Sistemas Inteligentes - Appunti

Francesco Lorenzoni

Febrero 2025

Contents

Part I

Introduction to SIN

Chapter 1

Introduction

1.1 Informaciones generales

En los Viernes las clases son online y sincronas.

La asignatura se divide en does partes:

1. Parte 1
   - T1 Introducción
   - T2 Planificación inteligente
     Algunos ejemplos son la planificación de procesos/planes de producción, o la planificación de rutas de reparto y logística; en general, aplicaciones incluyen sistemas de ayuda a la toma de decisiones y recomendación.
   - T3 Optimización inteligente Optimización de la logistica empresarial (horarios, recursos, personal, turnos de trabajo,...)
   - T4 Agentes inteligentes Sistemas multiagentes que tienen que negociar, coordinarse y cooperar para un objetivo común. Pero algo videojuegos donde existe una competición o cooperación entre agentes y se requiere una búsqueda de alternativas.
2. Parte 2
   - T5 Modelado de lenguaje y autómatas finitos
   - T6 Aprendizaje profundo
   - T7 Aprendizaje profundo para clasificación de texto
   - T8 Aprendizaje profundo para clasificación de imágenes

Profesor	T1	T2	T3	T4	T5	T6	T7	T8
Jaume Jordán								
Carlos Carrascosa								
Carlos Martínez (resp.)								

Figure 1.1: Profesores

La evaluación se efectuará mediante:

- Dos tests (**20 de junio a las 15:30**) sobre contenidos teóricos y prácticos de la parte 1 (15%) y parte 2 (15%), totalizando 30%.
- Trabajos académicos (70%):
  - Parte 1 (35%): tema 2 (15%) y tema 4 (20%).
  - Parte 2 (35%): tema 5 (10%), tema 7 (10%) y tema 8 (15%).

1.2 Inteligencia Artificial

IA no es solo IA generativa. IA es un campo muy amplio que incluye otras cosas. Inicialmente se asociaba con el Machine Learning, pero ahora se ha ampliado a otras cosas.

La Inteligencia Artificial es un campo interdisciplinario que implica máquinas capaces de imitar determinadas funcionalidades de la inteligencia humana, incluidas características como la percepción, el aprendizaje, el razonamiento, la resolución de problemas, la interacción lingüística e incluso la producción de trabajos creativos.

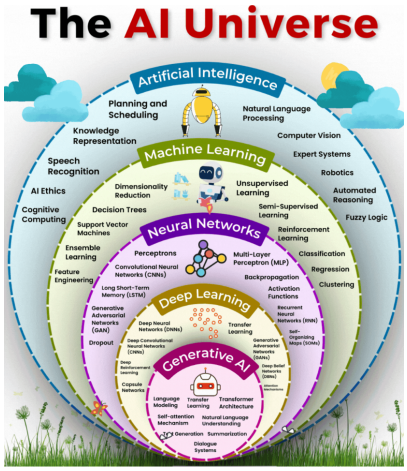

Figure 1.2: AI esquema. Según el profesor esto esquema no es completo, pero es cilustrativo.

Modelos de IA:

- ◇ Aprendizaje supervisado
- ◇ Aprendizaje no supervisado
- ◇ Aprendizaje por refuerzo

1.2.1 Aplicaciones de la IA

La Inteligencia artificial puede ser util para estas cosas: Áreas y aplicaciones

- ◇ Recuperación inteligente de la información
  - – Recuperación de información no explícitamente representada
  - – Bases de datos deductivas. Procesos inferenciales. Sentido común. Interfaz natural
- ◇ Ingeniería del conocimiento (SBC y Sistemas Expertos)
  - – Problemas NP: restricción de dominios, heurísticas y meta-heurísticas
  - – Planificación. Scheduling. Optimización. Sistemas de Ayuda a la Toma de Decisiones
- ◇ Problemas (de búsqueda) combinatorios y de planificación
  Este es el problema más importante de la IA.
  - – Problemas NP: restricción de dominios, heurísticas y meta-heurísticas
  - – Planificación. Scheduling. Optimización. Sistemas de Ayuda a la Toma de Decisiones

  Hay también otras aplicaciones
- ◇ Sistemas Multiagente y distribuidos que interactúan (cooperación)
- ◇ Robótica
- ◇ Percepción
- ◇ Procesamiento de lenguaje natural

Estado inicial

Información estática	Información dinámica
distance[home][airport_city1]=15 distance[home][center_city1]=5 distance[home][connection_c1_c2]=2 ..... vehicles(taxi)={taxi1} vehicles(plane)={plane1} ... location(airport)={airport_city1, airport_city2} location(to_stay)={home, hotel_city2, ...} .... city(home)=city1 city(airport_city1)=city1 city(airport_city2)=city2	at(me)=home at(taxi1)=center_city1 at(plane1)=airport_city1 at(car1)=home cash(me)=20 ;; dinero que tengo owe(me)=0 ;; dinero que debo **Objetivos** at(me)=hotel_city2

Figure 2.2: Suponen que tengo que viajar desde una ciudad a otra. Esta puede ser una formalización de la planificación.

Chapter 2

Planificación Inteligente

Podemos ver la Planificación Inteligente desde varios puntos de vista:

- ◇ Deducción
  A partir de algunos hechos ciertos, concer qué otros hechos también son ciertos (propagación)
- ◇ Aprendizaje
  "Mejorar la conudcta a partir de la experiencia"
- ◇ Razonamiento sobre acciones A partir de lagunos hechos ciertos (estado inicial), decidir qué acciones se deben ejecutar para alcanzar ciertos objectivos (es decir, que otros hechos sean también ciertos) en base a relaciones de causa-efecto.

El ser humano es inteligente porque es capaz de razonar acerca de las consecuencias de sus actos. Al fin y al cabo estamos planificando constantemente aunque no nos demos cuenta.

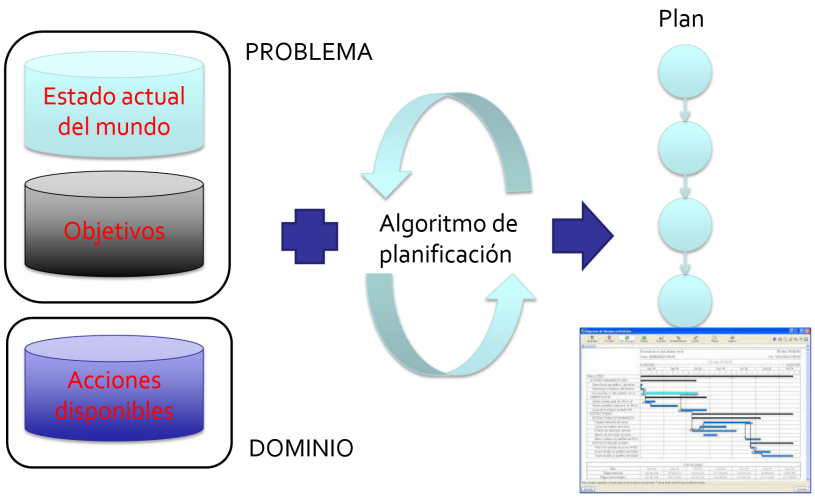

Figure 2.1: Planificación esquematizada

En Fig. 2.1 podemos ver un esquema de planificación. En el estado actual del mundo corresponde al *"**dominio**"*, y el objetivo es llegar a un estado deseado nel mundo.

Definition 2.1 (Plan) *Un plan* `P` *es una secuencia de acciones que transforma el estado inicial (*`I`*) en un estado que satisface los objetivos (*`G`*)*

Un plan puede estar parcial o totalmente ordenado. Un enlace causal (a_i, p, a_j) determina que la acción ai resuelve la condición p de la acción a.

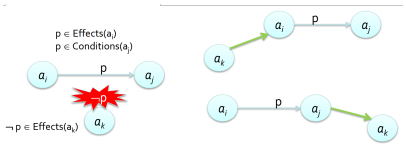

Figure 2.4: Plan

2.0.0.1 Modelo "planificación clásica"

- ◇ Determinista
- ◇ Estático (solo cambia por las acciones)
- ◇ Completamente observable
- ◇ Razonamiento temporal y numérico no contemplado

Es decir, el resultado de una acción aplicada a un estado completamente conocido se puede predecir correctamente, ya que no hay influencias externas que afecten al entorno.Para resolver este problema se utilizan algoritmos de búsqueda.

Esto modelo pero es muy limitado, y sus asuciones necesitan ser relajadas para que sea más realista y aplicable a problemas reales.
En primer lugar, las acciones tienen distintas **duraciones**, usan recursos y tienen **coste**; además, El mundo *no* es totalmente conocido (existe incertidumbre) y es **dinámico** (cambia debido a eventos exógenos).

2.1 Planificación Jerarquica

La idea principal de **HTN** Hierarchical Task Network planning es proceder de una forma más parecida a como los humanos resolvemos los problemas complejos: Primero los aspectos generales (tareas) y después los detalles.

Se definen métodos abstractos que se deben descomponer para resolver el problema. Métodos son los triángulos invertidos en la figura.

Cada método incluye estos componenetes:

- ◇ Nombre
- ◇ Párametros
- ◇ Precondiciones
- ◇ Red de tareas

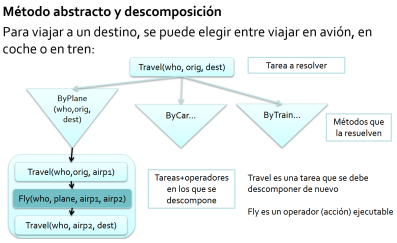

Figure 2.5: Método abstracto y decomposición

Métodos son tareas a descomponer, y los operadores (como `Fly`) representan las acciones que se pueden llevar a cabo dentro del mundo para poder alcanzar los distintos objetivos (acciones ejecutables que producen cambios en el estado del mundo).

- ◇ Nombre
- ◇ Parámetros
- ◇ Condiciones
- ◇ Efectos

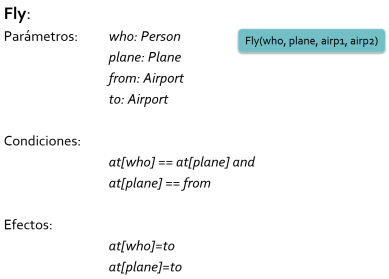

Figure 2.6: Accion `Fly`

Chapter 3

Pyhop

Para resolver un problema tenemos tres fases

1. Modelato
2. Planificación
3. Ejecución

El problema se define como una declaración de un nuevo estado del mundo (inicial), y una declaración de un objetivo. El **dominio**, son las acciones disponibles, que se estructuran en dos niveles: tareas/operadores y métodos.

- ◇ Información estática: no cambia durante la ejecución del problema y que sirve de soporte. Por ejemplo: distancias entre ciudades.
- ◇ Información dinámica: representa el estado actual del mundo. Por ejemplo: cantidad de dinero disponible por el agente en un momento determinado

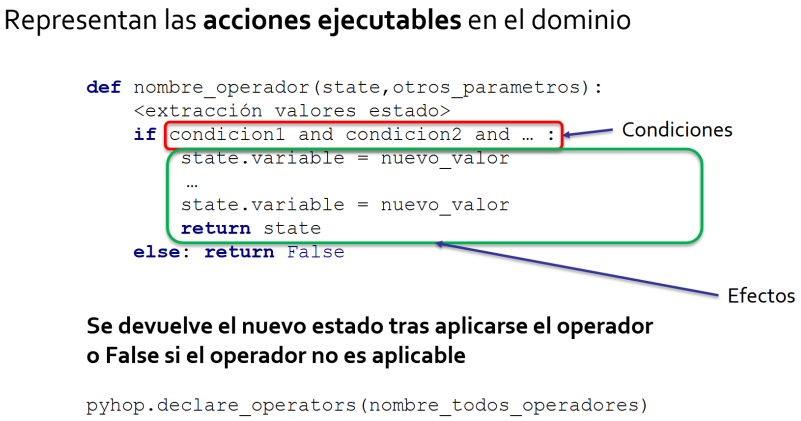

Figure 3.1: Como definir un operador en Pyhop

Chapter 4

Optimización Inteligente

Planificación y Scheduling representa un área de gran relevancia en Inteligencia Artificial. Muchos problemas reales se modelan como problemas de P&S.

La planificación es un proceso de deliberación que escoge y organiza acciones anticipando sus resultados o consecuencias.

Lo scheduling es un proceso de asignación de recursos a tareas sobre el tiempo para optimizar uno o más objetivos.

PLANNING	SCHEDULING
La tarea de planificación determina **QUÉ** acciones son necesarias llevar a cabo para alcanzar un estado objetivo	La tarea de scheduling se centra en **CUÁNDO/CÓMO** llevar a cabo las acciones para optimizar el problema

Figure 4.1: P&S

4.1 Scheduling

Un problema de scheduling consiste en organizar en el tiempo un conjunto de actividades que compiten por el uso limitado de recursos

4.1.1 JSP

JSP stands for Job-Shop Scheduling, y es un paradigma de los problemas de scheduling es muy simple de enunciar y muy difícil de resolver.

- n trabajos cada uno compuesto por un conjunto ordenado de tareas
- m maquinas donde se procesan las tareas
- Cada tarea debe ser procesada en una única máquina durante un tiempo determinado y en un orden prefijado;
- El objetivo es minimizar *makespan*: instante de finalización de la última tarea

- **Datos** - p_{ij} es el tiempo de proceso de la tarea i en el trabajo j;
- **Variables** - st_{ij} es el tiempo de inicio de tarea i en el trabajo j;
- **Restricciones** -
  - Secuential - $st_{ij} + p_{ij} \leq st_{i(j+1)}$
  - Capacidad - $st_{ij} + p_{ij} \leq st_{kl} \bigvee st_{kl} + pkl \leq st_{ij}$
  - Sin interrupción - $Cij = st_{ij} + p_{ij}$
- **Objetivo** - Construir una ordenación de las tareas en el tiempo de manera que se satisfagan todas las restricciones sobre cada máquina.
  - Mimimize the makespan: $C_{max} = maxC_{ij}$
  - Minimize total flow time: $C_{sum} = \sum Ci$
  - Minimize makespan + energy

4.1.2 CSP - Constraint Satisfaction Problem

Muchos problemas pueden ser expresados mediante:

- Un conjunto de variables.

- ◇ Un dominio de interpretación (valores) para las variables.
- ◇ Un conjunto de restricciones entre las variables.
- ◇ La solución al problema es una asignación válida de valores a las variables.

Formalization omitted here

En general, los problemas de optimización son particularmente difíciles de resolver. En general, basta con una solución razonablemente buena (factible, optimizada, pero no la óptima) en un tiempo razonable, y si puede obtener con métodos aproximados (como alternativa a los métodos exactos), cómo la **Búsqueda Heurística** / Metaheurística: La solución es obtenida (generada / mejorada), mediante un proceso de búsqueda en un amplio espacio de estados/-soluciones.

- ◇ Búsqueda Local (Local Search) - Se parte de una solución inicial y se busca una solución mejor en el vecindario de la solución actual.
- ◇ Búsqueda Sistemática (Systematic Search) - Se parte de una solución inicial y se busca una solución mejor en todo el espacio de soluciones.

- ◇ Métodos Constructivos - Construyen una solución paso a paso, añadiendo o eliminando elementos de la solución.
- ◇ Métodos de Mejora - Parten de una solución inicial y la mejoran iterativamente mediante movimientos locales.
- ◇ Métodos Evolutivos - Combinan soluciones previas

4.1.3 Algoritmo Genéticos

Los **algoritmos genéticos** son un tipo de algoritmo evolutivo que se utiliza para encontrar soluciones aproximadas a problemas de optimización y búsqueda. Estos algoritmos reflejan el proceso de selección natural en la evolución de las especies, y se basan en la idea de que las soluciones a un problema pueden ser representadas como individuos en una población, y que la evolución de la población puede conducir a la generación de soluciones cada vez mejores.

La operación fundamental es la **cruce**, que toma n padres (típicamente 2) y generar m hijos (típicamente 2); consiste en heredar material genético de los padres. La asunción es que si los padres tienen buen material genético, la mezcla del material genético de ambos también será buena.
Entonces, el efecto de la cruza es *explorar* el espacio de búsqueda con **material prometedor**.

La operación de **mutación** es una operación que se aplica a un individuo y cambia aleatoriamente uno o más genes de ese individuo. Corisponte a una búsqueda local, porque sólo se cambia un gen a la vez, y entonces el hijo serà cerca de su padre.

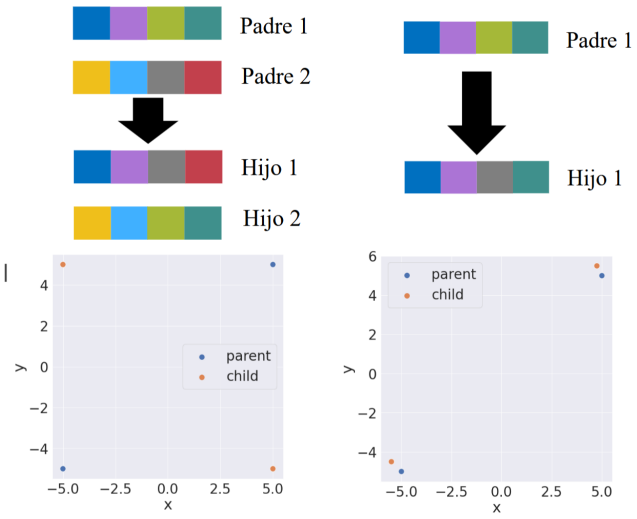

Figure 4.2: Cruce y mutación

TODO selección des padres TODO criterio de parada

4.1.3.1 Resumen

1. Una **representación** adecuada de las *soluciones* del problema (individuos).
   Cromosomas, genes, genotipo, fenotipo.
2. Una forma de crear una ***población*** *de soluciones inicial* (individuos iniciales). A menudo, aleatoria.
3. Una **función de evaluación** capaz de medir la adecuación de cualquier solución (individuo) al problema. Fitness.
4. Un conjunto de **operadores evolutivos** para combinar las soluciones existentes con el objetivo de obtener nuevas soluciones (nuevos individuos): *selección*, *cruce*, *mutación* y *reemplazo* de individuos. Guían el proceso de la búsqueda.
5. Conjunto de **parámetros de entrada**: tamaño de la población, número de iteraciones (generaciones), probabilidades de selección, etc.

Podemos resumir el proceso de un algoritmo genético en los siguientes pasos:

1. **Codificación** de Soluciones/instanciaciones.
2. **Población Inicial**:

   i. Generar población aleatoria de n individuos: x (posibles soluciones del problema)
   ii. Evaluar la aptitud o fitness f(x) de cada individuo.
3. **Ciclo-Generacional**:
   i. **Selección** Seleccionar dos padres de la población de acuerdo a su aptitud: Probabilidad de cruce (pc).
   ii. **Cruce** Combinar los genes de los dos padres para obtener descendientes.
   iii. **Mutación** Con una probabilidad de mutación (pmut), mutar cada gen de los cromosomas hijo.
   iv. **Reemplazo** Añadir nuevos hijos y determinar nueva población.
4. Verificar condición de parada:
   i. **Parada**: proporcionar como solución el individuo con mejor valor de aptitud f(x).
   ii. **Ciclo**: Ir al paso 2

Chapter 5

Agente Inteligentes

Agentes autónomos trabajan para resolver problemas en un entorno dinámico y complejo. Un agente es un sistema computacional que opera en un entorno y es capaz de percibirlo y actuar sobre él. Un agente inteligente es aquel que actúa de manera racional, es decir, que actúa para alcanzar sus objetivos, basándose en la información que percibe y en su conocimiento del entorno. Más formalmente,

Definition 5.1 (Wooldrdige) *Un agente es un sistema informático capaz de actuar autónomamente en algún entorno con el fin de alcanzar los objetivos que se le han delegado.*

Definition 5.2 (Wooldridge updated) *Cualquier proceso computacional dirigido por el objetivo capaz de interaccionar con su entorno de forma* ***flexible*** *(Reactivo, Proactivo, Social) y* ***robusta***

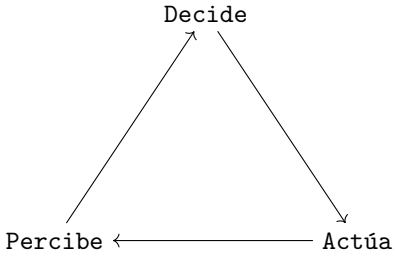

Figure 5.1: Agente bucle

Un agente reactivo reacciona a los cambios del entorno. No tiene que ser el más rápido, pero la reacción debe ser útil. Un agente proactivo actúa para alcanzar sus objetivos, planifica y actúa en consecuencia. Un agente social interactúa con otros agentes.

En entornos complejos, un agente no tiene control completo sobre su entorno, sólo tiene un control parcial. Control parcial significa que el agente puede influir sobre el entorno con sus acciones. Una acción ejecutada por un agente puede fallar o tener el efecto deseado. En conclusión, los entornos son no deterministas y los agentes deben estar preparados para posibles fallos.

- ◇ Accesible vs inaccesible.
  Un entorno accesible es aquel en el que el agente puede obtener información completa, exacta y actualizada del estado del entorno.
- ◇ Determinista vs no determinista. Un entorno determinista es aquel en el que cualquier acción tiene un único efecto garantizado, no hay incertidumbre sobre el estado resultante de la ejecución de una acción
  El mundo físico puede a todos los efectos ser considerado como no determinista.
  Los entornos no deterministas presentan grandes problemas para el diseñador de agentes.
- ◇ Episódico vs no episódico
  En un entorno episódico el desempeño/actuación de un agente depende de un número discreto de episodios, no existiendo enlaces (relación) entre el desempeño de un agente en escenarios distintos. Los entornos episódicos son, desde el punto de vista del desarrollador de agentes, más sencillos porque el agente puede decidir que acción ejecutar basándose únicamente en el episodio actual, no necesita razonar sobre las interacciones entre el episodio actual y los episodios futuros.
- ◇ Estático vs dinámico
  Un entorno estático es aquel en el que se puede asumir que no se producen cambios excepto los provocados por la ejecución de acciones del agente. Un entorno dinámico es aquel que tiene otros procesos que operan en el, y que por lo tanto se producen cambios que están fuera del control del agente.

◇ Discreto vs continuo
Un entorno es discreto si en él hay un número fijo y finito de acciones y percepciones. El juego del ajedrez es un ejemplo de entorno discreto, y la conducción de un taxi un ejemplo de entorno continuo (AIMA, Russell and Norvig).

Asumimos que el entorno puede estar en uno cualquiera de los estados de un conjunto finito de estados instantáneos discreto (E):

$$E = s_1, s_2, ... \tag{5.1}$$

• Se asume que los agentes tienen disponible un repertorio (conjunto finito) de posibles acciones que transforman el estado del entorno :

$$Ac = \alpha_1, \alpha_2, ... \tag{5.2}$$

$$Acción : E \to A \tag{5.3}$$
$$Percibir : E \to P \tag{5.4}$$
$$Decisión : P^* \to A \tag{5.5}$$
$$Selección : I \to Ac \tag{5.6}$$
$$actualizar\_estados : I \times P \to I \tag{5.7}$$

◇ P es un conjunto (no vacío) de percepciones (entradadas perceptivas), que relaciona estados del entorno con percepciones
◇ I es el conjunto de todos los estados internos del agente

5.1 Jason

Jason es un lenguaje de programación basado en agentes, que permite la programación de agentes inteligentes. Jason es un lenguaje de alto nivel, basado en la lógica, que permite la programación de agentes inteligentes, y es una extension de AgentSpeak.

Los elementos fundamentales del Lenguaje son:
◇ **Creencias** (Beliefs)
◇ **Objetivos** (Goals)
◇ **Planes** (Plans, Intentions)

5.1.1 Creencias

Cada agente tiene una base de **creencias**, colección de literales representados como *predicados*.
◇ `tall(jhon).`
◇ `likes(jhon, music)`

Anotaciones son detalles asociados a una creencia.
`busy(jhon)[expires(autum)]` Las anotaciones aportan "elegancia" al lenguaje y facilitan el manejo de la base de creencias.
Existen anotaciones que tienen un signicado especial para el interprete. En particular la anotacion `source`.

Hay tres tipos de fuentes de informacion para los agentes

◇ *Informacion perceptual*: aquella que percibe del entorno
  – `source(percept)`
  – `(colour(box1,blue)[source(percept)])`
◇ *Comunicacion*: aquella que proviene de otro agente del sistema
  – `source(id agente)`
  – `(colour(box1,blue)[source(bob)])`
◇ *Notas mentales*: creencias que provienen del propio agente
  – `source(self)`
  – `(colour(box1,blue)[source(Self)])`

5.1.2 Objetivos

Existen dos tipos de objetivos en Jason:
1. **Achievement goals**(operador `!`): Expresan un estado del mundo que el agente desea conseguir.
   ◇ `own(house)`
2. **Test goals**(operador `?`): Usados normalmente para recuperar información de la base de creencias.
   ◇ `bank_balance(BB)`

5.1.3 Planes

Un plan tiene tres partes

- ◇ **Triggering event**: Cambios en las creencias o objetivos del agente
- ◇ **Context**: Determinan si un plan es aplicable. Literales que deben ser una consecuencia lógica de la base de creencias para que el plan se instancie.
- ◇ **Body**: Sucesión de acciones que comporta el plan. Es una secuencia de acciones. Pueden existir nuevos subobjetivos.

Figure 5.2: Plan schema

5.1.3.1 Cuerpo de un plan

El cuerpo del plan es secuencia de instrucciones separadas por ";". Estas instrucciones puedes ser:

- ◇ Actions: Acciones externas que se denotan por un predicado. Proporcionan un "feedback".

```
rotate(leftarm, 45)
```

- ◇ Achievement goals: subobjetivos que deben ser alcanzados para que el plan continúe su ejecución (si en lugar de emplear "!" se emplea "!!', el plan no suspenderá su ejecución).

```
!!at(home); call(john) en lugar de !at(home); call(john).
```

- ◇ Test goals: Para recuperar información de la BB o verificar si el agente cree algo.

```
?coords(Tarjet,X,Y).
```

- ◇ **Mental Notes**: Anotación source(self). Sirven para añadir, modificar o eliminar nuevas creencias.

```
+currenttargets(NumTargets); [Se anade]
-+currenttargets(NumTargets); [Se modifica]
```

- ◇ **InternalActions**: Son acciones que no modifican el entorno. Se diferencian de acciones del entorno por el carácter ".".

```
.print(...); .send(...);
```

- ◇ **Expressions**: Su sintaxis es semejante a la de Prolog.

```
X >=Y*2
```

- ◇ **Plan Labels**: Los planes pueden estar etiquetados

```
@labelte: ctxt<-body.
```

5.1.3.2 Fallo de un Plan

- ◇ Un plan puede fallar por tres causas principales:
  - – Falta de planes relevantes o aplicables para un "achievement goal"
  - – Fallo de un "test goal"
  - – Fallo de una acción
- ◇ Cuando un plan falla se genera un evento `-!g` (goal deletion event) si se generó por la adición de un "achievement goal" o "test goal".
- ◇ El plan que se dispare por el fallo se añade a la pila de intenciones del plan que ha fallado.

5.1.4 Ejemplos

```
/* Creencias iniciales */
inicio.
/* Planes */
+inicio <- .print("Hola Mundo"). //plan que se dispara al inicio
```

```
/* Creencias iniciales */
fact(0,1).
/* Planes */
+fact(X,Y) : X < 5
<- +fact(X+1, (X+1) * Y ).
+fact(X,Y) : X = 5
<- .print("fact 5 == ", Y ).
```

```
/* Objetivos iniciales */
!dots.
!control.
/* Planes */
+!dots
<- .print("."); // imprime puntos en un bucle sin fin
!!dots.
+!control
<- .wait(30); // este plan es un bucle que activa y desactiva el otro plan
.suspend(dots); // suspende la intencion asociada al plan dots
.println;
.wait(200);
.resume(dots); // reactiva la intencion asociada al plan dots
!!control
```

```
// Un agente que soluciona el problema del mundo de bloques
/* Creencias iniciales y reglas */
clear(table).
clear(X) :- not(on(_,X)). // la creencia clear(X) es cierta cuando no tiene nada encima
tower([X]) :- on(X,table). // la torre X es cierta cuando X esta sobre la mesa
// X puede ser un bloque o varios
tower([X,Y|T]) :- on(X,Y) & tower([Y|T]).
// la torre va creciendo si un bloque X se pone sobre el bloque Y
/* Objetivos iniciales */
// Marca el estado final a alcanzar
!state([[a,e,b],[f,d,c],[g]]).

/* Planes */
// Alcanzar una torre
+!state([]) <- .print("Finalizado!").
+!state([H|T]) <- !tower(H); !state(T).
// Alcanzar un estado donde la torre esta construida
+!tower(T) : tower(T). // La torre deseada ya existe, no hay nada que hacer
+!tower([T]) <- !on(T,table). // La torre es de un solo elemento
+!tower([X,Y|T]) <- !tower([Y|T]); !on(X,Y). // divido el problema en subproblemas
```