percibe.
- Asocia lo percibido con los conceptos del dominio.
- Encuentra causas basándose en relaciones de conceptos.
- Infiere y verifica hipótesis.

Paradoja de la experiencia: los expertos más competentes son incapaces de describir los conocimientos que usan para resolver los problemas.

Técnicas de educción: ayudan a adquirir y validar el conocimiento de los expertos. No hace falta conocer todas las técnicas.

- **Técnicas naturales:** Implican al experto realizando tareas que haría normalmente como parte de su trabajo.

  Entrevistas (Abiertas, Semiestructuradas y Estructuradas), Cuestionarios, Técnicas de observación y Técnicas de grupo.

- **Técnicas artificiosas** Implican al experto realizando tareas que no realizaría normalmente como parte de su trabajo.

  Método triádico, Clasificación de conceptos, Emparrillado, Tareas con restricciones, 20 cuestiones, 'Teach back' y Análisis de Protocolos.

  Análisis de Protocolos o caso concreto de forma global

  Tiene tres profundidades: Análisis del asunto o materia, Análisis de tareas y Distinguir casos dentro de la materia y la tarea asociada.

  Etapas:

  - Grabación del protocolo, tener un registro de cómo se lleva a cabo.
  - Transcripción del protocolo segmentándolo en instrucciones.
  - Codificación del protocolo:
    - Identificar conceptos, características, valores, relaciones.
    - Identificar el razonamiento del experto.
    - Identificar acciones externas.
    - Identificar las inferencias.
    - Detección de sinónimos, metacomentarios e incertidumbres.
  - Interpretación del IC sobres reglas implícitas, planes y estrategias del experto.

- **Técnicas de modelado:** Técnicas basadas en el uso de modelos que guían la adquisición, validación y modificación del conocimiento. Representar el conocimiento para conceptualizar.

  Conceptos, Procesos y Estados.

  Tipos de modelado: Diagrama de proceso (conocimiento procedural), Mapa conceptual (conocimiento declarativo) y Diagramas de estados (variación de los diagramas de procesos, buenos para todo tipo de conocimiento.)

Reglas para una Buena Educción

- Conseguir la cooperación del experto.
- No imponer la visión del IC sobre la del experto.
- Limitar las sesiones en duración y contenido.
- Intentar comprobar la información.
- No limitarse al diálogo, usar textos.
- No espaciar demasiado las sesiones de educción.
- Elegir el lugar adecuadamente.
- No ser el propio experto.
- No creer todo lo que dice que hace.

Ciclo de Educción

El conocimiento se educe mediante la sucesiva aplicación de sesiones de educción.

Para completar una sesión el ingeniero del conocimiento debe realizar los siguientes pasos:

- Preparación de la sesión
- Ejecución
- Transcripción
- Análisis de la sesión

  Organización de la información: Diccionario de términos o conceptos, Librería de casos y Documento de conocimientos.

  Actualización tras el análisis de cada sesión: determinar comprensión del IC, y si es necesario preguntar más sobre el tema. Producir documentos para fases posteriores.

  Proceso de razonamiento de resolución: Búsqueda de inferencias, Analizar casos buscando, Comprobación de suposiciones obvias y Analizar si es necesario el uso de la incertidumbre.

- Evaluación

5. TEMA 3: CONCEPTUALIZACIÓN. INTRODUCCIÓN

Consiste en estructurar todo el conocimiento adquirido y establecer que conocimientos maneja el experto, como los utiliza, donde y cuando.

Es independiente de la implementación y herramienta.

Análisis: estructurar...

- **Conocimiento factual**, la información que se conoce (conceptos)
- **Conocimiento táctico**, como y cuando se añade información (inferencia)
- **Conocimiento estratégico**, que hacer, donde y por qué (pasos)
- **Metaconocimiento**, conocimiento sobre el conocimiento.

Síntesis: modelo conceptual

- **Modelo factual:** Tabla Conceptos-Atributos-Valores y Relaciones.
- **Modelo de procesos:** Inferencias, tareas, procedimientos, estrategias, control, restricciones y operaciones que actúan sobre el modelo estático.
- **Mapa de conocimiento:** Integra los dos modelos.

5.0.1. Concepto

Cualquier cosa que se nos ocurra acerca de la cual se quiere decir algo, pueden ser cosas concretas, abstractas, de ficción, descripción de una tarea, función, acción, estrategias o tareas de razonamiento.

Aunque se pueden generar infinidad de conceptos es conveniente reducir el número de conceptos mediante:

- **Generalización:** Considerar el conjunto que contiene un concepto, de manera que este contenga a muchos conceptos. Ejemplo: Herramienta con martillo, sierra, destornillador...
- **Abstracción:** Concentrarse en las cualidades esenciales de algo, en lugar de en sus relaciones concretas o casos.

Universo del discurso: Conjunto de conceptos acerca del cual se expresan los conocimientos.

Tabla Conceptos/Atributos/Valores

Concepto	Atributo	Posible valor
Herramienta	Nombre	Martillo1, Sierra1, Taladro1, Cincel1...
	Modelo	Un nombre
	Coste	Un número
	Fecha de compra	dd/mm/aa
	Fabricante	Un nombre
	N.º de registro	Un n.º de 7 dígitos
	Color	Rojo, Verde, Blanco ...

Tabla 5.1: Ejemplo Tabla Conceptos/Atributos/Valores

Las relaciones se pueden representar como concepto y las partes implicadas como atributos, si existen más características en la relación serian atributos extras.

5.0.2. Relaciones entre conceptos

Interacción entre conceptos en un universo de discurso.

Las relaciones más habituales son:

Tipo	Ejemplos
Taxonómica	A es B. A puede clasificarse como un B, un C, o un D
Estructurante	A es parte de B. A está formado por B y C
Topológica	A esta encima de B. A contiene a B. A intersecta a B
Causal	A causa B. A es efecto de B. Siempre que A, B
Funcional	A permite B. A necesita B. A dispara B
Cronológica	A sucede antes de B. A y B ocurren simultáneamente
Similaridad	La válvula A esta abierta = admite gasolina
Condicional	Si A y B entonces B
Finalidad	Los códigos se crearon para transmitir ideas

Tabla 5.2: Tipos de relaciones entre conceptos

5.0.3. Atributos

Propiedad o característica de un concepto.

Lista atributos: lista de elementos necesarios para inferir la/s meta/s del SE.

En el ejemplo de la herramienta los atributos podrían ser: Nombre de la herramienta, Número de registro, Coste, Fecha de compra, Fabricante y Modelo.

Para identificar los atributos nos podemos preguntar:

¿Es *sustantivo1* un atributo del concepto *sustantivo2*? ≡ ¿Tiene sentido hablar acerca del *sustantivo1* de un *sustantivo2*?

Definición de un Atributo

Los atributos de los atributos se llaman facetas y deben ser definidos sobre los atributos de la Tabla Conceptos/Atributos/Valores.

Los fundamentales son los que están en negrita.

INFORMACIÓN	**DESCRIPCIÓN**
Nombre	Termino estándar que nombra ese atributo
Descripción	Breve explicación de lo que representa ese atributo
Tipo Valor	Que puede contener el atributo (Entero, real, cadena, símbolo, ...)
Rango de valores	Una lista de posibles valores Un rango de valores numéricos
Cardinalidad	Máxima y mínima que puede tener (normalmente uno o muchos)
Valor por defecto	En caso de no especificar un valor
Grado de certeza	En caso de incertidumbre

Tabla 5.3: Definición de un Atributo

5.0.4. Relaciones entre Conceptos

Dependencia entre conceptos. Ejemplo: Herramienta y Persona la relación es de Propietario

Es necesario que exista alguna instancia de los conceptos padres.

Tipos de relación: Uno-a-Uno y Uno-a-Muchos.

Las relaciones y conceptos se pueden representar en un Mapa Conceptual, las cajas son los conceptos y las flechas son las relaciones en las que se indica el nombre y tipo.

5.0.5. Conocimientos Estratégicos

1. Definir pasos de alto nivel para realizar la tarea: Identificar los pasos de alto nivel a ejecutar, Establecer el orden y Definir las condiciones en que debería ejecutarse cada paso.
2. Descomponer pasos de alto nivel en subpasos.
3. Para cada subpaso descubrir: Entrada, Modo de razonamiento y Salida.

5.0.6. Conocimientos Táctico. Inferencias

Una inferencia consta de una parte de condición y otra de conclusión.

Hay múltiples posibles representaciones, pero la más habitual es:

- **SI** condiciones **ENTONCES** acciones

Paso de inferencia: Hay que indicar que atributo/s usar para inferir nueva información (concluir un atributo específico)

6. TEMA 4: CONCEPTUALIZACIÓN. ONTOLOGÍAS

Una ontología es una especificación formal y explicita de una conceptualización compartida (Thomas Gruber), nosotros las utilizaremos para conceptualizar normal.

- **Conceptualización:** desarrollo de un modelo abstracto del dominio basado esencialmente en el empleo de conceptos, atributos, valores, relaciones y funciones.
- **Especificación explicita:** descripción y representación de un dominio concreto mediante conceptos, atributos, valores, relaciones y funciones, definidos explícitamente.
- **Formal:** expresado mediante un formalismo (siempre idéntico) de manera que pueda ser reutilizada y leída por cualquier máquina.
- **Compartida:** favorablemente acogida por todos los usuarios de esta. Se supone que se comparten entre usuarios para usarla con diferentes propósitos.

6.0.1. Características de las Ontologías

- Disponibilidad, facilidad de gestión y divulgación
- Ingeniería ontológica
  - **Ciclo de vida:** construcción, refinamiento, modificaciones, uso, explotación y retiro.
  - Requieren metodologías.
- No son cerrados y están sujetos a procesos evolutivos.

6.0.2. Algunos problemas

- Imposibilidad de utilización directa de las ontologías previamente desarrolladas, almacenan el conocimiento junto, pero para hacer razonamientos hay que extraerlo (con un parser) y adaptarlo.
- **Web semántica:** se necesita desarrollar buscadores semánticos.

6.0.3. Terminología Básica

- **Clase:** representa un concepto.
- **Subclase:** concepto más específico que dicha clase, cuando se da es-un, no cuando se da la relación es-parte-de o compone-a, en esos casos se crea su propia clase o se pone como atributo.

  Las subclases heredan todas las propiedades de la clase.

  Si A es subclase de B, una instancia de A también lo es de B.

  Una nueva subclase puede tener nuevas propiedades, diferentes valores de las propiedades y participar en nuevas relaciones. Aunque no tienen por qué introducir nuevas propiedades.
- **Clase abstracta:** clase que no permite que existan instancias de ella. Se usa para agrupar conceptos e introducir cierto orden en la jerarquía.
- **Propiedad o atributo (slot):** característica de la clase que se concretara mediante un valor, pueden ser tipos básicos o incluso otra clase o instancia.
- **Faceta:** propiedad de los atributos. Tipo de dato, la cardinalidad, valor por defecto...
- **Instancia:** representan objetos concretos del dominio.
  - Pertenecen a una clase.
  - Todos los atributos toman un valor concreto.
  - La colección de instancias constituye la base de hechos.
- **Relación:** interacción o enlace entre los conceptos o clases del dominio.
  - Relaciones semánticas básicas: subclase de ('es un'), parte de...
  - Se pueden modelar mediante atributos de una clase o definiendo clases nuevas
- **Axioma:** reglas que son siempre ciertas. Se usan para Incluir restricciones, Verificar la corrección o Deducir conocimiento no codificado explícitamente.
- **Herencia:** propiedad de la relación 'es un', las clases relacionadas (heredadas) cuentan con los atributos de la clase padre.
- **Herencia múltiple:** una clase hereda o cuenta con las propiedades de dos clases padre.
- **Derivación:** organización de las clases de la ontología en un árbol de jerarquía mediante sucesivas relaciones 'es un' (kind of, is a) con la propiedad de herencia.

6.0.4. Metodologías para Hacer Ontologías

Son las metodologías para que dado un problema diseñar una ontología que lo represente.

- **Diligence:** basada en la colaboración de múltiples participantes
- **Competency Questions ('Preguntas Relevantes o 'Preguntas de Verificación'):** determina el dominio y el alcance de la ontología mediante la lista de preguntas
- **Methontology:** propone un ciclo de vida de construcción de la ontología basado en prototipos evolutivos
- **On-To-Knowledge:** hace hincapié en las aplicaciones futuras de la ontología a la hora de diseñarla.

6.0.5. Desarrollo de una Ontología

Algunas reglas:

- No existe un único modo correcto de modelizar un dominio.
- El desarrollo de una ontología es necesariamente un proceso iterativo, se va refinando.
- Los conceptos o clases de una ontología deberían ser muy cercanos a objetos o entes (tanto físicos como lógicos) y a las relaciones existentes en nuestro dominio de interés.

Pasos:

1. Determinar el dominio y alcance de la ontología
    - ¿Qué dominio cubrirá la ontología?
    - ¿Para qué se va a emplear la ontología?
    - ¿Qué preguntas debería contestar la ontología?
    - ¿Quién usará y mantendrá la ontología?
2. Considerar la reutilización de ontologías existentes
    - Ontolingua
    - DAML Library
    - UNSPSC
    - RosettaNet
    - DMOZ

3. Enumerar términos importantes de la ontología

   Listar todos los términos del dominio basándose en documentación o expertos.

   Al principio ignorar: relaciones, varios se enmarcan en un mismo concepto (sinónimos), o si corresponden a conceptos o propiedades.

   Algunos de estos términos se transformarán en clases, otros en propiedades y otros en instancias.

4. Definir las clases y la jerarquía de clases

   - **Top-down:** comienza con las clases **más generales** del dominio y posteriormente se lleva a cabo su especialización
   - **Botom-up:** comienza con la definición de las clases **más específicas**, reagrupando posteriormente estas clases en conceptos más generales
   - **Combinación de las anteriores:** comienza con la enumeración de las clases **más destacadas y posteriormente se generalizan** y especializan apropiadamente

5. Definir las propiedades/slots de las clases

   Características que pueden convertirse en slots: Características intrínsecas, extrínsecas, Partes del objeto y Relaciones con otras clases.

   Relaciones inversas: el valor de una propiedad puede depender de otra propiedad. Ej. Vino fue fabricado por una bodega, entonces, la bodega produce ese vino.

6. Definir las facetas o restricciones de las propiedades o slots

   - **Tipo de valor:** String, Número (Entero o Float), Boolean, Enumerado, Instancias
   - **Cardinalidad:** cuantos valores puede tener una propiedad
   - **Rango:** clases permitidas para una propiedad de tipo Instancia, valores numéricos o valores permitidos.
   - **Valor por defecto:** Si la mayoría de las instancias de la clase toman un valor concreto, el sistema lo completara automáticamente, pero se puede cambiar el valor posteriormente. No es lo mismo que el valor fijado.

7. Crear instancias

   Las instancias son los objetos más específicos representados en una ontología: Se debe elegir una clase para la que crear la instancia y rellenar todos los valores de propiedades.

6.0.6. Jerarquía de clases

- Las instancias son los conceptos más específicos representados en una ontología.
- No debemos emplear sinónimos de un mismo concepto para clases diferentes.
- No deben aparecer ciclos o bucles en la jerarquía de clases.
- Los conceptos de un mismo nivel de la jerarquía o clases hermanas deben presentar el mismo nivel de generalidad (excepto en la raíz).
- Dominio del discurso: no es necesario especializar o generalizar más de lo que necesitemos.
- Si los conceptos de un dominio forman una jerarquía natural, podríamos representarlos como clases abstractas.

6.0.7. Herramientas de Edición de Ontologías

- Protege (Universidad de Stanford)
- KAON (Universidad de Karlsruhe, Alemania)
- WebOnto (Open University, UK)
- OntoEdit (comercializada por Ontoprise GmbH)
- Ontolingua (Universidad de Stanford)
- OntoSaurus (Universidad de Southern California). Lenguaje LOOM

7. TEMA 5: FORMALIZACIÓN E IMPLEMENTACIÓN. REPRESENTACIÓN DEL CONOCIMIENTO

Modelización del problema desde el punto de vista del sistema.

Un modelo formal es una representación computable (entendible por la máquina) de los conocimientos y conducta del experto.

Formalizar consiste en: Representar simbólicamente los conceptos mediante alguno de los formalismos existentes, organizarlos de acuerdo con algún modelo de diseño y determinar los métodos de inferencia.

El modelo formal debe contener los conocimientos del modelo conceptual y se expresa mediante el uso de un formalismo de representación del conocimiento.

Se puede suponer la existencia de un motor de inferencias (programación propia o herramienta) independiente del dominio de aplicación

Normalmente la técnica de resolución del problema viene dada por el formalismo de representación elegido. En sistemas grandes se pueden combinar varios formalismos.

Algunos formalismos son:

- **Basados en acciones:** Sistemas de producción y guiones.
- **Basados en conceptos:** Marcos.
- **Basados en relaciones:** Lógica y Redes semánticas.

Cuando se formaliza hay que pensar en el problema para poder representar las preguntas correctamente (que la representación facilita las reglas).

La representación en el ordenador no siempre refleja la realidad.

Todas las representaciones son imperfectas.

Una misma información se puede representar de múltiples formas.

La representación que se utilice puede facilitar las inferencias, no es lo mismo representar (Red car12), que (car12, Red) o (Propiedad color car12 Red).

8. TEMA 6: FORMALIZACIÓN E IMPLEMENTACIÓN. SISTEMAS DE PRODUCCIÓN

8.0.1. Componentes de un Sistema de Producción

Base de hechos o memoria de trabajo (BH o WM): conocimiento sobre el dominio en un determinado momento

Base de reglas (BR): conjunto de reglas (producciones). SI Condiciones ENTONCES Acciones

Estrategia de control, interprete de reglas, o **motor de inferencias (EC o MI):** responsable de encadenar los ciclos de funcionamiento.

Base de Hechos

Almacena el estado actual de la tarea o problema

Los datos almacenados pueden ser permanentes, temporales, generales y específicos

Inicialmente se representa todo el problema con los hechos iniciales, se va desarrollando hasta llegar finalmente a un estado meta.

La BH presenta tres estados:

- **Estado inicial:** situación origen del problema.
- **Estados metas:** situación objetivo.
- **Estados intermedios:** representan en cada instante, el estado del problema en curso de solución.

Base de Reglas

Está el conocimiento de cómo cambiar entre estados.

La mayor parte de los conocimientos sobre la resolución del problema se representan en un conjunto de reglas o producciones.

Cada regla presenta la forma: **SI** Condiciones **ENTONCES** Acciones

- **Condición o antecedente:** lista de cosas a verificar en la BH
- **Consecuente o acción:** conjunto de acciones a realizar sobre la BH

Las reglas van cambiando el estado muy poco a poco, no tratar de hacer una regla que lo haga todo.

Motor de Inferencia

Examina en cada ciclo de funcionamiento la BH y decide que regla ejecutar, encadenando las reglas en los ciclos de resolución.

Funcionamiento:

1. **Fase de decisión:** selección de la regla.
   - (Etapa de restricción (opcional))
   - **Etapa de equiparación o filtrado.** Coge todas las reglas y comprueba las precondiciones con los hechos y ver cuál se puede ejecutar.
   - **Etapa de resolución del conjunto conflicto.** Según la estrategia, selecciona una regla.

   Una regla se activa cuando sus precondiciones son ciertas en el estado actual de la BH
2. **Fase de acción:** ejecución de la regla.

Las reglas se pueden hacer en dos sentidos:

- **Encadenamiento hacia Adelante:** Partiendo de los hechos se va viendo que reglas se pueden ejecutar y utilizando las salidas y los hechos generados se van encadenando las reglas hasta llegar a una meta.
- **Encadenamiento hacia Atrás:** Pensar desde el estado final (meta) que necesito para que retrocediendo se llegue a los hechos iniciales que se tienen.

Estrategias de Resolución del Conjunto Conflicto

- Primera regla.
- Más prioridad.
- Más específica.
- Más general.
- Aleatoriamente.
- ...

8.1. Ejercicios

1. **Se tienen 5 monedas dispuestas como se muestra a continuación: ARARA**

   El anverso de la moneda está representado por A y el reverso por R. En cada movimiento se puede dar la vuelta a cualquier par de monedas contiguas. La situación final que se quiere obtener es la siguiente: RRRAR.

   Se pide:

   - **Descripción de la Base de Hechos**
   - **Descripción de las reglas que forman la Base de Reglas**
   - **Determinar las reglas que se ejecutan para pasar a la situación final**

   Base de Hechos

   - (moneda A 1)
   - (moneda R 2)
   - (moneda A 3)
   - (moneda R 4)
   - (moneda A 5)
   - (inverso A R)
   - (inverso R A)

   Base de Reglas

```
(defrule volcar
        ?mn1 <- (moneda ?v1 ?p1)
        ?mn2 <- (moneda ?v2 ?p2)
        (inverso ?v1 ?nv1)
        (inverso ?v2 ?nv2)
        (test (= ?p1 (+ 1 ?p2)))
=>
        (retract ?mn1)
        (retract ?mn2)
        (assert (moneda ?nv1 ?p1))
        (assert (moneda ?nv2 ?p2))
)
```

   Regla final con los valores esperados para cada moneda.

2. **Imprimir por pantalla los n.º del 1 al 100.**

   Crear un contador con valor 1 y una regla, que lo que haga es ir sumando 1 al número mientras este sea menor que 100 (cuando sea 99 sumara por última vez y dará el 100)

3. **Contar el n.º de instancias de un tipo particular.**

   Crear un contador que parte de 0 e ira subiendo el valor según las va contando, cuando cuenta una instancia crea un hecho de que esa instancia se ha contado, esto es necesario dado que al actualizar el contador tratara de aplicar de nuevo la regla sobre los mismos hechos.

9. TEMA 7: FORMALIZACIÓN E IMPLEMENTACIÓN. MARCOS

Tenemos que representar todo el conocimiento en tablas del tipo objeto-atributo-valor.

Cada marco representa un prototipo de algo, debe representar un concepto, en el que cada slot es un atributo. Se diferencia entre:

- **Marco clase**: concepto (prototipo)
- **Marco instancia**: objeto (individuos de la clase)

Unificación de todas las propiedades de un objeto en la misma estructura y se establece una estructura de jerarquía con herencia.

Los desarrolló Minsky en 1975

- **Conceptos**: Marcos = Clases = Frames
- **Subconceptos**: subclases.
- **Herencia**: simple o múltiple.
- **Instancias**: objetos.
- **Atributos**: slots, campos, características...
- **Facetas**: atributos de atributos (valor, tipo, cardinalidad, ...)
- **Relaciones**: subclase-de (herencia), parte-de (entre atributos), instancia-de.
- **Demonios**: Funciones que vigilan las operaciones sobre los atributos. Esta al acecho de que ocurra un evento para realizar una serie de operaciones.
- **Métodos**: Funciones asociadas a los marcos, que facilita mucho la inferencia.

9.1. Descripción de un Marco

```
(CLASE o MARCO equivale a Concepto (es-un CLASE-PADRE)
        (Atributo1 equivale a SLOT O CAMPO
                (FACETAS)
                tipo de dato
                cardinalidad minima/maxima
                valor por defecto
                valores permitidos
                DEMONIO1 DEMONIO2 ...)
        (Atributo2 ...)
        ...
        (AtributoN ...)
        METODO1 METODO2 ...
)
```

9.2. Relaciones entre Marcos

Relaciones estándar:

- Son independientes del dominio
- Son dos: Subclase-de e Instancia-de

Relaciones no estándar:

- Representan dependencias entre conceptos de un dominio.
- Hay que buscar formas de implementarlas: atributos cuyo valor es una referencia a otro marco.

9.2.1. Relación Subclase-de

Se utiliza para la herencia, una clase hija hereda todos los atributos de la clase padre. Tener cuidado que no es una relación parte-de (elefante y trompa).

- Ej. Un elefante es un animal, pero una trompa NO es un elefante.

Especialización de conceptos generales en conceptos más específicos

Semántica: el concepto representado por M_h es una especialización o subconjunto del concepto representado por M_p.

Permite la herencia de propiedades: la clase hija tiene todas las propiedades (atributos) de la clase padre y de las clases anteriores en la jerarquía.

Definir los atributos comunes en la clase más abstracta y en las subclases los atributos particulares.

9.2.2. Relación Instancia-de

Representa un ejemplar o individuo de una clase.

Siempre están relacionados con un marco clase.

Tienen todos los atributos rellenos: Bien con valores específicos dado por el usuario o valores por defecto asignados por el sistema.

Los mecanismos de inferencias trabajan sobre instancias de marcos.

9.3. Facetas

- Tipo.
- Cardinalidad Mínima.
- Cardinalidad Máxima.
- Multivaluada.
- Propiedades que se rellenan en las instancias.
  - Valor por omisión (defecto).
  - Valores permitidos.
  - Demonios: procedimientos o reglas que se ejecutan automáticamente.
- Propiedades de clase/relación.
  - Propiedad general: valor o puntero a marco.

9.4. Inferencia en un Sistema de Marcos

1. Con reglas.
2. Usando métodos y demonios.
3. Combinación de ambas.

9.5. Relaciones

El uso de clases/marcos nos permite no solo crear una jerarquía entre conceptos, sino además generalizar a la hora de definir las reglas de inferencias pudiéndonos referir a la clase padre para operar con los hijos.

A la hora de relacionar conceptos, no se crea un slot con un puntero hacia la otra instancia, sino que utilizaremos identificadores en las instancias para que a la hora de referirnos lo hagamos de esta manera. Cuando se hace en una regla nos permite obligar a que la instancia relacionada tenga ese identificador.

Cuando la relación es 1 a 1 se puede hacer como se ha descrito antes, pero cuando se trata de una relación 1 a n o n a n, lo que se hace es crear una clase cuyas instancias estarán compuestas por los identificadores relacionados de manera que se recojan todas. (produce bodega vino1) (produce bodega vino2) aunque en este caso se podría haber puesto en vino, pero si fuese necesario en bodega se haría de esta manera.

9.6. Métodos

Funciones definidas sobre una CLASE para manipular los atributos.

Los métodos se definen de forma genérica en la definición del marco, pero se invocan (ejecutan) a través de una INSTANCIA de la clase.

Posible pseudocódigo:

- **send** o enviar: llama a algún método de una INSTANCIA.

  send ?alumno matricular ?self Se manda así mismo, instancia como argumento de la función matricular de la clase alumno

- **get**: coge el valor de un atributo de una instancia.

- **set**: da valor al atributo de una instancia.

- **add** o añadir: añadir: añade un valor a un multislot.

  añadir ?self.alumnos ?alumno

- **self**: referencia a la propia instancia.

  ?self.numero_alumnos = ?self.numero_alumnos + 1

No nos aprenderemos la sintaxis de CLIPS, usaremos pseudocódigo, incluso en el examen. Con los métodos anteriores es suficiente para realizar las tareas.

- Primero se define como se invoca el método, nombre y argumentos.
- Después los pasos que sigue el método, lo que devuelve o modifica sobre la base de hechos.

Ejemplo:

```
Metodo ASIGNATURA.matricular (?alumno in ALUMNO)
        anadir ?self.alumnos ?alumno
        ?self.numero-alumnos= (?self.numero-alumnos) + 1
        enviar ?self.titulacion matricular ?alumno
        enviar ?alumno matricular ?self

Metodo TITULACION.matricular (?alumno 2 ALUMNO)
        if not(member(?alumno, ?self.alumnos)))
        then anadir ?self.alumnos ?alumno
                ?self.numero-alumnos=(?self.numero-alumnos) + 1
```

9.7. Demonios

Son funciones que se ejecutan automáticamente, cuando le ocurre algo a un determinado atributo de una clase se disparan.

Se definen sobre ATRIBUTOS de las clases.

Además de definir qué es lo que hace, hay que definir que evento es el que lo dispara, pueden ser:

- **if-need**: antes de leer el valor del atributo.
- **if-set**: después de cambiar el valor.
- **if-added**: antes de añadir un valor a un multislot.

Se utilizan para:

- Requerir el valor de un atributo.
- Mantener la integridad de la base de conocimiento.
- Gestión dinámica de valores (actualizar valores a partir de otros)

Aquí no nos queda otra manera de referirnos a las instancias que apuntando a ellas, no podemos emplear los identificadores. El tipo instancia o lista de instancias se utilizará.

Ejemplo:

```
Demonio UNIVERSIDAD.numero-alumnos:if-needed ()
        numero=0
        For ?titulacion in ?self.titulaciones
                numero=numero+(enviar ?titulacion dame-numero-alumnos)
        self.numero-alumnos=numero
```

10. TEMA 8: PLANIFICACIÓN AUTOMÁTICA (PA)

10.1. Introducción

La idea es resolver problemas de manera automática. Tenemos un problema de planificación con determinadas características, si somos capaces de representar el problema con una determinada sintaxis podremos aplicar un planificar que lo resuelva, es fundamental poderlo representar en estos lenguajes.

Son problemas en los que estamos en un estado inicial, poseemos unos recursos y podemos realizar una serie de acciones para llegar al estado final deseado. La solución al problema será la secuencia de acciones que permiten llegar hasta este estado meta.

Entradas: Estado inicial s_0 (donde estamos), Metas G (que queremos conseguir) y Acciones $A : s \to s', \gamma(s, a) = s'$ (nos permite pasar de un estado a otro)

Solución: Un plan π, secuencia de acciones que nos lleva desde el estado inicial s_0 hasta una meta $s_n \subseteq G$.

10.1.1. Representación en Planificación

Es necesario para que un ordenador pueda resolver los problemas que estén descritos estos problemas en un lenguaje que este pueda entender. Hay muchas formas de expresar esta información.

Lógica de predicados es la más usada. Hay que representar estado y operadores. Los predicados en lógica permiten representar la verdad o falsedad de los hechos del mundo.

- Emplea: términos, predicados, conectivas y cuantificadores.

No existe una única representación válida, pero habrá diferentes niveles de abstracción del problema a los que se debe adaptar.

Lógica de Predicados

Elementos:

- **Términos:** constantes, variables y funciones.
- **Predicados**, los cuales representan: Relaciones, propiedades o igualdad.
- **Cuantificadores:** $\forall$ y $\exists$.
- **Conectiva:** $\vee$, $\wedge$, $\neg$, $\to$.

Inferencia: Modus Ponens

- SI P, P $\rightarrow$ Q ENTONCES Q

Unificación: encontrar los valores de las variables que hacen que la parte izquierda de la implicación sea igual a P(a).

- $\forall X(P(X) \rightarrow Q(X))$ tenemos P(a)
- $X = a$
- $P(a) \rightarrow Q(a)$ como tenemos P(a) entonces Q(a)

El proceso de planificación ira mirando los estados y en función de las acciones que pueda aplicar ira avanzando, este proceso es una búsqueda del estado meta y una vez se encuentra el camino ya tenemos la solución.

Esta clase de problema son PSPACE-Complete, requieren tanta memoria que no somos capaces de resolverlos, se verán algunos problemas de los que es posible encontrar la solución.

10.2. Representación

Simplificaciones: planificación clásica

- Se supone que los **estados son finitos**
- **Mundo aislado**, no hay factores externos que puedan variar nuestro estado, todo está controlado por nuestras acciones.
- **Determinista y acciones instantáneas**, siempre ocurren las acciones que se lanzan y se hacen al momento.

Necesitamos representar

- **Estados:** inicial, final y estados intermedios
- **Acciones:** cambian el estado

Formalización: lenguajes de representación que el ordenador entienda. La mayoría están basados en lógica de predicados

Estados y Acciones

Los estados se describen:

- Conjunto de **tipos de objetos**: persona, transporte, sitio, monumento.
- Conjunto de **objetos**: yo, bus1, castellana, Cibeles...
- Conjunto de **predicados genéricos**: at(?persona, ?lugar).
- Un **estado se representa por un conjunto de predicados instanciados**: at(yo, mi-casa), at(Bernabé, castellana).

Para representar una **acción** tenemos que saber **en qué estado** se puede ejecutar y **como cambia el estado y sus parámetros**.

- viajar-en-metro(?persona, ?linea, ?estacion1, ?estacion2)
- **Estado:** at(?persona, ?estacion1), link(?linea, ?estacion1, ?estacion2)
- **Cambio estado:** at(?persona, ?estacion2), at(?persona, ?estacion1)

Hipótesis del mundo cerrado

10.2.1. Representación en STRIPS

Los estados $s \in S$ se representan mediante variables proposicionales.

El **estado inicial** s_0 es I.

Los **estados metas** $s_G \in S$ son aquellos tales que $G \subseteq s_G$.

Las **acciones** A(s) son los $a \in A$ que cumplen $Pre(a) \subseteq s$.

El estado resultante al aplicar a en s es $s' = s - Del(a) + Add(a)$ (función de **transición de estados**).

El **coste** c(a, s) es siempre 1, por ahora siempre será mejor el plan más corto, pero se pueden asociar costes.

$P = (P, A, I, G)$

- **P:** conjunto proposiciones (variables lógicas).
- **A:** conjunto de acciones.
- **I:** estado inicial.
- **G:** metas.

Cada operador se representa por:

- Lista con los tipos de los objetos involucrados.
- **Pre(a)** $\subseteq$ **P**: precondiciones.
- **Add(a)** $\subseteq$ **P**: literales que pasan a ser verdaderos tras ejecutar a.
- **Del(a)** $\subseteq$ **P**: literales que pasan a ser falsos tras ejecutar a.

10.2.2. Definición Problemas Planificación

$\sqcap$ = (F, O, I, G)

- **F:** conjunto de fluents (cosa que es verdadera o falsa) (describen los estados, s $\subseteq$ F)
- **O:** conjunto de operadores
- **I** $\subseteq$ **F:** estado inicial
- **G** $\subseteq$ **F:** metas del problema

Un **plan** es una secuencia de acciones que convierte el estado inicial en un estado donde se alcanzan todas las metas

Heurísticas: conocimiento utilizado para elegir las acciones.

A partir del fichero **dominio D** y el de **problema P** se genera $\sqcap$.

10.3. Implementación

Fichero de dominio D (genérico), define el problema, como serán los estados y las acciones que se pueden aplicar.

- **Predicados genéricos** para definir los estados. (at ?person ?place) (link ?s1 ?s2)
- **Operadores genéricos**: Move
  - Parámetros: (?p ?from ?to)
  - Precondiciones: (at ?p ?from)
  - Efectos:
    - Añadir: (at ?p ?to)
    - Borrado: not(at ?p ?from)

Fichero de problema P (instanciado), fichero con todo los objetos, estado inicial y meta.

- **Objetos:** John, castellana, Cibeles...
- **Estados iniciales:** (at John castellana)...
- **Metas:** (at John Cibeles)...

Tendremos un Parser para instanciar estados a partir de D y P, se definirán los elementos de P con la forma que define en D.

Algoritmo de búsqueda.

10.3.1. Problemas y Dominios

El conjunto de posibles predicados (no instanciados) y objetos genéricos (types) será el mismo en ambos problemas.

El conjunto de posibles operadores (no instanciadas) será el mismo.

Pero las metas, el estado inicial y los objetos son distintos.

Podemos separar la información (lo que no depende del problema): lista de predicados, tipos de objetos y operadores: domain.

de la información específica del problema: objetos, estado inicial y metas: problem.

10.4. Lenguaje PDDL

Planning Domain Description Language

Creado en 1998 por Drew McDermott et al. para la primera International Planning Competition (IPC) http://ipc.icaps-conference.org/

Versiones:

- 1.0: 2000, IPC2 (:strips :typing :adl)
- 2.1: 2002, IPC3 (:fluents y :duration)
- 2.2: 2004, IPC4 (:derived-predicates y :timed-initial-literals, PPDDL)
- 3.0: 2006, IPC5 (:constraints y :preferences)
- 3.1: 2008, IPC6

Cada versión introduce alguna característica nueva.

Aplicación http://editor.planning.domains/

10.4.1. Características Generales

Sintaxis basada en Lisp.

Ficheros diferentes para definir el dominio y los problemas.

Diferentes niveles de expresividad, desde :strips hasta :preferences.

En los dominios se definen los requirements que un planificador tiene que tener para poder resolver problemas de ese dominio: :requirements flags

10.4.2. Estructura del Fichero de Dominio y Problema

```
(define (domain <nombreD>)
        (:requirements <:req1> . . . <:reqn>)
        (:types <subt1> . . . <subtn> - <tipo1>
                . . .
        (:constants <cons1> . . . <consn>)
        (:predicates <p1> <p2> . . . <pn>)
        (:action 1
                :parameters (...)
                :precondition (and ...)
                :effect (and ...)
        )
        . . .
        (:action n)
)

(define (problem <problem>)
        (:domain <nombreD>)
        (:objects
                <o11> . . . <o1m> - <subt1>
                . . .
                <o1n> . . . <onm> - <tipon>
        )
        (:init <l1> <l2> . . . <ln>)
        (:goal (and (<g1> <g2> . . . <gn>))
        (:metric minimize (. . . ))
)
```

10.4.3. Calidad Solución

1. Por defecto: longitud del plan.
2. Métrica especifica definida en el fichero de problema.

10.4.4. PDDL 1.0

Requirements:

- :strips. Básico
- :typing. Permite definir tipos de variables
- :negative-preconditions
- :disjunctive-preconditions
- :equality. Permite la igualdad en los predicados
- :existential-preconditions. Permite usar exists
- :universal-preconditions. Permite usar forall
- :quantified-preconditions. Permite usar exists y forall.
- :conditional-effects. Permite usar when
- :adl. Engloba :strips, :typing, :negative-preconditions, :disjunctive-preconditions, :equality, :quantified-preconditions y :conditional-effects

Operadores:

- Operadores en las precondiciones:
  - and / or / not
  - imply <cond><effect>
  - exists <variable><literal>
  - forall <variable><literal>
- Operadores en los efectos:
  - and / not
  - forall <variable><effect>
  - when <expression><effect>

Por defecto tratará de minimizar el número de acciones.

10.4.5. PDDL 2.1

Creado en 2002 por Maria Fox y Derek Long para la IPC3.

Características:

- Variables numéricas.
- Métricas de calidad de los planes (metrics)
- Acciones durativas (durative actions)

Variables numéricas

Hay que añadir :fluents en los :requirements.

Se añaden con un nuevo campo :functions en la definición del dominio.

- (:functions (fuel-level ?r - rover)(fuel-used ?r - rover))(fuel-needed ?w1 ?w2 - waypoint)(total-fuel-used))

Significado: es cierto que el valor de la función en el estado es x.

Se pueden usar en las precondiciones y efectos de las acciones.

- Precondiciones, con operaciones relacionales: =, >, <, >=, <=.
- Efectos, para cambiar su valor: increse, decrease, assign, scale-up, scale-down.

El valor inicial se tiene que dar en el fichero de problema.

- (:init (= (fuel-level ?truck) 100))

Métricas de Calidad

- Se añade un nuevo campo en el fichero de problema, :metric.
- Se puede maximizar o minimizar.
- (:metric minimize (total-fuel))
- (:metric minimize (+ (* 2 (fuel-used car))(fuel-used truck)))

Acciones durativas

Cada acción puede tener una duración diferente: requirement :duration

Cambia la definición de las acciones:

```
(:durative-action <nombre>
        :parameters (?a - <tipo>)
        :duration (<duration>)
        :condition (and ...)
        :effect (and ...)
)
```

Formas de definir la duración:

- **valor numérico:** :duration (= ?duration 5)
- **valor calculado:** :duration (= ?duration (+ ?x ?y))
- **intervalo:** :duration (and (>?duration 0) (<?duration 10))
  (necesita requirement :duration-inequalities)
- **sin valor** (dura hasta que una precondición se haga falsa)

Tienen que estar temporalmente cualificados:

- at start
- at end
- over all

10.4.6. PDDL 2.2

Creado por Stefan Edelkamp, Jorg Hoffman y Michael Littman en 2004 para la IPC4.

3 niveles de PDDL 2.1 mas

- **Derived predicates** (axiomas de PDDL 1)
  Requirement: :derived-predicates
- **Timed initial literals**
  Requierement: :timed-initial-literals
  Literales que se hacen ciertas o falsas en un determinado instante.
  Se definen en el fichero de problema, en el init.

10.4.7. PDDL 3.0

Desarrollado por Alfonso Gerevini y Derek Long para la IPC5

Introduce:

- **State trajectory constraints:**

  Necesita requirement: :constraints

  Metas intermedias

- **Goal preferences:**

  Necesita requirement: :preferences

  Metas que no son obligatorias (pero si deseables)

State Trajectory Constraints

Se definen en la sección :constraints del problema o del dominio (válido para todos los problemas)

Se representan utilizando logica temporal con los siguientes operadores modales:

- always ϕ: ϕ tiene que ser cierto en todos los estados
- sometime ϕ: ϕ tiene que ser cierto en algún momento
- at-most-once ϕ: ϕ tiene que ser cierto una sola vez
- at end ϕ: ϕ tiene que ser cierto al final
- within t ϕ: ϕ tiene que ser cierto durante t instantes de tiempo
- sometime-before ϕ ϕ_2: ϕ tiene que ser cierto antes que ϕ_2
- sometime-after ϕ ϕ_2: ϕ tiene que ser cierto después que ϕ_2
- always-within t ϕ ϕ_2: cuando ϕ sea cierto, ϕ_2 debe ser cierto antes de t instantes de tiempo
- hold-during t_1 t_2 ϕ: ϕ tiene que ser cierto en el intervalo de tiempo entre t_1 y t_2
- hold-after t ϕ: ϕ tiene que ser cierto después de t

Preferencias

Se pueden definir en:

- Fichero de dominio: (aplicable a todos los problemas)
  En la parte de :constraints
  En las precondiciones de los operadores
- Fichero de problema
  En la parte de :constraints
  En la parte de :goals

Sintaxis: (preference <nombre>ϕ)

- <nombre>es opcional
- Varias preferencias pueden tener el mismo nombre

Las preferencias pueden tener un peso para establecer prioridades

Solo aplicables a las que tienen nombre

Por defecto se les asigna peso=1

Se definen en la parte de métricas del problema

10.5. Algoritmos de Planificación

10.5.1. Estrategias de Búsqueda

Dirección:

- Encadenamiento hacia delante (forward chaining)
  Desde el estado inicial
  Aplicar operadores cuyas precondiciones son ciertas en el estado
  Obtener un nuevo estado
- Encadenamiento hacia detrás (backward-chaining)
  Desde las metas del problema
  Encontrar un operador cuyos efectos añadan la meta
  Añadir las precondiciones de dicho operador al conjunto de metas

Características deseables: Sound, completo, optimalidad y escalabilidad

10.5.2. Modelo de Planificación STRIPS

Suposiciones:

- Acciones
  - Deterministas
  - Instantáneas
  - Recursos discretos
  - Coste unitario
- Entorno
  - Observabilidad total
  - Estático
- Objetivo
  - Satisfacción total
  - Métrica longitud del plan

Los costes son 1, unitarios.

Conocemos todo, observabilidad total, todo lo que ocurre es conocido y podremos actuar.

Buscamos el plan más corto.

10.5.3. Modelo de Planificación con Costes

Suposiciones:

- Acciones
  - Deterministas
  - Instantáneas
  - Recursos discretos
  - **Coste no unitario**
- Entorno
  - Observabilidad total
  - Estático

- Objetivo
    - Satisfacción total
    - Métrica **coste del plan**

Se dejan de tener costes unitarios, de manera que cada acción tendrá un coste diferente.

Buscaremos el plan de menor costes, no el más corto

10.5.4. Modelo de Planificación Temporal

Suposiciones:

- Acciones
    - Deterministas
    - **Durativas y concurrentes**
    - Recursos discretos
    - Coste unitario
- Entorno
    - Observabilidad total
    - Estático
- Objetivo
    - Satisfacción total
    - Métrica **duración del plan**

Las instancias ya no son ciertas siempre, sino que serán durativas y concurrentes dependerán del tiempo.

Buscaremos el pan de menor duración.

10.5.5. Modelo de Planificación Con Recursos

Suposiciones:

- Acciones
    - Deterministas
    - Instantáneas
    - **Recursos continuos**

        - Coste unitario
    - Entorno
        - Observabilidad total
        - Estático
    - Objetivo
        - Satisfacción total
        - Métrica longitud del plan

Los recursos son continuos, no continuos.

10.5.6. Modelo de Planificación con Incertidumbre

Suposiciones:

- Acciones
    - Deterministas / **No deterministas**
    - instantáneas
    - Recursos discretos
    - Coste unitario
- Entorno
    - Observabilidad total / **Observabilidad parcial / Observabilidad nula** (P. probabilidad / P. contingente / P. conformante)
    - Estático
- Objetivo
    - Satisfacción total
    - Métrica longitud del plan

Las acciones ya no tienen lugar siempre, sin que tiene una probabilidad de fallar o no tener lugar.

El entorno ya no es conocido con totalidad, sino que será parcial o nulo el conocimiento. Tendremos que hacer suposiciones y tratar las posibilidades.

10.5.7. Clasificación

Planificación determinista

- Conocimiento completo del dominio (estado y acciones)
- No hay retroalimentación (feedback) al ejecutar planes

Planificación probabilística:

- Estado completamente observable
- Acciones no deterministas: su ejecución puede producir distintos estados siguiendo una distribución de probabilidad
- Necesaria retroalimentación

Planificación contingente (contingent planning): planes condicionales con ramas de actuación alternativas para contingencias

Planificación conformante (conformant planning): planes condicionales tanto para acciones como para estados