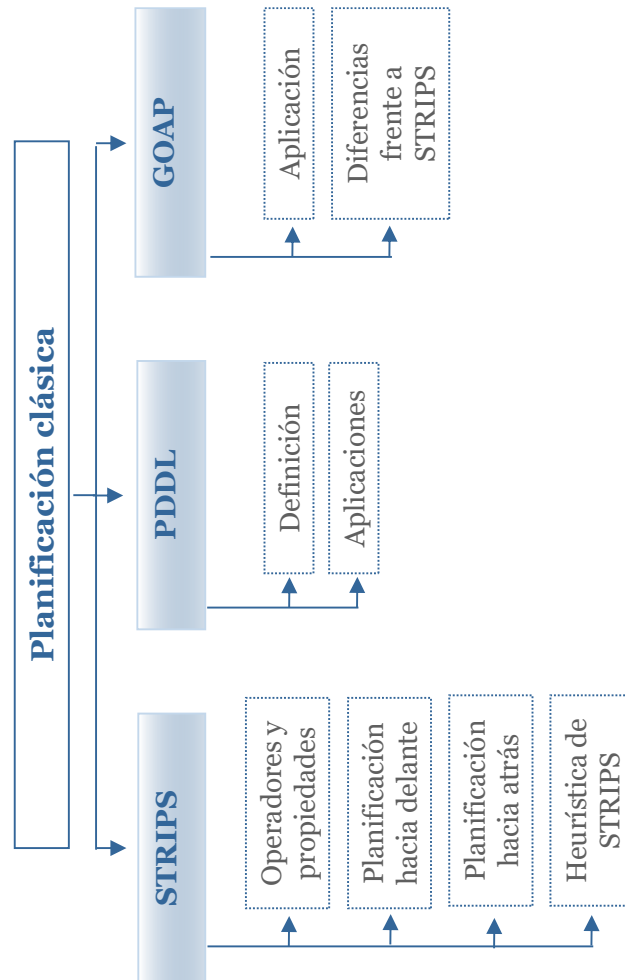


Razonamiento y Planificación Automática

Sistemas basados en STRIP

Índice

Esquema	3
Ideas clave	4
9.1. ¿Cómo estudiar este tema?	4
9.2. STRIPS	4
9.3. PDDL	14
9.4. GOAP	17
9.5. Referencias bibliográficas	23
Lo + recomendado	24
+ Información	25
Test	28



9.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** que encontrarás a continuación.

En este tema trataremos de definir los planificadores tradicionales basados en la descripción de STRIPS (Fikes y Nilsson, 1971) y su evolución en el lenguaje de definición de planes PDDL (McDermott, Ghallab, Howe, Knoblock, Ram y Veloso, 1998).

Por último, presentaremos una adaptación de los sistemas STRIPS pensada para sistemas con reacción en tiempo real, para poder crear sistemas que puedan reaccionar a entornos cambiantes de modo eficiente. El sistema GOAP, ideado por Jeff Orkin, adapta los estándares de STRIPS para estos entornos.

9.2. STRIPS

Definición y Base

Partiremos de una información que conoce *a priori* un agente y tenemos una representación de los conocimientos basada en la idea de los mecanismos de búsqueda. Por ello contaremos con:

- ▶ Un estado inicial: símbolo (s_0).
- ▶ Un conjunto de estados sucesores producidos por la función expandir ($s_7 \mapsto \{s_8, s_{10}, s_{12}, s_{27}, s_{112}\}$).

- Un conjunto de estados meta, en los que la función meta? evalúa como finales o no ($s_{112} \mapsto \text{true}$).
- Una función de coste asociado a la aplicación de un operador: $c(s_7, s_{112}) \mapsto 5$.
- Una o varias funciones heurísticas: $h^*(s_7 \mapsto 235)$.

Con esta información tenemos una abstracción completa de las características de los estados, pero en esta abstracción tenemos una pérdida de información acerca de la conformación de los estados (por ejemplo: «¿bloque C encima de bloque A?», «¿el agente se encuentra en Praga?»...

Esto nos genera varios problemas potenciales que derivan en un aumento de la complejidad para poder representar la información y el conocimiento (por ejemplo, para definir la función expandir), y hace más difícil el mantenimiento del conocimiento. Es por esto que buscamos otros mecanismos para poder trabajar con la información en estos agentes inteligentes. Para ello, tendremos en cuenta que los estados presentan una «estructura» y definiremos un estado no por su nombre, sino por el conjunto de sus **propiedades**.

Elementos básicos de STRIPS

A principios de los años 70, se creó el lenguaje estandarizado para formalizar un problema de planificación (Fikes y Nilsson, 1971). STRIPS fue creado para describir los componentes necesarios para que un agente planificador pudiera resolver problemas complejos en los que la estructura de los estados era determinante para entender qué proceso sería necesario llevar a cabo para resolver el problema de modo racional.

Este estándar sigue muy vigente en la definición de paradigmas de planificación actual con la creación de nuevos modelos como son PDDL y/o GOAP.

Para esta sección emplearemos el problema de apilado de bloques (el mundo de los bloques), en el que, desde una configuración de partida de un conjunto de bloques,

deseamos crear una configuración final de los mismos por medio de operaciones de apilado y desapilado.

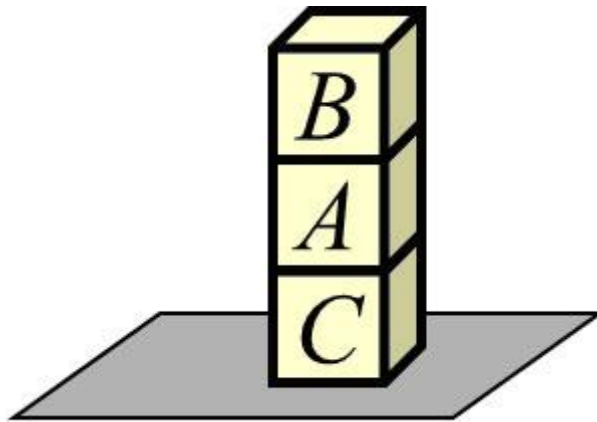


Figura 1. Bloques apilados.

Existen dos componentes básicos en una definición de problema de STRIPS:

- ▶ Propiedades.
- ▶ Operadores.

Propiedades

Para describir un estado, tendremos en cuenta el conjunto de propiedades relevantes que lo caracteriza. Estas propiedades, inicialmente, son expresadas por medio de valores *booleanos*, es decir, de existencia o no existencia en el estado actual.

Por ejemplo, para el problema del mundo de los bloques, con tres bloques tendríamos las siguientes propiedades relevantes para caracterizar estados:

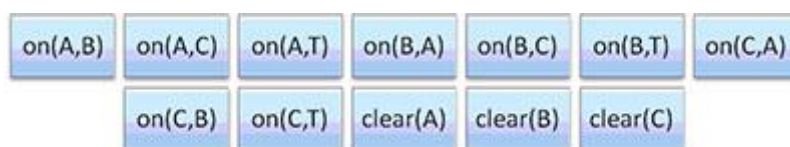


Figura 2. Propiedades para el mundo de los bloques.

Para el caso del ejemplo anterior, el estado representado en la imagen estaría formado por las siguientes propiedades:

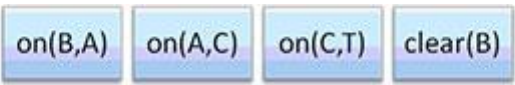


Figura 3. Ejemplo de propiedades de un apilado.

Es importante tener en cuenta que no todas las combinaciones de propiedades son **consistentes**. Por ejemplo, {on(C,B), on(B,C)} no representa un estado válido.

Operadores

Expresan las acciones que el agente contempla en el problema y por las cuales desea resolver el estado actual y transformar el entorno para alcanzar la meta o estado final.

Se definen por una signatura del tipo:

<nombre>(<PC>,<A>,>E>)		
PC	Precondición	Conjunto de propiedades que deben estar presentes en un estado para poder aplicar dicho operador.
A	Lista de añadir	Conjunto de propiedades que se añadirán al estado tras aplicar el operador.
E	Lista de eliminar	Son aquellas propiedades que se quitarán del estado tras la aplicación del operador. Debería ser un subconjunto de PC.

Tabla 1. Componentes de un operador de STRIPS.

move (A,B,C)	move (B,A,T)
PC: On (A,B) , Clear (A) , Clear (C)	PC: On (B,A) , Clear (B)
E: On (A,B) , Clear (C)	E: On (B,A)
A: On (A,C) , Clear (B)	A: On (B,T) , Clear (A)

Figura 4. Ejemplos de operadores en el mundo de los bloques.

En el esquema definido por STRIPS, los operadores se ejecutan teniendo en cuenta que un operador sea aplicable. Diremos que un operador es aplicable a un estado si las precondiciones del operador son un subconjunto del estado: $PC_{Op} \subseteq S$.

Cuando aplicamos un operador a un estado, produciremos un nuevo estado derivado en el cual deberemos eliminar las propiedades descritas en E y añadiremos las presentes en A.

$$S' \leftarrow S - E_{Op} \cup A_{Op}$$

Y asumiremos que, todas aquellas propiedades que no se especifican, no se dan y, aquellas que no se modifican, se quedan como estaban. Esto se define como la **asunción de mundo cerrado**.

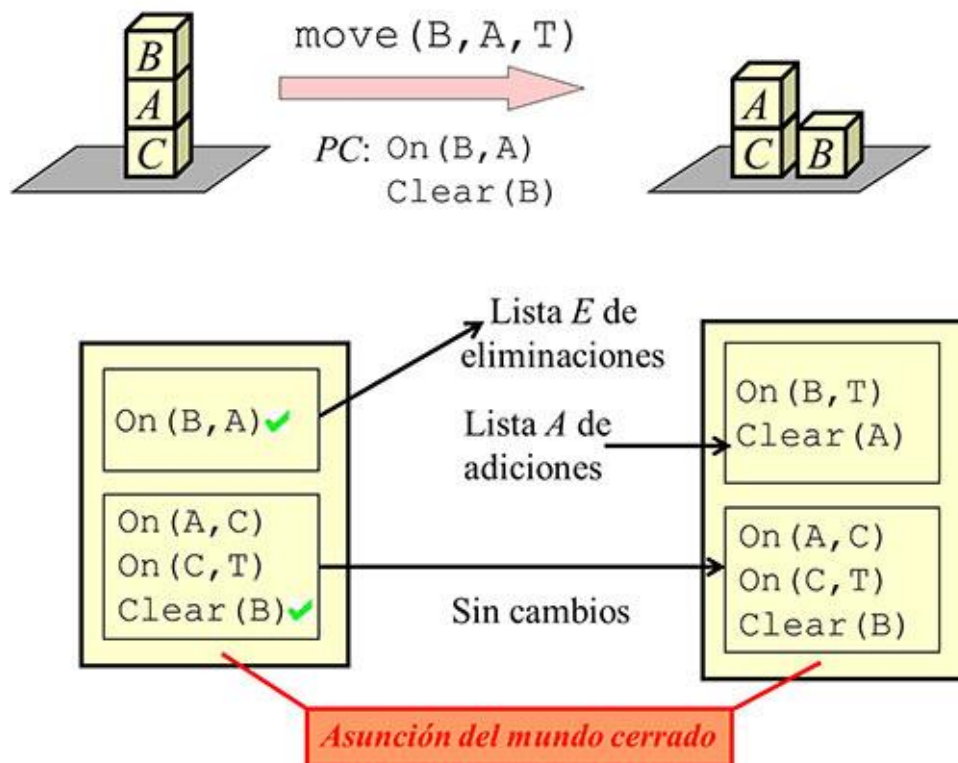


Figura 5. Esquema de ejecución de un operador STRIPS.

Planificación progresiva

La primera idea que nos surge para resolver un problema de planificación expresado por medio de estos operadores y propiedades STRIPS es crear una planificación que explore el espacio de búsqueda de estados, aplicando operadores como se hace en los problemas de búsqueda normales.

De este modo, y asumiendo el proceso de exploración y búsqueda que se basa en la existencia de un conjunto de acciones candidatas en un estado determinado, crearemos dicha función de expansión por medio de la identificación de todos aquellos operadores que sean aplicables en el estado.

Con esta idea, aplicamos algoritmos como los de búsqueda en amplitud, profundidad e incluso búsquedas heurísticas de A*.

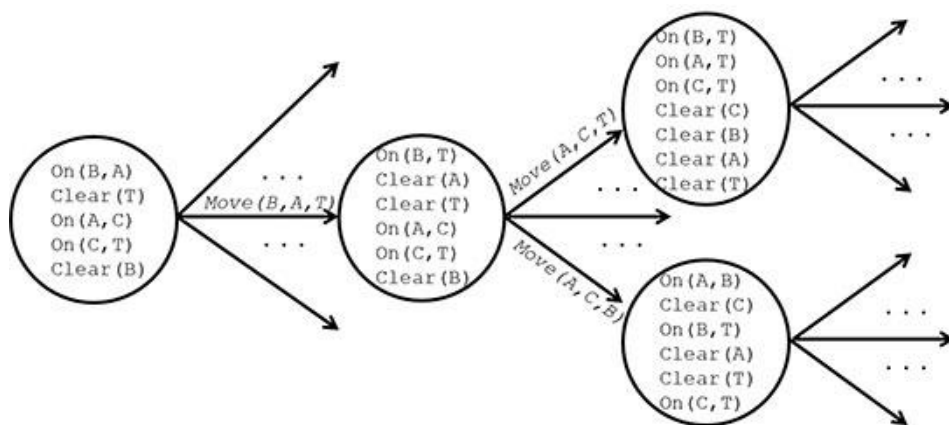


Figura 6. Ejemplo de planificación progresiva.

Pero rápidamente nos damos cuenta de que el factor de ramificación es muy elevado y supone un problema porque el estado inicial se describe por todas las propiedades que podrían ser relevantes (por ejemplo, las posiciones de diez bloques) y la descripción del estado meta suele ser parcial, es decir, solo contiene lo que realmente importa (por ejemplo, las posiciones de tres bloques).

Planificación regresiva

Dada la problemática de la planificación progresiva, surge la idea de aplicar operadores en orden inverso; es decir, para poder reducir el conjunto de propiedades no satisfechas de estado meta, aplicamos operadores de tal modo que podamos conseguir encontrar la secuencia de operadores que nos regresan desde el estado meta hasta el estado inicial.

Regresar un estado S' por un operador Op consiste en encontrar el conjunto de propiedades menos restrictivo (más débil) que permitiría aplicar el operador al estado S . Para *alcanzar* dicho conjunto, lo que hacemos es eliminar del estado la lista de A del operador y añadir aquellos elementos que no tenemos en PC .

$$S \leftarrow S' - A_{Op} \cup PC_{Op}$$

Pero para conseguir la aplicabilidad de dicho operador, ningún elemento de S' es eliminado (*e. d.*: $S' \cap EOp = \emptyset$).

Hay que tener cuidado porque en este proceso de aplicación no todos los estados que se obtienen de la regresión son **consistentes**.

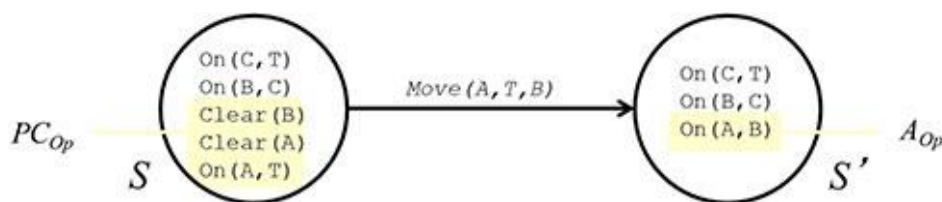


Figura 7. Regresión de un estado STRIPS de $S' \rightarrow S$.

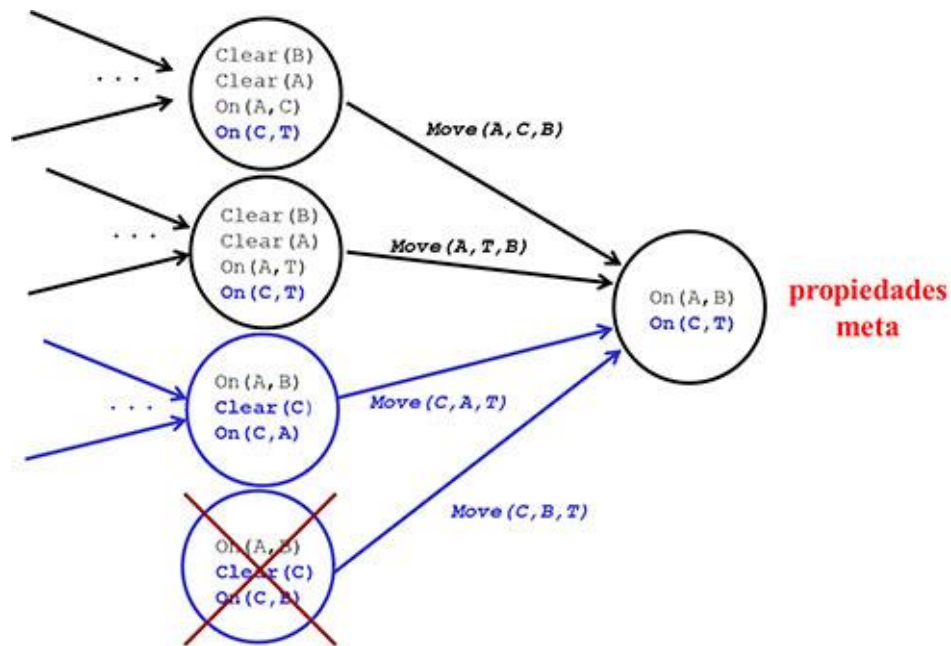


Figura 8. Ejemplo de regresión en el mundo de los bloques. Eliminación de los estados inconsistentes.

Heurística STRIPS

Aun contando con un factor de ramificación menor y un crecimiento de la exploración guiado por las metas, debemos combatir la complejidad de la planificación. Al igual que en las búsquedas, la idea es buscar un conocimiento *a priori* que nos mejore la complejidad, es decir, una heurística.

La **heurística STRIPS** consiste en encontrar los planes parciales para alcanzar cada una de las propiedades meta aparte. Para construir el plan final, concatenaremos todos los subplanes parciales que tengamos que construir.

Podemos definir la **planificación de tipo STRIPS** de la siguiente manera.

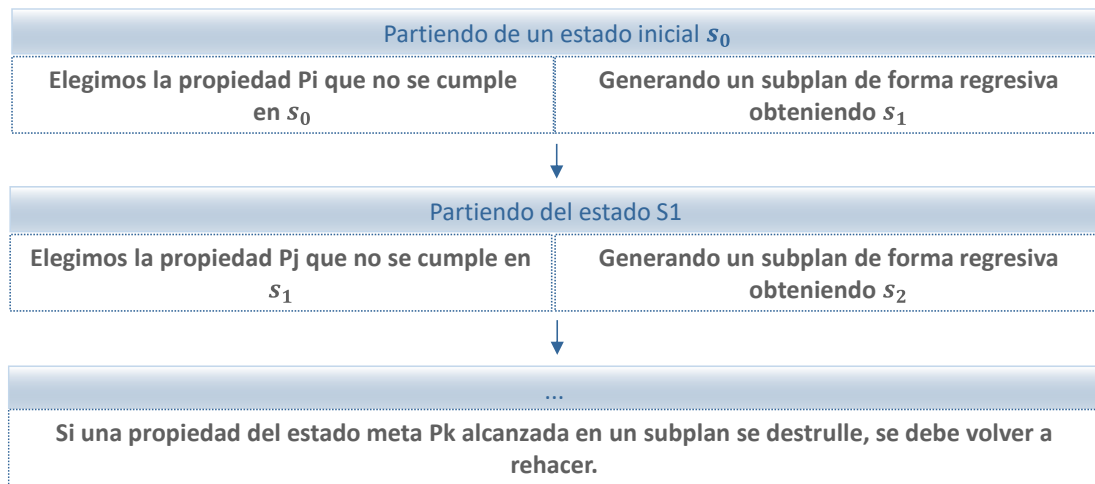


Figura 9. Planificación de tipo STRIPS.

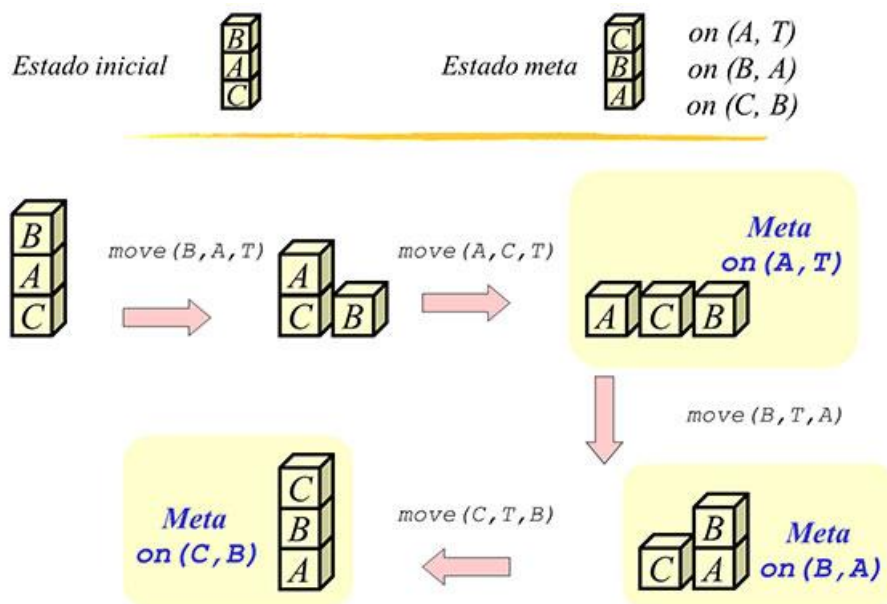


Figura 10. Uso de la heurística de STRIPS.

Pero pueden surgir problemas por el orden de exploración de las soluciones:

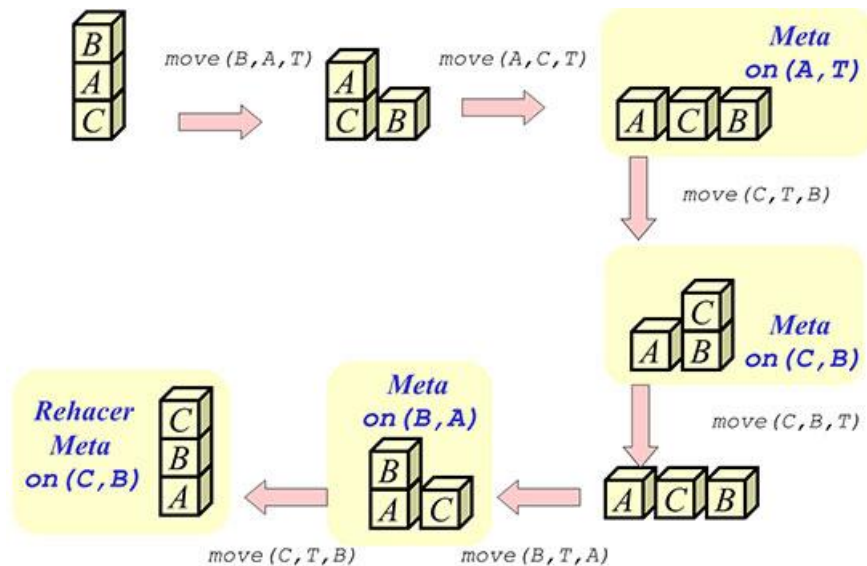


Figura 11. El orden de exploración puede suponer problemas.

El orden de elección de metas y/u operadores influye en los planes obtenidos.

```

STRIPS(S, Metas, Plan) devuelve (SMeta, PlanMeta) ó false
  Mientras que Metas  $\not\subseteq$  S Hacer                                % Mientras haya Metas no satisfechas en S
    (1) Elegir  $M \in \text{Meta}$  tal que  $M \notin S$                   % M no está satisfecha en S
    (2) Elegir operador Op tal que  $M \in A_{Op}$                 % Op puede alcanzar M
    (3)  $(S, \text{Plan}) \leftarrow \text{STRIPS}(S, PC_{Op}, \text{Plan})$       % Alcanzar precond. de Op
    (4) Si STRIPS devuelve false Entonces falla
    (5)  $S \leftarrow S - E_{Op} \cup A_{Op}$                         % aplicar el Op al nuevo estado S (resultado de (3))
    (6)  $\text{Plan} \leftarrow \text{Plan} + Op$                             % añadir el Op al final de nuevo Plan (resultado de (3))
  Fin
  Devolver(S, Plan)
  
```

Figura 12. Algoritmo no determinista de la heurística STRIPS.

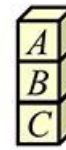
La heurística STRIPS es una heurística fuerte que implica que las metas son independientes, pero para conseguir alguna propiedad se puede estar destruyendo de modo cíclico otra propiedad, quedando atrapado en un bucle de destrucción de propiedades. De este modo, no se podría alcanzar un plan global óptimo a partir de los subplanes parciales debido a que, por diseño, la linealización de la heurística de STRIPS no permite intercalar acciones de dos submetas independientes. Para poder solucionarlo, deberíamos poder intercalar operadores de los distintos subplanes.

Estado inicial:



Estado meta:

$On(A, B)$
 $On(B, C)$



Plan óptimo:

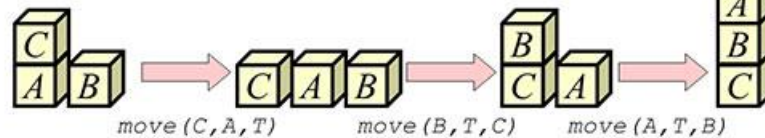


Figura 13. El plan óptimo para un caso del mundo de los cubos.

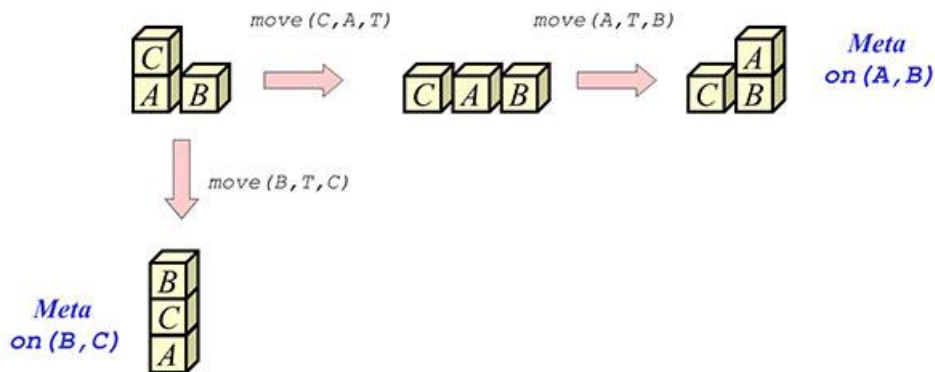


Figura 14. Anomalía de Sussman

Hay metas que suponen la destrucción de submetas obtenidas: anomalía de Sussman.

9.3. PDDL

El *Planning Domain Description Language* (PDDL) fue creado por Drew McDermott y su equipo. Se basaba en los estándares especificados de STRIPS y ADL (McDermott, Ghallab, Howe, Knoblock, Ram y Veloso, 1998).

El objetivo inicial era crear un lenguaje común y estándar de creación de planes para emplear como *benchmark* entre planificadores, y así poder comparar agentes de

planificación en competiciones o estudios. En la actualidad, es un estándar que presenta varias versiones, desde la 1.0 a la 3.1, cada una de ellas con diferentes niveles de expresividad (si bien es cierto que actualmente no hay ningún planificador que soporte la versión 3.1 completa).

Se basa en una descripción de los componentes de un planificador en dos conjuntos: uno de **definición del dominio** y otro de **definición del problema**.

```
(define (domain DOMAIN_NAME)

  (:requirements [:strips] [:equality] [:typing] [:adl])

  (:predicates
    (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
    (PREDICATE_2_NAME [?A1 ?A2 ... ?AN]) ...)

  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA] )
  (:action ACTION_2_NAME ...)

  ...)
```

Figura 15. Definición del dominio del planificador.

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA) )
```

Figura 16. Definición del problema.

En general, en las especificaciones de PDDL se representa el conjunto de objetos del mundo (**constantes**, en mayúsculas) y se usan **variables** para representar cualquier objeto del universo del discurso. Para expresar propiedades de los objetos se emplean **símbolos de predicado** y los operadores los representamos por medio de **símbolos de acciones**.

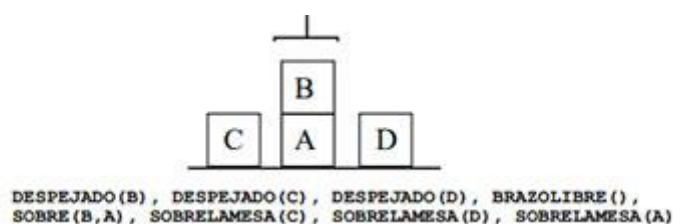
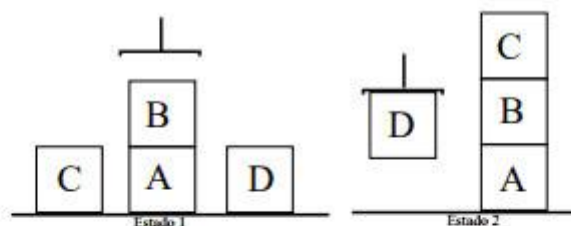


Figura 17. Ejemplo de descripción de propiedades. Fuente: Graciani y Romero, 2016.

A la hora de construir un problema completo, usamos las siguientes fórmulas de la forma $P(o_1, \dots, o_n)$, donde el predicado P se aplica a las variables o constantes o_i (estas fórmulas se llaman **átomos**). Con estos elementos atómicos formaremos **literales** que pueden ser afirmativos o negativos. En nuestra definición de un problema pueden existir átomos y literales cerrados, que no se aplican a ninguna variable. En toda la definición, el elemento clave son los **estados**, que definiremos por medio de conjuntos de átomos cerrados.



• Ejemplos de objetivos:

- **SOBRE(B,A), SOBRELAMESA(A), -SOBRE(C,B)** es satisfecho por el estado 1 y por el estado 2
- **SOBRE(x,A), DESPEJADO(x), BRAZOLIBRE()** es satisfecho por el estado 1 pero no por el estado 2
- **SOBRE(x,A), SOBRE(y,x)** no es satisfecho por el estado 1 pero sí por el estado 2
- El objetivo **SOBRE(x,A), -SOBRE(C,x)** es satisfecho por el estado 1 pero no por el estado 2

Figura 18. Ejemplo de objetivos en PDDL. Fuente: Graciani y Romero, 2016.

Al igual que en la descripción de STRIPS, la hipótesis del mundo cerrado hace que los átomos que no sean nombrados explícitamente en una descripción sean considerados como falsos.

9.4. GOAP

El sistema *Goal - Oriented Action Planning (GOAP)* fue creado por Jeff Orkin en 2004 (*Symbolic Representation of Game World State: Toward real-time planning in games*) con el fin de adaptar un sistema planificador como STRIPS a un entorno de simulación juegos. Fue implementado por primera vez en el videojuego F.E.A.R., de Monolith (2005) (Orkin, 2006).

Puedes ver más sobre F.E.A.R en el siguiente enlace:

<https://www.3djuegos.com/juegos/analisis/750/0/f-e-a-r/>

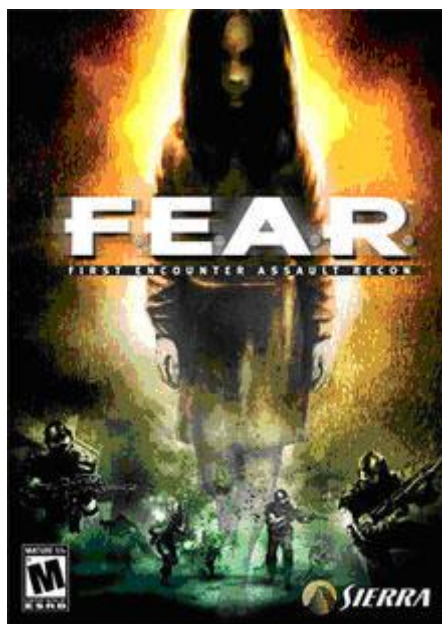


Figura 19. Portada del videojuego F.E.A.R, de Monolith (2005). Fuente:

<https://www.3djuegos.com/juegos/pc/750/f-e-a-r/>

Orkin se basó en la arquitectura de agentes C_4 del MIT y extendió ideas del planificador de STRIPS. Posee cuatro diferencias con STRIPS:

- Establece costes a las acciones con el fin de poder asignar prioridad a unas acciones frente a otras, que es una consideración necesaria para el diseño de agentes en videojuegos.

- ▶ Elimina las listas de añadir y eliminar objetos y las convierte en una única lista de modificaciones del estado en la que las propiedades pueden ser modificadas de manera más flexible que por simple operación *booleana*.
- ▶ Añade precondiciones procedurales que permiten mayor flexibilidad a la hora de expresar condiciones que se deban dar en el entorno para poder aplicar un operador.
- ▶ Añade efectos procedurales con la misma filosofía de poder modificar el entorno con mayor flexibilidad.

La primera consideración es que representa las propiedades de un estado dentro de un vector *multitipado*, en lugar de considerar las propiedades como valores *booleanos* aislados.



Ejemplo:

$(\text{AtLocation}, \text{Wearing}) = (\text{Home}, \text{Tie})$

Figura 20. Ejemplo de par estado valor que representa una propiedad GOAP.

El resto de elementos de definición del dominio del problema y del entorno siguen las especificaciones de STRIPS. Contamos con precondiciones necesarias en una acción (operador) que produce cambios en las propiedades del entorno para alcanzar una meta.

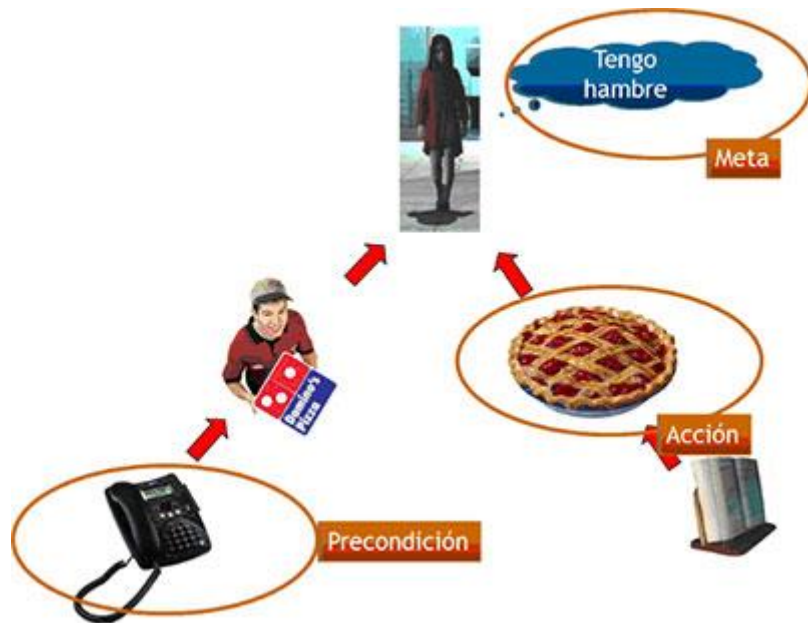


Figura 21. Ejemplo de problema análogo en STRIPS y GOAP. Fuente: adaptada de Orkin (2006).

La idea es tener varios modelos de agente que puedan emplear el mismo motor de planificación con solo tener que definir distintos conjuntos de acciones, lo cual es de mucha importancia en entornos como el de los videojuegos, donde debemos diseñar varios tipos de agentes que resuelvan el problema con distintas capacidades.

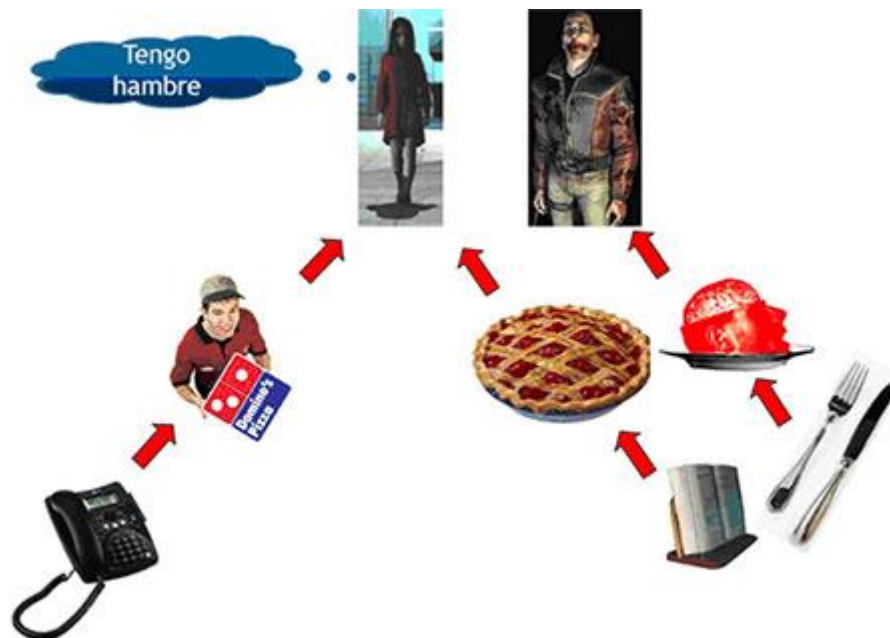


Figura 22. Varios agentes con el mismo problema, pero varios conjuntos de acciones distintas.

Fuente: adaptada de Orkin (2006).

En tiempo de diseño, en estos entornos es importante poder tener en cuenta distintas preferencias que caractericen a los personajes de un modo sencillo. Para ello, en GOAP se permite precisar un coste específico de cada acción y, en el proceso de búsqueda de planes, se puede aplicar un algoritmo como A* para hacer una búsqueda de costes mínimos en la consecución de un plan.

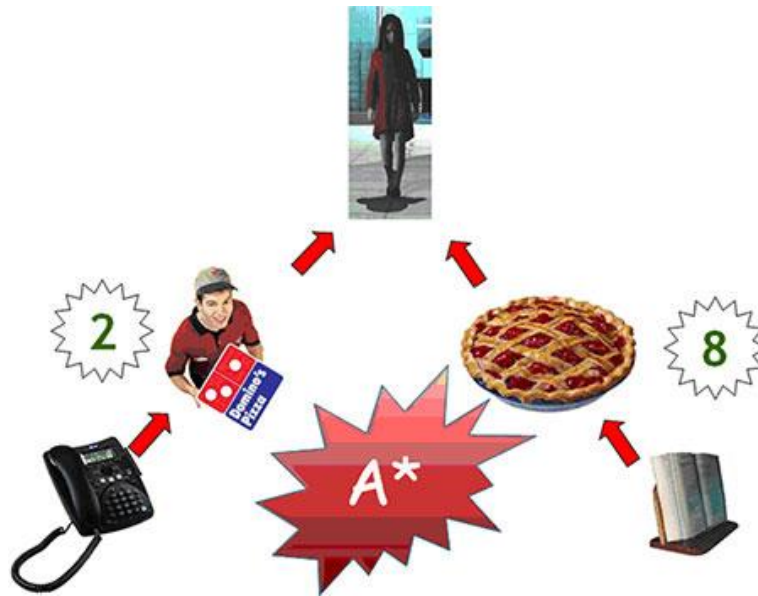


Figura 23. Aplicando distintos costes a las acciones en GOAP. Fuente: adaptada de Orkin (2006).

Para resolver el problema de las cláusulas de mundo cerrado y permitir mecanismos de aceleración del cómputo dentro del motor, se suprimen las listas de añadir y eliminar y se crea una única lista de modificaciones.

State: (phone#, recipe, hungry?)



Action

Preconditions: (phone, --, --)

Effects:

Delete List: Hungry(YES)


Add List: Hungry(NO)

Figura 24. Ejemplo de operador de STRIPS. Fuente: adaptada de Orkin (2006).

State: [phone#, recipe, hungry?]



Action

Preconditions: [ , -- , --]

Effects: [-- , -- , **NO**]

Figura 25. Ejemplo de operador de GOAP. adaptada de Orkin (2006).

Y un vector de propiedades *multitipado* para almacenar los valores que alcanzarán las variables del estado tras la aplicación de un operador.

[-- , -- , --] 4-byte values

TargetDead	[bool]
WeaponLoaded	[bool]
OnVehicleType	[enum]
AtNode	[HANDLE]
- or -	
AtNode	[variable*]

Figura 26. Posible vector de propiedades para un sistema GOAP.

Entenderemos en GOAP que las acciones son clases y, como tales, representan las precondiciones como un *array* de variables de estado del mundo. Estas precondiciones serán como una función para permitir filtros adicionales. Esta función permite la ejecución de cualquier trozo de código que fuera necesario.

```

class Action
{
    // [ -- , -- , -- ]
    WORLD_STATE m_Preconditions;
    WORLD_STATE m_Effects;

    bool CheckPreconditions();
};

```

Figura 27. Acciones con precondiciones procedurales. Fuente: Orkin (2006).

Sucede lo mismo con las acciones que se podrán programar como un efecto procedural. Esto nos permite delegar el proceso de cambio en el entorno y gestionar interrupciones que se puedan producir del plan en tiempo real.

```

class Action
{
    // [ -- , -- , -- ]
    WORLD_STATE m_Preconditions;
    WORLD_STATE m_Effects;

    bool CheckPreconditions();
    void ActivateAction();
};

```

Figura 28. Acciones procedurales. Fuente: Orkin (2006).

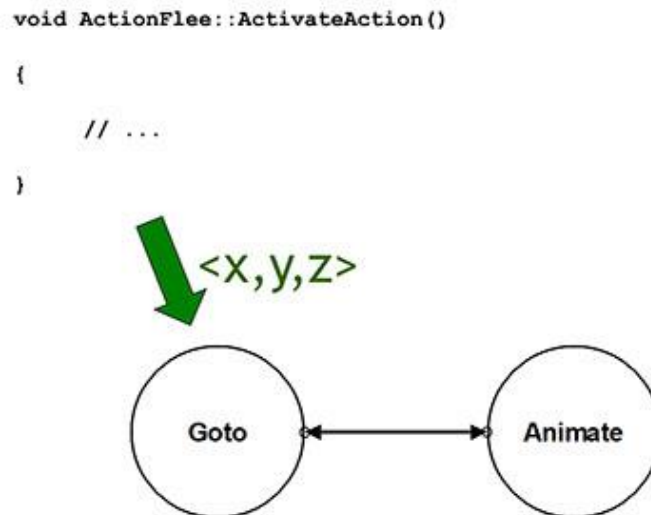


Figura 29. Una acción puede suponer varios ciclos de reloj, por lo que no son atómicas.

9.5. Referencias bibliográficas

Fikes, R. y Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*. 2(3-4), 189-208.

Graciani, C. y Romero, A. (2016). *Curso de inteligencia artificial*. Sevilla: Universidad de Sevilla.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M. et al. (1998). *PDDL - The Planning Domain Definition Language*. Technical Report CVC TR98003/DCS TR1165. Yale Center for Computational Vision and Control. Recuperado de: <http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf>

Orkin, J. (2004). Symbolic Representation of Game World State: Toward real-time planning in games. *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence*, 5, 26-30.

Orkin, J. (2006). Three states and a plan: the AI of FEAR. *Proceedings of the 2006 Game Developers Conference (GDC '06)*. San José, California.

Lo + recomendado

No dejes de leer

HTN Planning en juegos

Champanard, A. (2010). Hierarchical Task Networks for Mission Generation and Real-Time Behaviour [Mensaje en un blog]. Aigamedev.com.

Una introducción general y resumida de la aplicación de HTN en un posible entorno de juegos.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

<http://aigamedev.com/open/coverage/htn-planning-discussion/>

No dejes de ver

GOAP

La charla de Game Developers Conference 2015 sobre GOAP.



Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=gm7K68663rA&feature=youtu.be>

A fondo

GOAP C++

Gutiérrez, C. (s.f.). GOAP AI [Mensaje en un blog]. Carlosplusplus.

Un ejemplo de programación de GOAP en un *engine* de C++.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

<https://www.carlosplusplus.com/goal-oriented-action-planning.html>

STRIPS en juegos

Becker, K. (2015). Artificial Intelligence Planning with STRIPS, a Gentle Introduction. [Mensaje en un blog]. Primary Objects.

Introducción sobre STRIPS y su aplicación en los juegos.

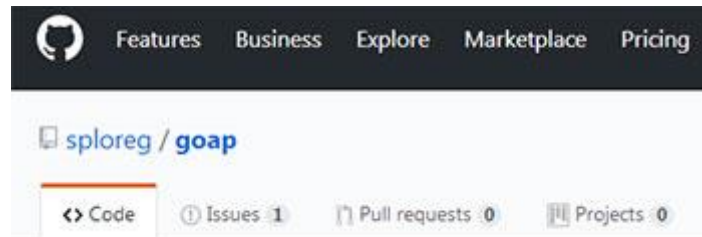
Accede al documento a través del aula virtual o desde la siguiente dirección web:

<http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/>

Webgrafía

GOAP

Ejemplo de implementación de GOAP.



Accede a la página web a través del aula virtual o desde la siguiente dirección web:

<https://github.com/sploreg/goap>

Jeff Orkin

Página web de Jeff Orkin en la que puedes encontrar información sobre el GOAP.



Accede a la página web a través del aula virtual o desde la siguiente dirección web:

<http://alumni.media.mit.edu/~jorkin/goap.html>

Bibliografía

McDermott, D. (2000). The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2), 35-55.

1. ¿Qué afirmación sobre STRIPS es verdadera?
 - A. STRIPS define problemas de búsqueda.
 - B. Los operadores STRIPS se pueden proyectar en las acciones de un agente.
 - C. STRIPS es un planificador de orden parcial.

2. Una propiedad definida para STRIPS es:
 - A. De tipo entero.
 - B. De tipo *booleano*.
 - C. De cualquier tipo.

3. Una planificación hacia delante:
 - A. Tiene un factor de ramificación potencialmente muy alto.
 - B. Empieza en las metas a conseguir y explora las acciones hasta el estado inicial.
 - C. Se descompone en tareas.

4. Un operador de STRIPS está compuesto de:
 - A. Precondiciones, lista de acciones y lista de estados.
 - B. Precondiciones, lista de adición y lista de eliminación.
 - C. Precondiciones y lista de modificación.

5. PDDL es:
 - A. Un lenguaje estándar para la definición de problemas de planificación.
 - B. Un algoritmo de tareas jerárquicas.
 - C. Un estándar con una especificación única, la 1.7.

6. Las acciones en GOAP:
 - A. Son atómicas.
 - B. Son secuenciales.
 - C. Son procedurales.

7. La heurística de STRIPS:

- A. Descompone el problema en subplanes que concatena para resolver cada una de las propiedades abiertas en el estado meta.
- B. No usa operadores.
- C. Emplea planificación hacia delante.

8. Las tareas pueden tener acciones que no se implementen directamente en el entorno:

- A. Sí, porque pueden ser parte de una jerarquía de controladores.
- B. No, todas las tareas deben ejecutar acciones concretas del mundo.
- C. No, porque las tareas pueden descomponerse siempre en subtareas.

9. La anomalía de Sussman:

- A. Deriva de las acciones que se concatenan en orden inverso.
- B. Presenta el problema de destruir y construir de modo infinito submetas.
- C. No se da en STRIPS.

10. En GOAP:

- A. Existe una lista de añadir y otra de eliminar en los operadores.
- B. Todas las acciones deben tener el mismo coste.
- C. Existe una única lista de modificación de propiedades en los operadores.