



**decsai.ugr.es**

Universidad de Granada

## **Tema 3: Aprendizaje en Planificación Automática**

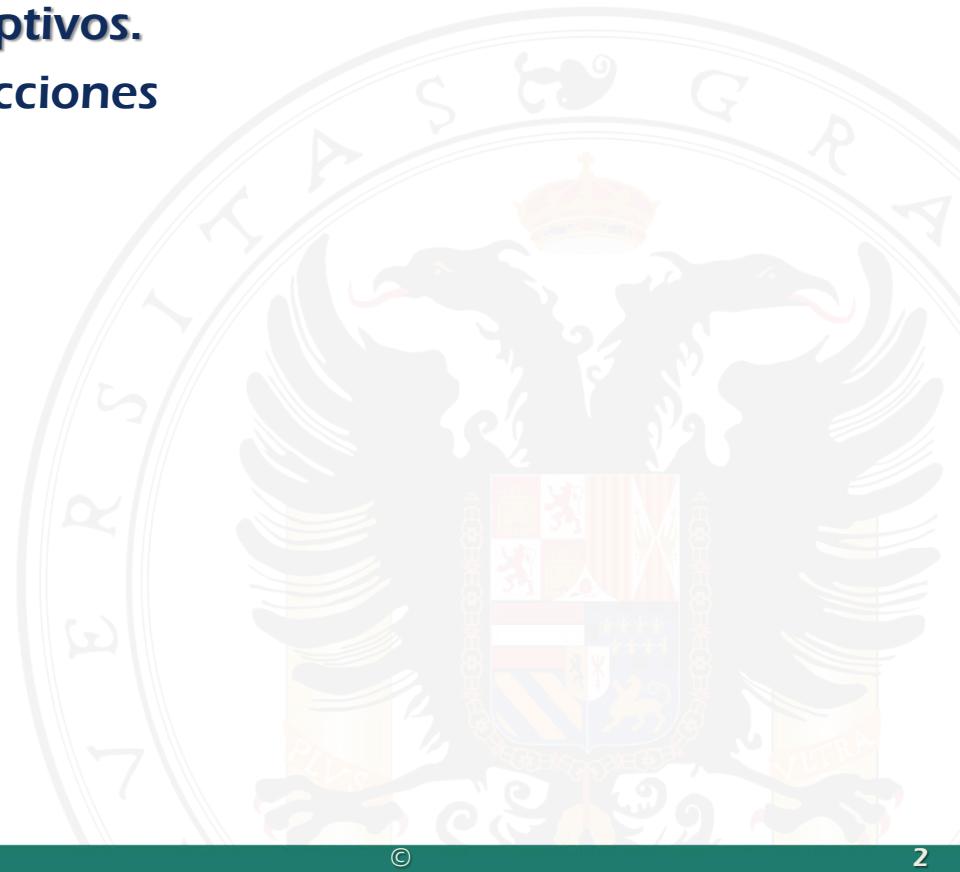
**Curso en Minería de Procesos**

**Juan Fernández Olivares**



**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**

- 1. ¿Por qué aplicar aprendizaje en Planificación?**
- 2. ¿Dónde y qué aprender?**
- 3. Autonomía y aprendizaje por refuerzo en agentes reactivos.**
  1. Para entender aprendizaje en deliberativos.
- 4. Mejorar la autonomía del planificador**
  1. Aprender modelos prescriptivos.
    1. Modelos basados en acciones
    2. Modelos jerárquicos.



## Mejorar el comportamiento autónomo

- Para sistemas autónomos la necesidades imponen que los agentes se adapten al entorno, para reducir barreras de adopción por usuarios.
  - Pocos usuarios aceptarían un robot que tienen que programar,
  - Los videojuego con NPCs inteligentes son más atractivos y tienen mayor índice de adopción (más ventas).

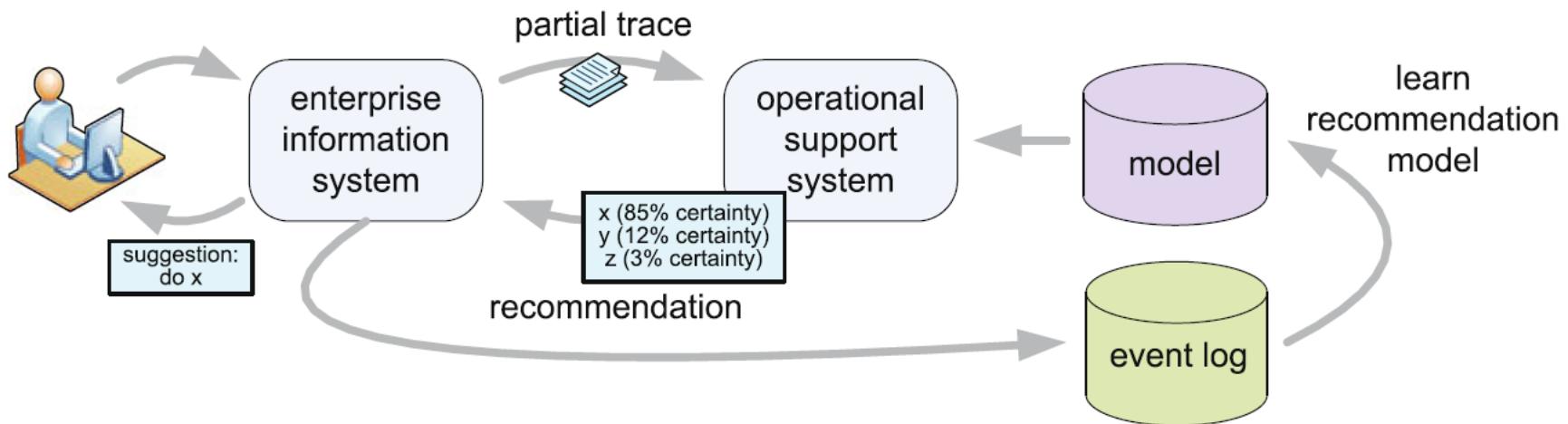
## • Reducir el esfuerzo de modelado (ayudar a Ingenieros del Conocimiento)

- Para herramientas inteligentes (como las usadas en ayuda a la toma de decisiones) el modelado es un cuello de botella que dificulta la adopción.
  - ¿Para qué esforzarme en codificar un conocimiento del que ya dispongo ejemplos en los logs?.

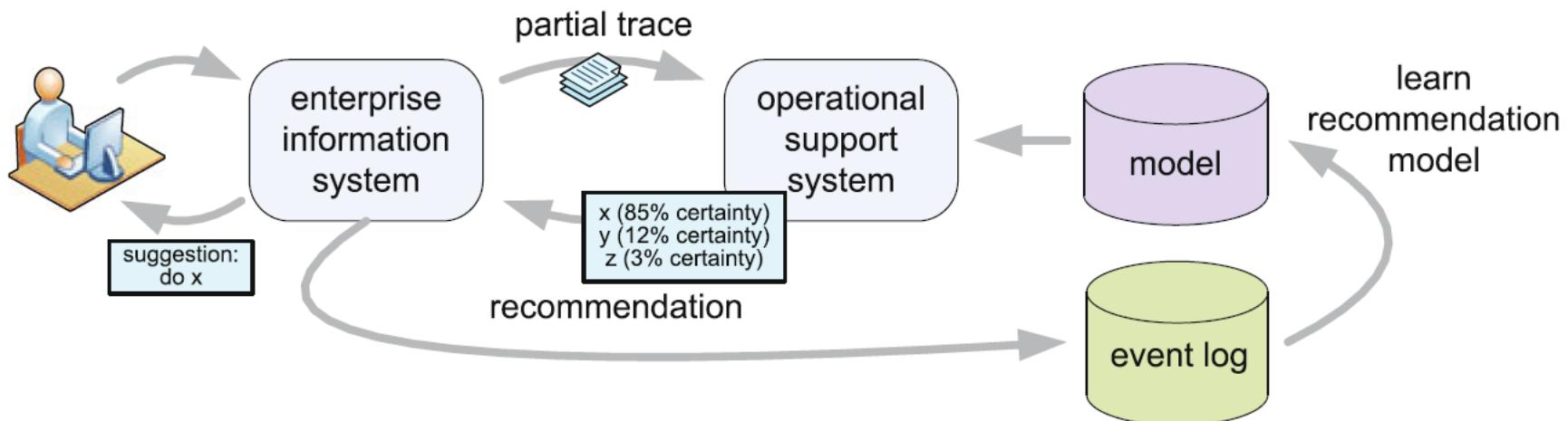
## • Ayudar a los usuarios (expertos de un dominio) a tomar mejores decisiones.

- La interacción con el usuario y la experiencia acumulada en la resolución de episodios permite mejorar el modelo de proceso (dominio de planificación) y afinar mejor en las decisiones.
- Mejorar eficiencia del proceso de planificación (**aprender heurísticas**)
  - La planificación es un proceso de búsqueda, con espacios de búsqueda exponenciales. El aprendizaje a partir de episodios previos ayuda a reducir las alternativas posibles en el proceso de planificación.

- Caso de uso: predecir el tiempo restante de un proceso o recomendar acciones para minimizar coste.
- ¿Qué funcionalidades son interesantes para ayuda a la decisión online?
  - **Detectar:** eventos y modelos se usan para explorar procesos en tiempo de ejecución. Los casos actuales en ejecución pueden compararse con casos similares previos.
  - **Predecir:** Combinando información de casos en ejecución con modelos es posible hacer predicciones sobre el futuro (¿cuánto tiempo resta para finalizar, qué probabilidad de éxito tiene el proceso?)
  - **Recomendar:** la información usada para predecir puede también ser usada para **recomendar** posibles acciones orientadas a objetivos (minimizar el coste o el tiempo ...)



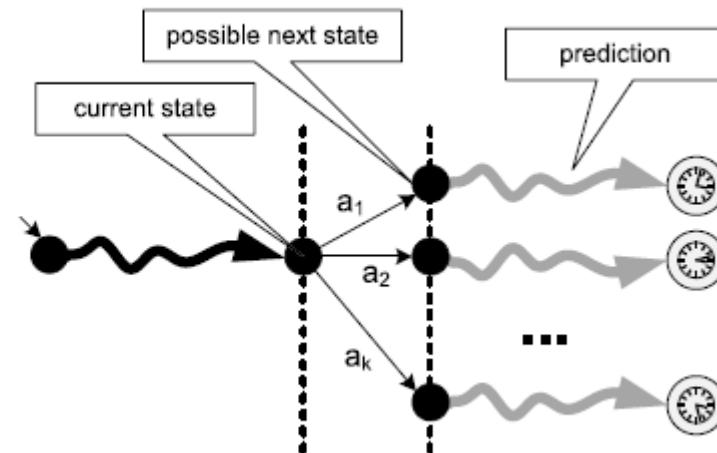
- Recomendación/prescripción: ¿qué se puede hacer a continuación?. Elementos:
  - Aprender un **modelo prescriptivo** a partir de logs de eventos.
  - El Sistema de Soporte Operativo (motor BPM, p. ej) necesita conocer el **espacio de decisión** (el conjunto de posibles acciones a escoger)
  - Las acciones se ordenan en el tiempo de acuerdo al modelo prescriptivo
  - Ejemplo: el sistema recomienda “hacer x con un 85% de certidumbre”
  - No siempre es posible una recomendación óptima
  - La mejor elección para la siguiente acción (o etapa) depende de eventos exógenos en el futuro.



- Una prescripción/recomendación se hace siempre respecto a un *objetivo/meta específico*:
  - Minimizar el tiempo restante
  - Minimizar los costes totales
  - Maximizar la fracción de casos gestionados en 4 semanas
  - Maximizar la fracción de casos
- Los objetivos pueden agregarse/combinarse: por ejemplo, balancear reducción de coste y de tiempo restante.
- Para hacer operativos objetivos, hay que definir indicadores de rendimiento (KPI: key performance indicators). P.e: en los logs tiene que aparecer información sobre tiempo o coste de actividades.

**Fig. 10.12**

Recommendations can be based on predictions. For every possible choice, simply predict the performance indicator of interest. Then, recommend the best one(s)



- Las recomendaciones no solo se refieren al orden en que se ejecutan las acciones, se consideran otras perspectivas: recursos, tiempo, etc.
- En el estado actual
  - ¿cuál es la mejor acción candidata dado el objetivo seleccionado.?
  - ¿cuál es el mejor recurso para ejecutar una acción.?
  - Por ejemplo “asignar a1 a Mike” para minimizar el tiempo restante
- **Si no disponemos de modelo prescriptivo**, podemos usar un modelo predictivo para simular la evolución de un indicador a partir de cada alternativa.

¿Cómo disponer de un modelo prescriptivo?

Nuestro foco de interés es

**modelo prescriptivo = dominio de planificación**

- **Diseñar un modelo prescriptivo** a mano es **escribir un dominio de planificación** usando lenguajes para planificación (PDDL, HPDL), como hemos visto en el tema anterior y en los ejercicios.
- **Aprender un modelo prescriptivo** es aprender **un dominio de planificación**:
  - considerando como observaciones las ejecuciones de trazas de planes
  - Aplicando un algoritmo de aprendizaje sobre esas observaciones para generar automáticamente un dominio de planificación (expresado en algún lenguaje de planificación)

El aprendizaje automático se puede aplicar a diferentes aspectos relacionados con la planificación automática.

- Aprender comportamiento de agente autónomo
- Aprender a plantearse objetivos.
- Aprender modelo de proceso (dominio de planificación)
- Aprender a mejorar el propio proceso de planificación

[1]

S. Jiménez, T. De la Rosa, S. Fernández, F. Fernández, y D. Borrajo, «A review of machine learning for automated planning», *The Knowledge Engineering Review*, vol. 27, n.º 04, pp. 433–467, 2012.

<http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>

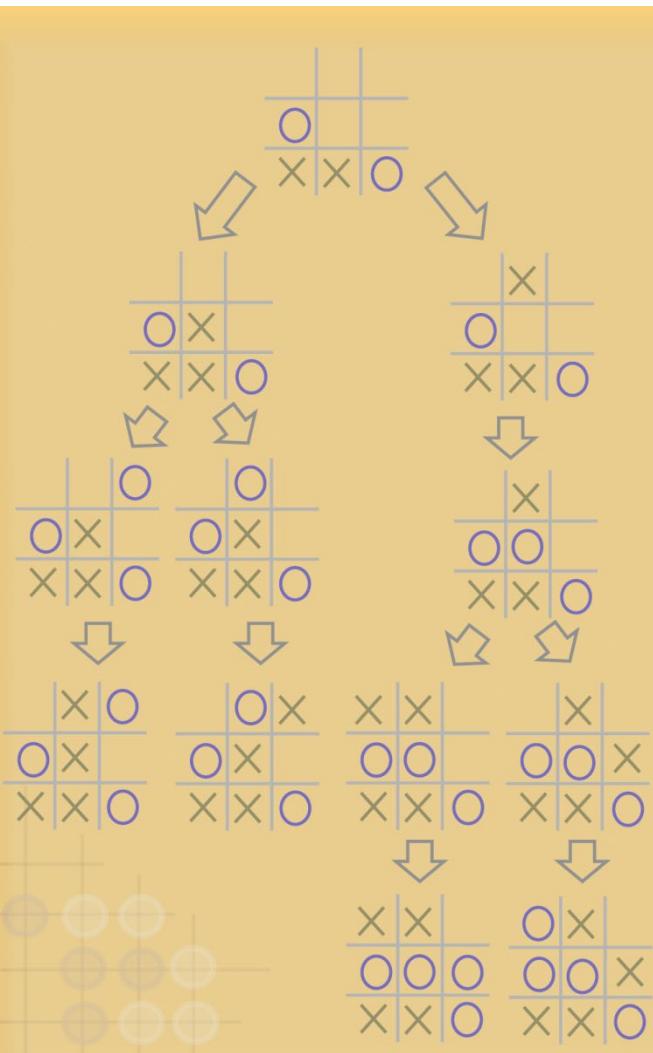
- Vamos a ver un ejemplo de agente reactivo que mejora su comportamiento.
- Atención: es un ejemplo motivador para ver cómo puede aprender un agente reactivo, con habilidades cognitivas de bajo nivel (reacciona ante situaciones, no delibera).
- El tema se centra en agentes deliberativos que establecen políticas de actuación (planes) para alcanzar objetivos a largo plazo.



Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. "Human-Level Control through Deep Reinforcement Learning." *Nature* 518, no. 7540 (February 26, 2015): 529–33. doi:10.1038/nature14236.

- Objetivo: aprendizaje del modelo de comportamiento, mostrando rendimiento de nivel humano.
  - Empresa DeepMind, adquirida por Google.
- Inteligencia Artificial como ciencia experimental.
  - Psicología cognitiva, neurociencia, inteligencia artificial -> mejora del comportamiento animal -> estudio del comportamiento humano a nivel cognitivo.
- Pero ... **agente reactivo**, puede afinar al máximo su reacción a partir de su experiencia previa, pero **no puede planificar**, anticiparse a lo que puede ocurrir.
- Nosotros nos centramos en agentes deliberativos, pero vamos a ver en un poco más detalle aprendizaje por refuerzo

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. «Mastering the Game of Go with Deep Neural Networks and Tree Search». *Nature* 529, n.º 7587 (28 de enero de 2016): 484-89.  
doi:10.1038/nature16961.



Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. «Mastering the Game of Go with Deep Neural Networks and Tree Search». *Nature* 529, n.º 7587 (28 de enero de 2016): 484-89.  
doi:10.1038/nature16961.



Búsqueda en profundidad  
 $b^d$

Impracticable en ajedrez ...  
 $b \approx 35, d \approx 80$

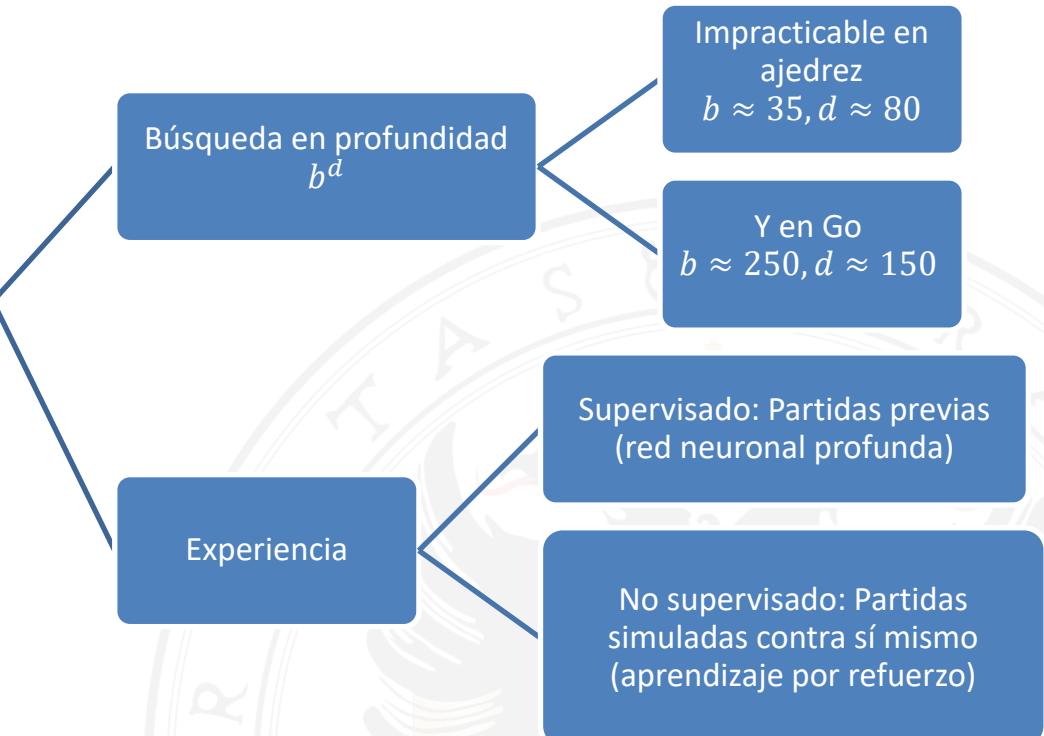
...y en Go  
 $b \approx 250, d \approx 150$

Supervisado: Partidas previas (red neuronal profunda)

Experiencia

No supervisado: Partidas simuladas contra sí mismo (aprendizaje por refuerzo)

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. «Mastering the Game of Go with Deep Neural Networks and Tree Search». *Nature* 529, n.º 7587 (28 de enero de 2016): 484-89.  
doi:10.1038/nature16961.



Go es modélico: toma de decisiones deliberativas, espacio de búsqueda intratable, solución óptima aparentemente inviable

# Aprender objetivos

Núñez-Molina, Carlos, Juan Fernández-Olivares, and Raúl Pérez. "Learning to Select Goals in Automated Planning with Deep-Q Learning." *Expert Systems with Applications* 202 (September 15, 2022): 117265. <https://doi.org/10.1016/j.eswa.2022.117265>.



- **Eficiencia:** Al aprender a seleccionar submetas adecuadas, el sistema puede planificar rutas más eficientes, reduciendo el tiempo y los recursos computacionales necesarios.
- **Generalización:** El método puede generalizar su aprendizaje a nuevos niveles del juego o variaciones en la distribución de las gemas y obstáculos.

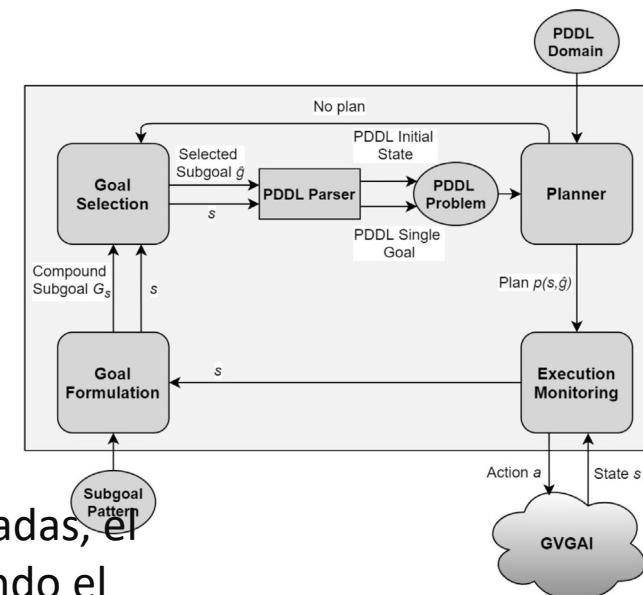
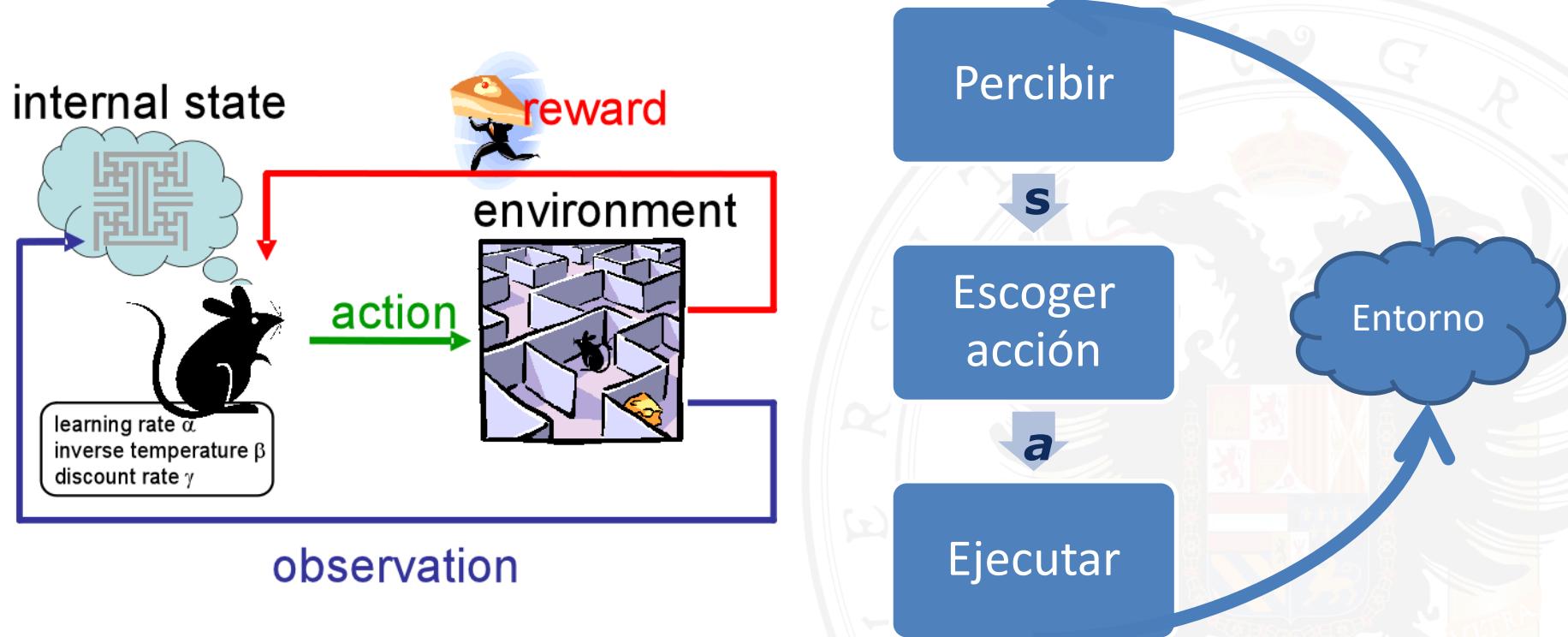


Fig. 2. An overview of the planning and acting architecture.

## Aprender modelos de acciones (modelos prescriptivos)

- No se conocen los efectos de las acciones (y las Acciones son no deterministas)
  - Pero mediante la observación de la ejecución obtenemos información (recompensa) de los estados resultantes.
  - Aprendizaje por refuerzo
- Se conocen los efectos de las acciones (hay un modo basado en observación para saberlo)
  - Aprendizaje de modelos de acciones.

Aprendizaje por refuerzo para un agente reactivo: determinar cuál es la **acción más adecuada para cada estado** posible en el mundo, recibiendo recompensas ( premio o castigo) por el comportamiento realizado.



Ejemplo: un robot en mundo cuadriculado que sólo puede ejecutar 4 acciones. Sólo se pueden percibir estados y **no existe relación conocida entre estado y acción**. Pero sí se conocen las recompensas.

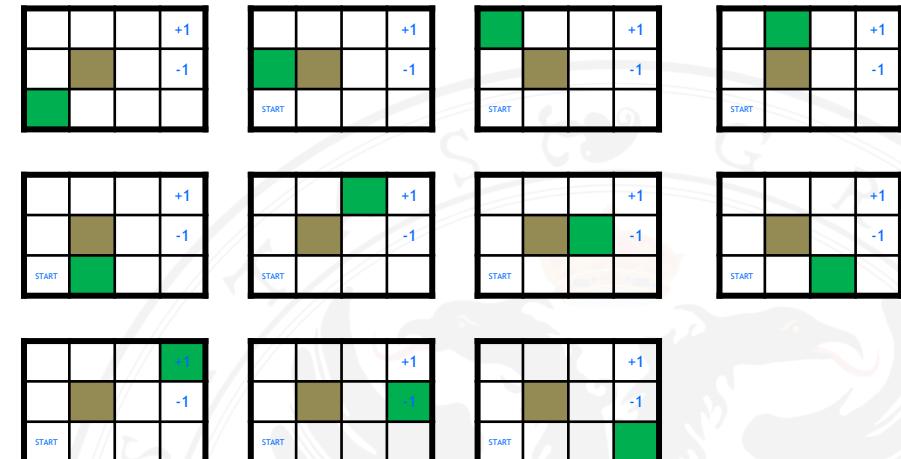
## Recompensas

-0.04	-0.04	-0.04	Premio +1
-0.04	Muro	-0.04	Castigo -1
START	-0.04	-0.04	-0.04

Premio si llega a [4,3], Castigo si llega a [4,2] y pequeña penalización por cada paso

**Acciones:** UP, DOWN, LEFT, RIGHT

## Estados



[1]

R. S. Sutton y A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Mass: A Bradford Book, 1998

2ª edición descargable en pdf (incompleta)

- Objetivo:

- Determinar una **política de actuación**-> ¿qué acción selecciono en cada instante (estado) para ejecutar?
- Política de actuación: una función  $\Pi(s):S \rightarrow A$  que asocia cada estado con una acción (representada por la tabla de abajo)
- **Utilidad**: la aplicabilidad de una acción  $a$  en un estado  $s$  tiene asociado un valor de utilidad  $Q(a,s)$ .
- Se trata de encontrar la política que maximiza la utilidad.

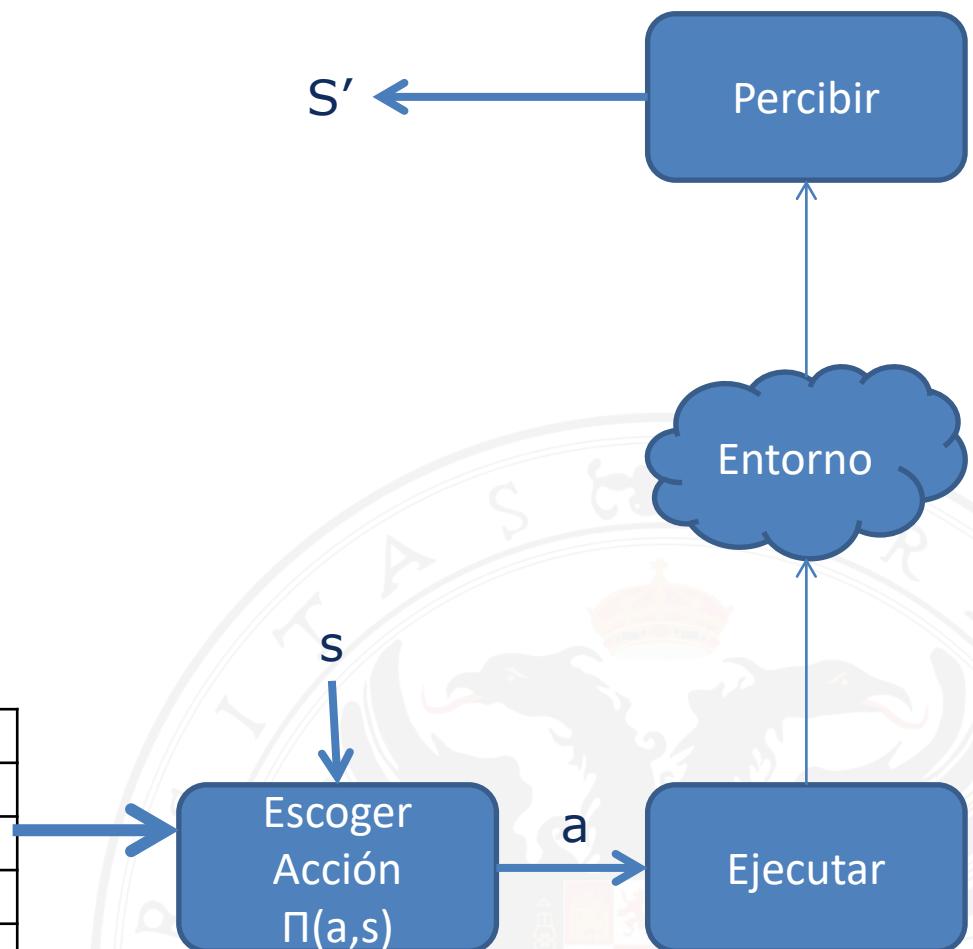


	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
UP	Q(1,1)	Q(1,2)	Q(1,3)								
RIGHT	Q(2,1)										
LEFT	Q(3,1)										
DOWN	Q(4,1)										

El aprendizaje por refuerzo **sólo es aplicable si se apoya en la ejecución** en el mundo real o en la ejecución en una simulación para obtener el estado resultante de ejecutar acciones.

Política  $\Pi(a,s)$ 

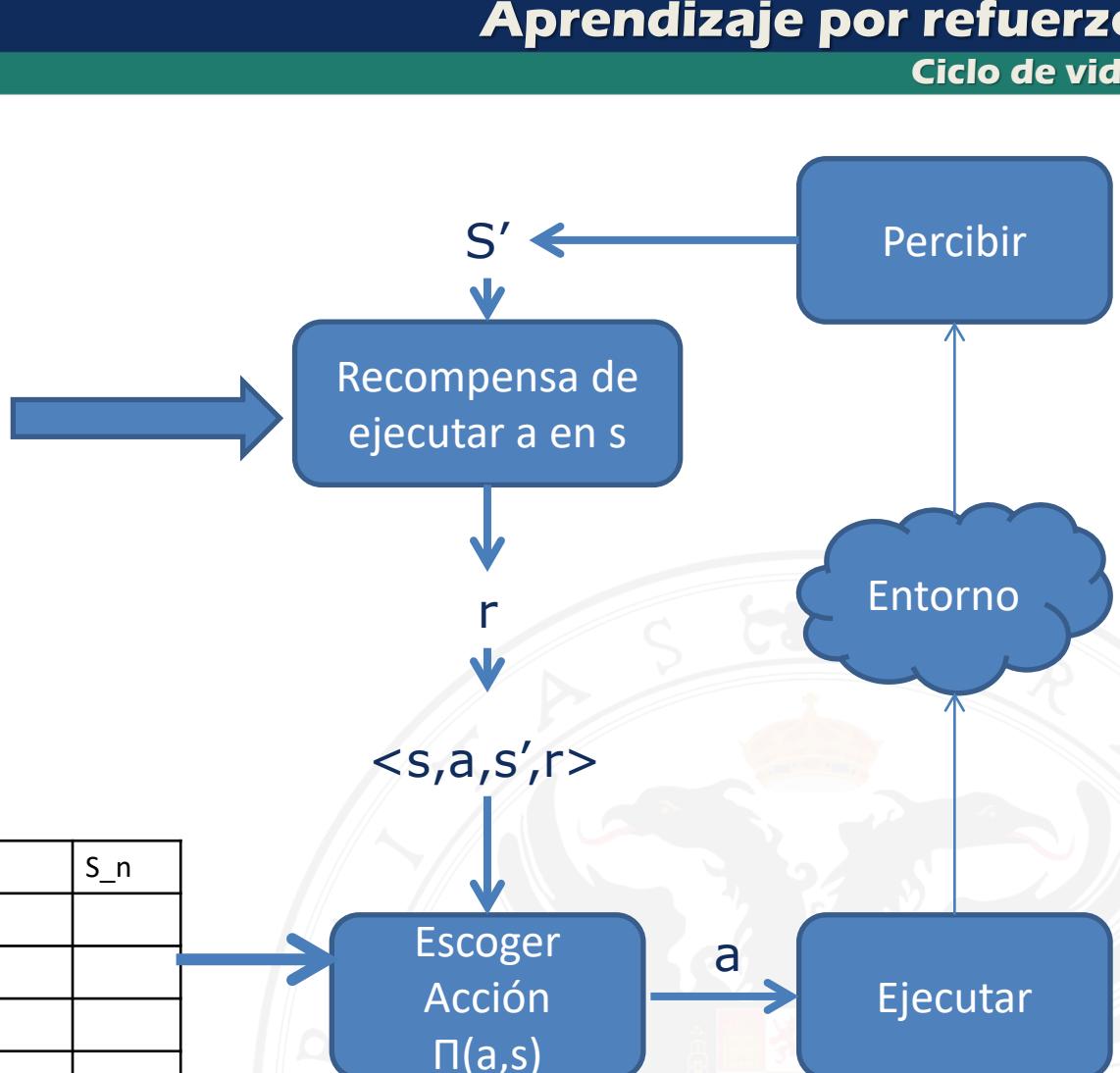
	$S_1$	...	$S$	...	$S'$	...	$S_n$
$A_1$							
...							
$A$			$Q(a,s)$				
...							
$A_n$							



-0.04	-0.04	-0.04	+1
-0.04		-0.04	-1
START	-0.04	-0.04	-0.04

Política  $\Pi(a,s)$

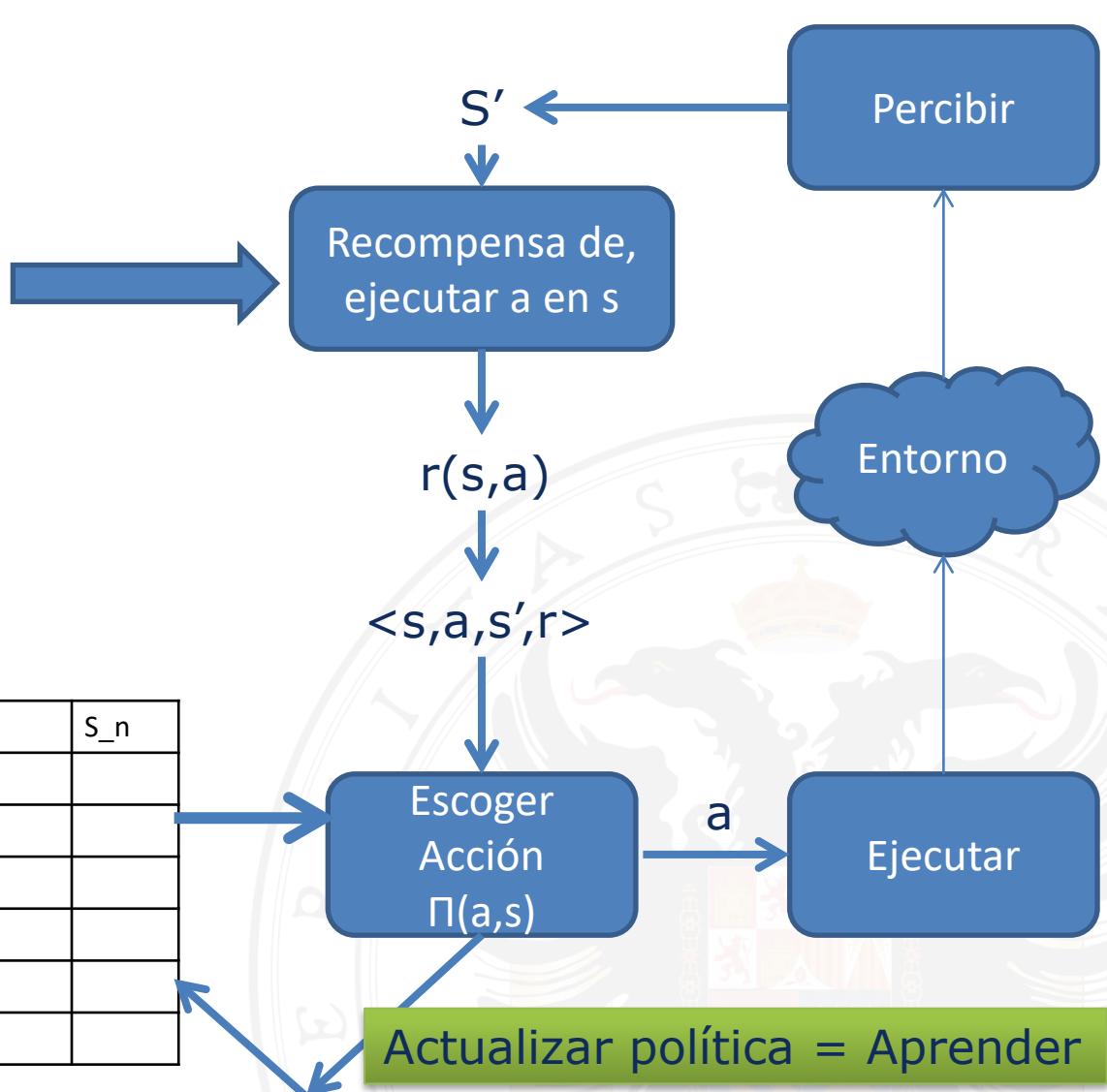
	S1	...	S	...	S'	...	S_n
A1							
...							
A			Q(a,s)				
...							
A_n							



-0.04	-0.04	-0.04	+1
-0.04		-0.04	-1
START	-0.04	-0.04	-0.04

Política  $\Pi(a,s)$

	S1	...	S	...	S'	...	S_n
A1					Q'1		
...					Q'2		
A		O			Q'n-1		
...					Q'n		
A_n							



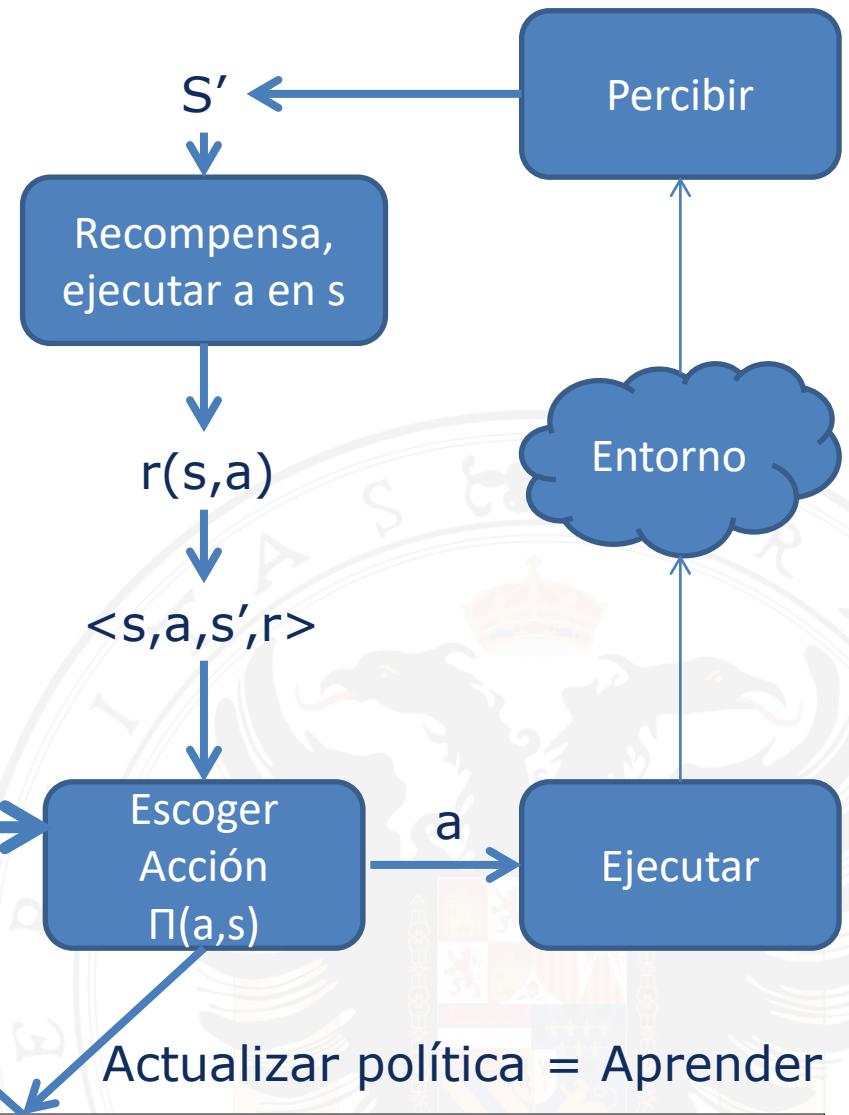
$$Q_{t+1}(s, a) := Q_t(s, a) + \alpha(r(s, a) + \gamma \max_{a'} Q(s', a') - Q_t(s, a))$$

### Alpha: Tasa de aprendizaje

- Alpha = 0, no aprende nada
  - Alpha = 1, sólo más reciente información
- Gamma: factor de descuento (importancia del futuro)
- Gamma = 0, greedy: solo considera recompensas locales
  - Gamma = 1, se esmera por recompensas a largo plazo

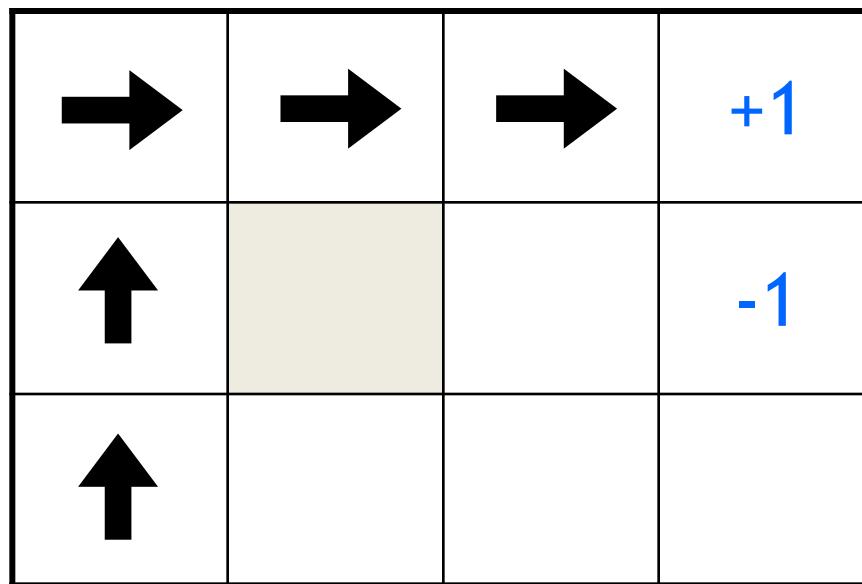
### Política $\Pi(a,s)$

	S1	...	S	...	S'	...	S_n
A1					Q'1		
...					Q'2		
A			O				
...					Q'n-1		
A_n					Q'n		

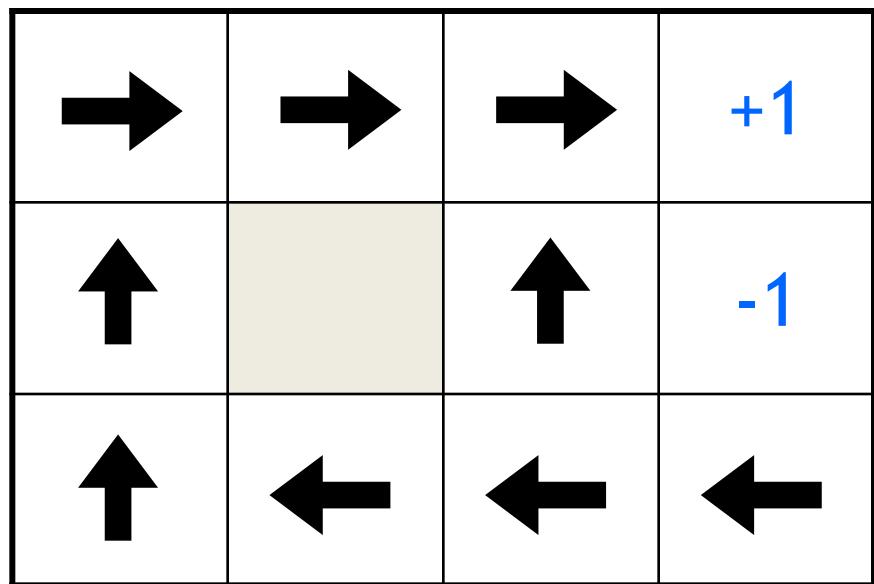


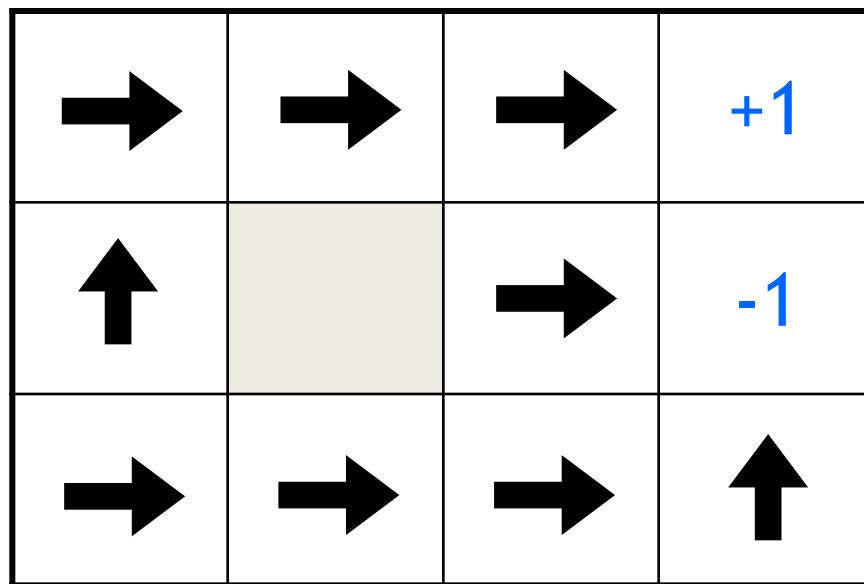
$$Q_{t+1}(s, a) := Q_t(s, a) + \alpha(r(s, a) + \gamma \max_{a'} Q(s', a') - Q_t(s, a))$$

(... de presentación de Peter Bodik, y las 6 siguientes)



- SOLO si las acciones son deterministas.
  - Que no es nuestro caso (acciones son estocásticas)
- IMPORTANTE: la solución es una **política**
  - Una función que hace corresponder a cada estado una acción





→	→	→	+1
↑		↑	-1
↑	→	↑	←

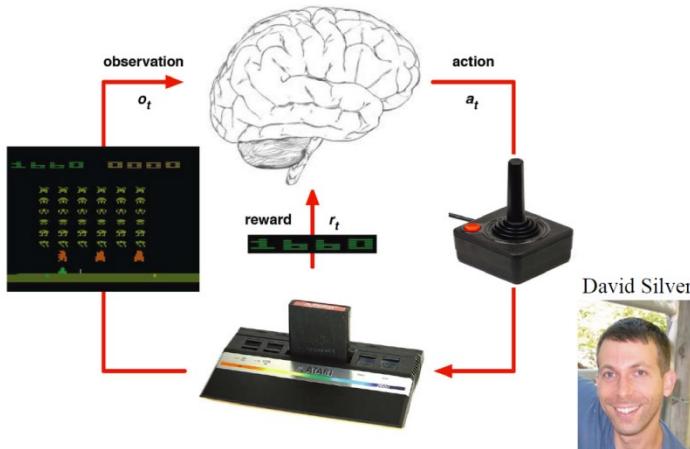
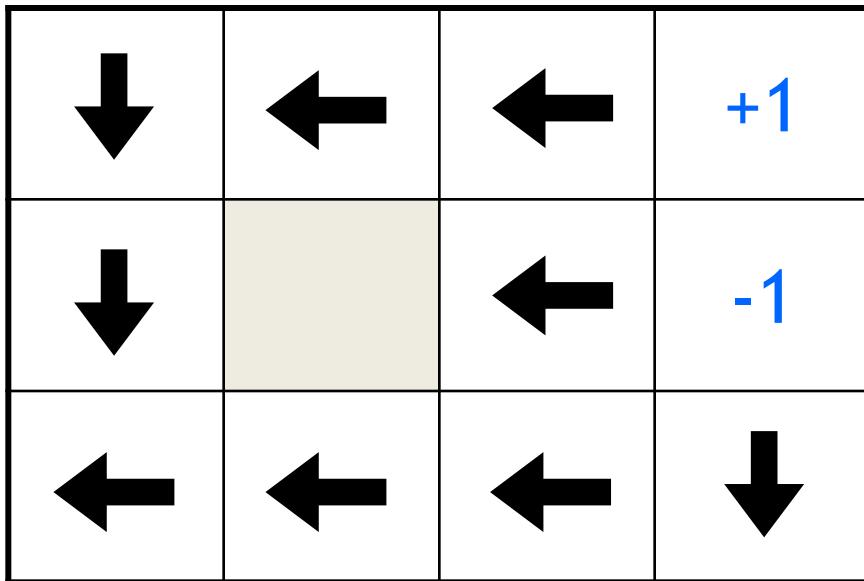


→	→	→	+1
↑		↑	-1
↑	←	←	←



→	→	→	+1
↑		←	-1
↑	←	←	↓





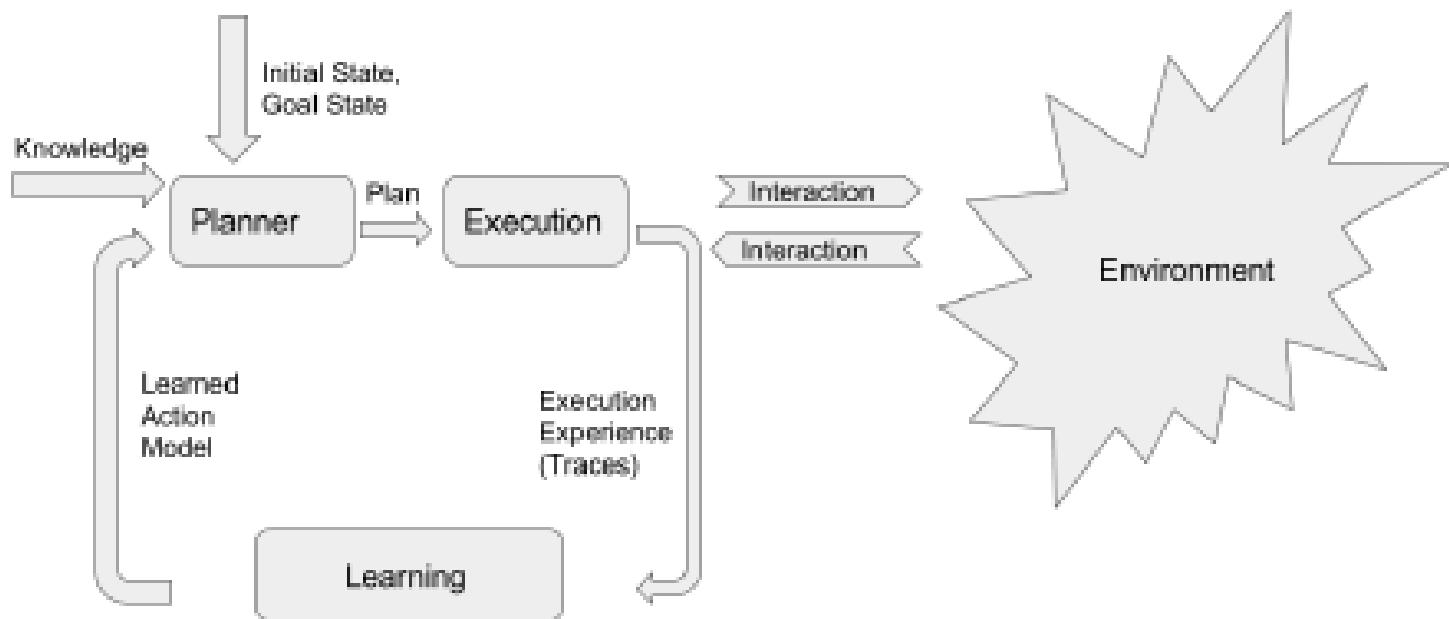
Un agente reactivo se preocupa de decidir qué acción ejecutar para un estado actual y extrae conclusiones de la observación del estado posterior.

Un agente deliberativo establece proyecciones de acciones (planes) para alcanzar un estado “más alejado en el tiempo”. Los planes incorporan trayectorias de estados, los planes pueden fallar, hay que plantearse qué hacer cuando hay un fallo,... ¿cómo afrontar la autonomía total de un agente deliberativo?

Un modelo prescriptivo puede hacerse a mano y usarlo para ayudar a tomar decisiones sobre recomendación de acciones. Pero es costoso diseñarlo y puede obtenerse a partir de trazas de planes, como en Process Mining clásico.

*Arora, Ankuj, Humbert Fiorino, Damien Pellier, Marc Métivier, y Sylvie Pesty. «A Review of Learning Planning Action Models». The Knowledge Engineering Review 33 (ed de 2018). <https://doi.org/10.1017/S0269888918000188>.*

	<b>Process Mining</b>	<b>Aprendizaje planificación</b>
Representación modelo	<ul style="list-style-type: none"> <li>Procesos rutinarios, estáticos.</li> <li>Un modelo de proceso clásico (Red de Petri, Diagrama BPMN,...)</li> </ul>	<ul style="list-style-type: none"> <li>Procesos dinámicos y orientados a objetivos.</li> <li>Dominio de planificación: basado en acciones o jerárquico</li> </ul>
Ejemplos de entrada	<ul style="list-style-type: none"> <li>Logs de eventos (conjuntos de secuencias de actividades)</li> </ul>	<ul style="list-style-type: none"> <li>Log (trazas) de planes</li> <li>Por cada traza: estado inicial, estado final</li> <li>Jerárquicos: tareas indicando a qué objetivo están destinadas.</li> </ul>
Modelo aprendido	<ul style="list-style-type: none"> <li>Un modelo de proceso con relaciones de orden/causales entre actividades y contemplando patrones de proceso comunes (control flow)</li> </ul>	<ul style="list-style-type: none"> <li>Un modelo de acciones basado en precondiciones y efectos de acciones</li> <li>Jerárquico: jerarquía de tareas y métodos, y precondiciones de los métodos</li> </ul>
Explotación del conocimiento aprendido	<ul style="list-style-type: none"> <li>Despliegue de procesos rutinarios que se ajusten mejor a los procedimientos de una organización (normativa, estándars, ....)</li> </ul>	<ul style="list-style-type: none"> <li>Agentes deliberativos autónomos (videojuegos, robots)</li> <li>Soporte a decisiones y trabajo del conocimiento.</li> </ul>



- Ejemplos de sistemas de aprendizaje en planificación:

- ARMS: aprendizaje de dominios basados en acciones

Q. Yang, K. Wu, y Y. Jiang, «Learning action models from plan examples using weighted MAX-SAT», *Artificial Intelligence*, vol. 171, n.º 2, pp. 107–143, 2007.

K. Wu, Q. Yang, y Y. Jiang, «Arms: an automatic knowledge engineering tool for learning action models for ai planning», *The Knowledge Engineering Review*, vol. 22, n.º 02, pp. 135–152, 2007.

- PlanMiner: aprendizaje de dominios basados en acciones mediante técnicas de clasificación y regresión

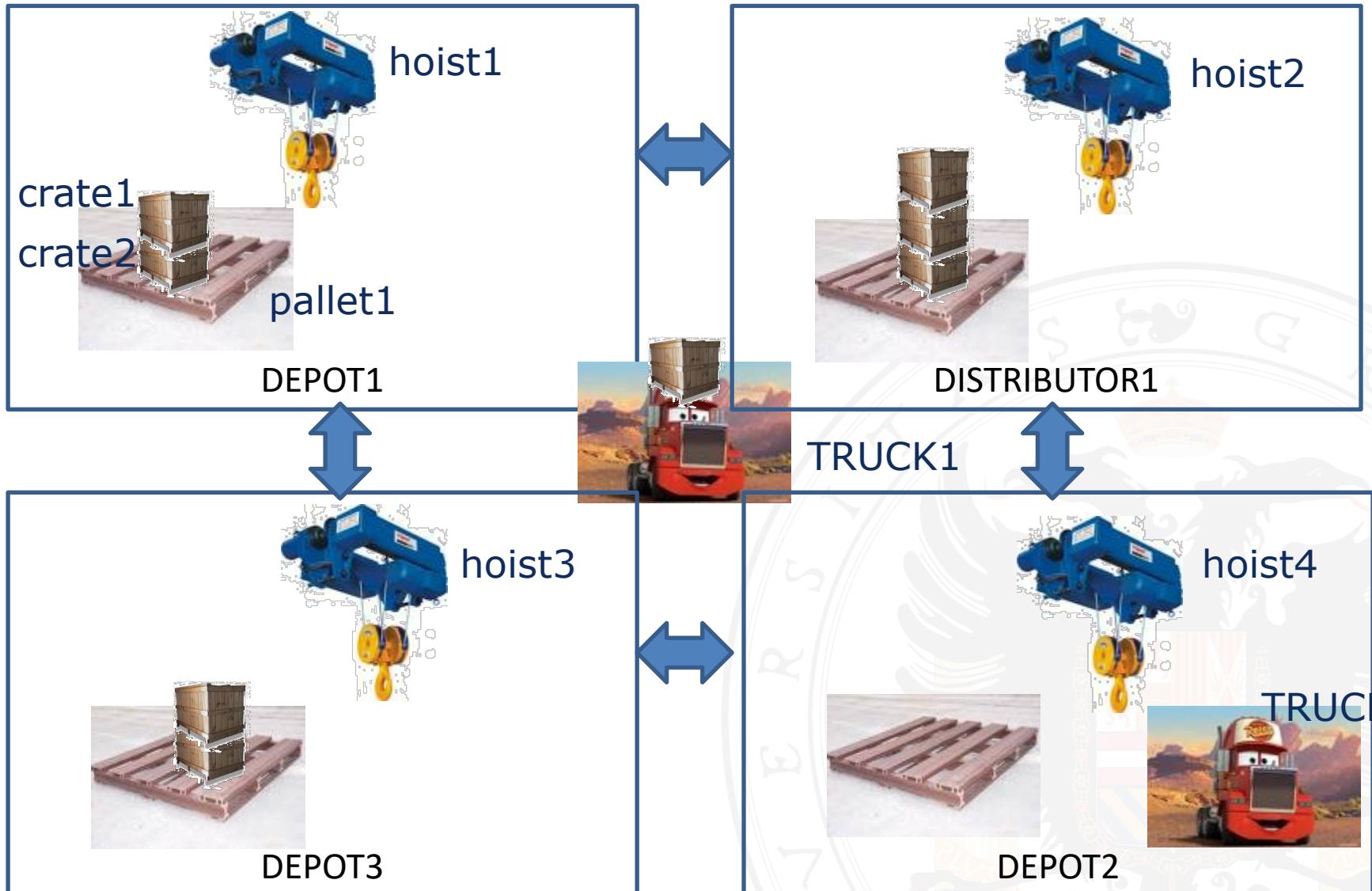
Segura-Muros, J. Á., Pérez, R., & Fernández-Olivares, J. (2021). Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence*, 51(11), 7973-7989.

- HTN-maker: aprendizaje de dominios jerárquicos.

C. Hogg, H. Muñoz-Avila, y U. Kuter, «Learning Hierarchical Task Models from Input Traces», *Computational Intelligence*, 2014.

Zhuo, Hankz Hankui, Héctor Muñoz-Avila, y Qiang Yang. «Learning hierarchical task network domains from partially observed plan traces». *Artificial Intelligence* 212 (2014): 134–157.

- Depots domain



- Objetos
  - Cajas y pallets, Almacenes y Distribuidores, Grúas y camiones
- Aspectos relevantes
  - Pilas de cajas en pallets, distribuidas por Almacenes y Distribuidores
  - Todos las ubicaciones están conectadas
    - Camiones ir directamente a cualquier sitio
    - Camiones pueden cargarse ilimitadamente de cajas
  - Grúas
    - Cargan y Descargan cajas de camiones
    - Cogen y Dejan (Apilan) cajas (sobre cajas o sobre pallets)
  - Pallets son como “mesas” del mundo de bloques
  - Camiones también como “mesas” que cambian de sitio

- **Conducir (Drive)** un camión de una ubicación origen a otra destino

(:action Drive

:parameters (?x - truck ?y - place ?z - place)

:precondition (and (at ?x ?y))

:effect (and (not (at ?x ?y)) (at ?x ?z)))

- **Coger (Lift)** una caja que puede estar en un pallet o apilada, con una grúa, estando todos en la misma ubicación

(:action Lift

:parameters (?x - hoist ?y - crate ?z - surface ?p - place)

:precondition (and (at ?x ?p) (available ?x) (at ?y ?p)

(on ?y ?z) (clear ?y))

:effect (and (not (at ?y ?p)) (lifting ?x ?y) (not (clear ?y))  
(not (available ?x))(clear ?z) (not (on ?y ?z))))

.....

A partir de trazas de ejemplos de resolución de problemas en este dominio, ¿cómo aprender las precondiciones y efectos de las acciones?.

- **Inputs: tipos de objetos del dominio, relaciones y cabeceras de acciones**

Table 1

Input domain description for Depot planning domain

Domain	Depot
types	place locatable - object depot distributor - place truck hoist surface - locatable pallet crate - surface
relations	(at ?x:locatable ?y:place) (on ?x:crate ?y:surface) (in ?x:crate ?y:truck) (lifting ?x:hoist ?y:crate) (available ?x:hoist) (clear ?x:surface)
actions	drive(?x:truck ?y:place ?z:place) lift(?x:hoist ?y:crate ?z:surface ?p:place) drop(?x:hoist ?y:crate ?z:surface ?p:place) load(?x:hoist ?y:crate ?z:truck ?p:place) unload(?x:hoist ?y:crate ?z:truck ?p:place)



- Inputs: trazas de planes, con estado inicial y objetivo, algún estado intermedio. Cada acción del plan tiene parámetros instanciados**

Table 2

Three plan traces as part of the training examples

	Plan1	Plan2	Plan3
Initial	$I_1$	$I_2$	$I_3$
Step1	lift(h1 c0 p1 ds0), drive(t0 dp0 ds0)	lift(h1 c1 c0 ds0)	lift(h2 c1 c0 ds0)
State		(lifting h1 c1)	
Step2	load(h1 c0 t0 ds0)	load(h1 c1 t0 ds0)	load(h2 c1 t1 ds0)
Step3	drive(t0 ds0 dp0)	lift(h1 c0 p1 ds0)	lift(h2 c0 p2 ds0), drive(t1 ds0 dp1)
State	(available h1)		
Step4	unload(h0 c0 t0 dp0)	load(h1 c0 t0 ds0)	unload(h1 c1 t1 dp1), load(h2 c0 t0 ds0)
State	(lifting h0 c0)		
Step5	drop (h0 c0 p0 dp0)	drive(t0 ds0 dp0)	drop(h1 c1 p1 dp1), drive(t0 ds0 dp0)
Step6		unload(h0 c1 t0 dp0)	unload(h0 c0 t0 dp0)
Step7		drop(h0 c1 p0 dp0)	drop(h0 c0 p0 dp0)
Step8		unload(h0 c0 t0 dp0)	
Step9		drop(h0 c0 c1 dp0)	
Goal	(on c0 p0)	(on c1 p0) (on c0 c1)	(on c0 p0) (on c1 p1)

$I_1$ : (at p0 dp0), (clear p0), (available h0), (at h0 dp0), (at t0 dp0), (at p1 ds0), (clear c0), (on c0 p1), (available h1), (at h1 ds0).

$I_2$ : (at p0 dp0), (clear p0), (available h0), (at h0 dp0), (at t0 ds0), (at p1 ds0), (clear c1), (on c1 c0), (on c0 p1), (available h1), (at h1 ds0).

$I_3$ : (at p0 dp0), (clear p0), (available h0), (at h0 dp0), (at p1 dp1), (clear p1), (available h1), (at h1 dp1), (at p2 ds0), (clear c1), (on c1 c0), (on c0 p2), (available h2), (at h2 ds0), (at t0 ds0), (at t1 ds0).

## Log de planes



ARMS: a partir de logs de planes aprende dominios de planificación basados en acciones.

Un agente deliberativo autónomo puede aprender a desarrollar su comportamiento mediante ejemplos de planes de actuación.

Precondiciones y  
Efectos de las  
acciones.

## 1. Obtener el grafo causal

1. Parecido al visto en el Tema 1, pero con distintas relaciones y medidas de causalidad.
2. **Un par de acciones está conectado causalmente si algunos de sus tipos de parámetros coinciden.**
  1. Observar que dos acciones pueden aparecer ordenadas en el plan, pero no tienen por qué estar conectadas.
  3. Cada par conectado ( $a_i, a_j$ ) del grafo causal se etiqueta con
    1. Soporte( $a_i, a_j$ )=Número de veces que aparece el par
    2. Tasa Soporte( $a_i, a_j$ )= Soporte( $a_i, a_j$ )/Número total de pares de acciones
  4. Solo considerar las relaciones causales que cumplan
    1. Tasa\_Soporte > Umbral

## 2. Idea principal

1. Si un par ( $a_1, a_2$ ) aparece en el grafo causal es porque tiene que existir un predicado “conector” para  $a_1 \rightarrow p \rightarrow a_2$
2. Además,  $p$  tiene que estar en la lista de adición de  $a_1$  y en las precondiciones de  $a_2$ .
3. Los predicados así obtenidos y asignados tienen que respetar las reglas de escritura de dominios basados en operadores y tienen que garantizar que los planes son correctos.

1. ¿Cómo determinar ese predicado conector?
  1. Mediante búsqueda (prueba y error) entre los predicados dados como entrada.
    1. Hay maneras de limitar el número de candidatos, dependiendo de las acciones.
    2. Problema de optimización de restricciones booleanas. MAX-SAT
2. Restricciones
  1. Siguiente transparencia (solo para hacerse una idea)
3. La solución al problema de satisfacción de restricciones es una asignación de predicados (relaciones) a listas de precondiciones y efectos de las acciones.

(1) (Constraint A.1) The intersection of the precondition and add lists of all actions must be empty.

$$pre_i \cap add_i = \emptyset.$$

(2) (Constraint A.2) In addition, if an action's delete list includes a relation, this relation is in the action's precondition list. Thus, for every action, we require that the delete list is a subset of the precondition list.

$$del_i \subseteq pre_i.$$

- (Constraint I.1) The relation  $p$  must be generated by an action  $a_{ik}$  ( $0 \leq i_k \leq n$ ), that is,  $p$  is selected to be in the add-list of  $a_{ik}$ .  $p \in (add_{i_1} \cup add_{i_2} \cup \dots \cup add_{i_k})$ , where  $\cup$  means logical “or”.
- (Constraint I.2) The last action  $a_{ik}$  must not delete the relation  $p$ ; that is,  $p$  must not be selected to be in the delete list of  $a_{ik}$ :  $p \notin del_{ik}$ .
- (Constraint I.3) We define the weight value of a relation-action pair  $(p, a)$  as the occurrence probability of this pair in all plan examples. If the probability of a relation-action pair is higher than the probability threshold  $\theta$ , then we set a corresponding relation constraint  $p \in PRECOND_a$ , which receives a weight value equal to its prior probability.
- (Constraint P.1) Every precondition  $p$  of every action  $b$  must be in the add list of a preceding action  $a$  and is not deleted by any actions between  $a$  and  $b$ .
- (Constraint P.2) In addition, at least one relation  $r$  in the add list of an action must be useful in achieving a precondition of a later action. That is, for every action  $a$ , an add list relation  $r$  must be in the precondition of a later action  $b$ , and there is no other action between  $a$  and  $b$  that either adds or deletes  $r$ .
- (Plan constraint P.3) One of the relevant relations  $p$  must be chosen to be in the preconditions of both  $a_i$  and  $a_j$ , but not in the delete list of  $a_i$ ,

$$\exists p (p \in (pre_i \cap pre_j) \wedge p \notin (del_i))$$

- (Constraint P.4) The first action  $a_i$  adds a relevant relation that is in the precondition list of the second action  $a_j$  in the pair,

$$\exists p (p \in (add_i \cap pre_j))$$

- (Constraint P.5) A relevant relation  $p$  that is deleted by the first action  $a_i$  is added by  $a_j$ . The second clause is designed for the event when an action re-establishes a fact that is deleted by a previous action.

$$\exists p (p \in (del_i \cap add_j))$$

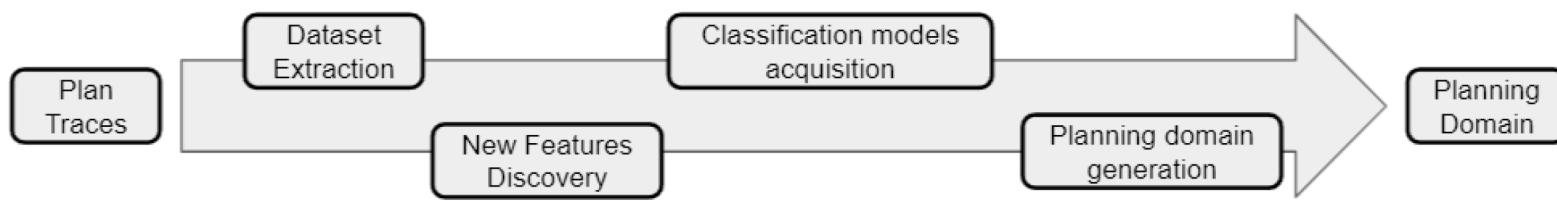


Figure 3.1: The learning pipeline of PlanMiner

# Generación de trazas de ejecución.

**Table 1 Extract of a plan trace**

From: [Discovering relational and numerical expressions from plan traces for learning action models](#)

(a) Plan	
Start	End Action
0	1 ( <i>goto rov1 wp1 wp2</i> )
1	2 ( <i>goto rov1 wp2 wp3</i> )
(b) States list	
Index	Predicates
0	( <i>at rov1 wp1</i> ) $\wedge$ ( $\neg$ ( <i>at rov1 wp2</i> )) $\wedge$ ( $\neg$ ( <i>at rov1 wp3</i> )) $\wedge$ $\neg$ ( <i>scanned wp3</i> ) $\wedge$ (= ( <i>bat_usage rov1</i> ) 3) $\wedge$ (= ( <i>energy rov1</i> ) 450) $\wedge$ $=$ ( <i>dist wp1 wp2</i> ) 50 $\wedge$ (= ( <i>dist wp2 wp3</i> ) 80)
1	$\neg$ ( <i>at rov1 wp1</i> ) $\wedge$ ( <i>at rov1 wp2</i> ) $\wedge$ ( $\neg$ ( <i>at rov1 wp3</i> )) $\wedge$ $\neg$ ( <i>scanned wp3</i> ) $\wedge$ (= ( <i>bat_usage rov1</i> ) 3) $\wedge$ (= ( <i>energy rov1</i> ) 300) $\wedge$ $=$ ( <i>dist wp1 wp2</i> ) 50 $\wedge$ (= ( <i>dist wp2 wp3</i> ) 80)
2	( <i>at rov1 wp3</i> ) $\wedge$ ( $\neg$ ( <i>at rov1 wp1</i> )) $\wedge$ ( $\neg$ ( <i>at rov1 wp2</i> )) $\wedge$ $\neg$ ( <i>scanned wp3</i> ) $\wedge$ (= ( <i>bat_usage rov1</i> ) 3) $\wedge$ (= ( <i>energy rov1</i> ) 60) $\wedge$ $=$ ( <i>dist wp1 wp2</i> ) 50 $\wedge$ (= ( <i>dist wp2 wp3</i> ) 80)

The trace is an extract of a solution plan for a specific problem in the Rovers domain. In this plan, a rover (rov1) moves from wp1 to wp3, traversing wp2. Subtable **(a)** displays the plan (sequence of actions) executed, Start and End columns display the index of associated states before and after applying a given action. Subtable **(b)** shows the set of intermediate states associated with each action of the plan during the execution. Index references a state in the Start/End columns of Subtable **(a)**. Note that the same state may appear fulfilling different roles for different actions (pre-state or post-state)

# Preprocesamiento para generar los datasets

Fig. 1

From: [Discovering relational and numerical expressions from plan traces for learning action models](#)

Action:  $(\text{goto } rov1 \text{ wp1 wp2})$

- **pre-state:**  $(\text{at } rov1 \text{ wp1}) \wedge (\neg (\text{at } rov1 \text{ wp2})) \wedge$   
 $(\neg (\text{at } rov1 \text{ wp3})) \wedge (\neg (\text{scanned } wp3)) \wedge$   
 $(= (\text{bat\_usage } rov1) 3) \wedge (= (\text{energy } rov1) 450) \wedge$   
 $(= (\text{dist } wp1 \text{ wp2}) 50) \wedge (= (\text{dist } wp2 \text{ wp3}) 80)$
- **post-state:**  $(\neg (\text{at } rov1 \text{ wp1})) \wedge (\text{at } rov1 \text{ wp2}) \wedge$   
 $(\neg (\text{at } rov1 \text{ wp3})) \wedge (\neg (\text{scanned } wp3)) \wedge$   
 $(= (\text{bat\_usage } rov1) 3) \wedge (= (\text{energy } rov1) 300) \wedge$   
 $(= (\text{dist } wp1 \text{ wp2}) 50) \wedge (= (\text{dist } wp2 \text{ wp3}) 80)$

State transition of the  $(\text{goto } rov1 \text{ wp1 wp2})$  extracted from Table 1 plan trace

Fig. 2

From: [Discovering relational and numerical expressions from plan traces for learning action models](#)

Action:  $(\text{goto } ?\text{arg1} ?\text{arg2} ?\text{arg3})$

- **pre-state:**  $(\text{at } ?\text{arg1} ?\text{arg2}) \wedge (\neg (\text{at } ?\text{arg1} ?\text{arg3})) \wedge$   
 $\underline{(\neg (\text{at } ?\text{arg1} wp3))} \wedge \underline{(\neg (\text{scanned } wp3))} \wedge$   
 $\underline{(= (\text{bat\_usage } ?\text{arg1}) 3)} \wedge (= (\text{energy } ?\text{arg1}) 450) \wedge$   
 $\underline{(= (\text{dist } ?\text{arg2} ?\text{arg3}) 50)} \wedge (= (\text{dist } ?\text{arg3} wp3) 80)$
- **post-state:**  $(\neg (\text{at } ?\text{arg1} ?\text{arg2})) \wedge (\text{at } ?\text{arg1} ?\text{arg3}) \wedge$   
 $\underline{(\neg (\text{at } ?\text{arg1} wp3))} \wedge \underline{(\neg (\text{scanned } wp3))} \wedge$   
 $\underline{(= (\text{bat\_usage } ?\text{arg1}) 3)} \wedge (= (\text{energy } ?\text{arg1}) 300) \wedge$   
 $\underline{(= (\text{dist } ?\text{arg2} ?\text{arg3}) 50)} \wedge (= (\text{dist } ?\text{arg3} wp3) 80)$

Schema form of a  $(\text{goto } ?\text{arg1} ?\text{arg2} ?\text{arg3})$  action before erasing irrelevant predicates (underlined)

Aprender modelos de acciones como un problema de clasificación.  
Asociar un data set por cada esquema de acción.

## Table 2 Dataset associated with the $(goto ?arg1 ?arg2 ?arg3)$ action

From: [Discovering relational and numerical expressions from plan traces for learning action models](#)

(at ?arg1 ?arg2)	(at ?arg1 ?arg3)	(bat_usage ?arg1)	(energy ?arg1)	(dist ?arg2 ?arg3)	(scanned ?arg3)	Class
True	False	3	450	50	MV	pre - state
False	True	3	300	50	MV	post - state
True	False	3	300	80	False	pre - state
False	True	3	60	80	False	post - state

Table shows how the state transitions of the  $(goto ?arg1 ?arg2 ?arg3)$  actions defined in Table 1 are displayed as an attribute-value matrix. Each state in the state transitions is included as an instance in the dataset, where its predicates are displayed as attributes, and the class labels define its role in the state transition. Note that state 1 (Table 1) appears twice in the dataset (instances 2 and 3) with different values and different class label

- Aprender dominios jerárquicos



```
(:task Tstack
:parameters (?x ?y - block)
(:method clear
:precondition (clear ?y)
:tasks ((Tpickup ?x) (stack ?x ?y)))
(:method not_clear
:precondition (on ?z ?y)
:tasks ((Tunstack ?z ?y)
(Tputdown ?z ?y))))
```

```
(:task Tunstack
:parameters (?x ?y - block)
(:method clear
:precondition (clear ?x)
:tasks (unstack ?x ?y))
(:method not_clear
:precondition (on ?z ?x)
:tasks ((Tunstack ?z ?x) (Tputdown ?z) (unstack ?x
?y))))
```

```
(:action pickup
:parameters (?x - block)
:precondition (and (ontable ?x)(clear ?x)(handempty))
:effect (and (not (ontable ?x)) (not (clear ?x))(not (handempty)) (holding ?x)))
```

```
(:action putdown
:parameters (?x - block)
:precondition (holding ?x)
:effect (and (ontable ?x) (clear ?x) (handempty) (not (holding ?x))))
```

```
(:task Tpickup
:parameters (?x - block)
(:method clear
:precondition (and (clear ?x) (ontable ?x))
:tasks (pickup ?x))
(:method clear2
:precondition (and (clear ?x) (on ?x ?z))
:tasks ((Tputdown ?z ?x)
(Tunstack ?z ?x))))
```

```
(:task Tputdown
:parameters (?x - block)
(:method clear
:precondition (holding ?x)
:tasks (putdown ?x))
(:method ocupado
:precondition ()
:tasks ((Tpickup ?x) (putdown ?x))))
```

## ¿Cómo aprender esto?

## ¿Conocido esto?

```
(:action stack
:parameters (?x ?y - block)
:precondition (and (holding ?x)(clear ?y))
:effect (and (not (holding ?x)) (not (clear ?y)) (clear ?x)(on ?x ?y) (handempty)))

(:action unstack
:parameters (?x ?y - block)
:precondition (and (handempty) (clear ?x)(on ?x ?y))
:effect (and (holding ?x)(clear ?y)(not (clear ?x)) (not (on ?x ?y)) (not (handempty))))
```

Dominio de acciones

Conjunto de trazas de planes  
con todos los estados  
intermedios.

Tareas anotadas con pre y  
post-condiciones

Conjunto de métodos  
preexistente (para que sea  
incremental)



HTN-maker



Un conjunto de  
métodos  
asociados a las  
tareas de entrada.

## Dominio de acciones

Conjunto de trazas de planes con todos los estados intermedios.

Tareas anotadas con pre y post-condiciones

Conjunto de métodos preexistente (para que sea incremental)

```
(:action pickup
:parameters (?x - block)
:precondition (and (ontable ?x) (clear ?x) (handempty))
:effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))

(:action putdown
:parameters (?x - block)
:precondition (holding ?x)
:effect (and (ontable ?x) (clear ?x) (handempty) (not (holding ?x)))))

(:action stack
:parameters (?x ?y - block)
:precondition (and (holding ?x) (clear ?y))
:effect (and (not (holding ?x)) (not (clear ?y)) (clear ?x) (on ?x ?y) (handempty)))

(:action unstack
:parameters (?x ?y - block)
:precondition (and (handempty) (clear ?x) (on ?x ?y))
:effect (and (holding ?x) (clear ?y) (not (clear ?x)) (not (on ?x ?y)) (not (handempty)))
```

## Dominio de acciones

Conjunto de trazas de planes  
con todos los estados  
intermedios.

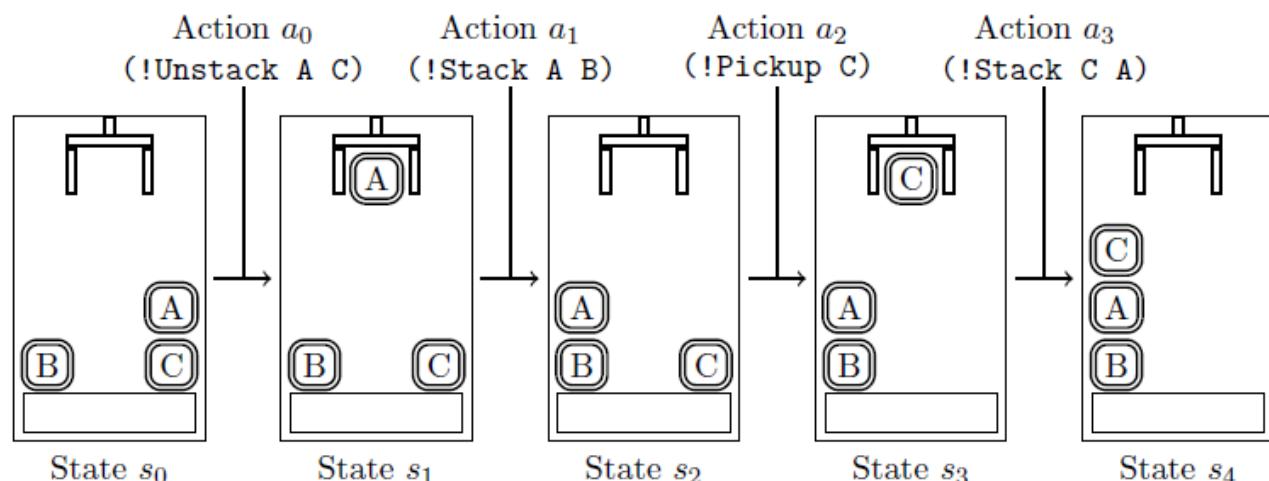


Figure 6: An example plan in the BLOCKS-WORLD domain.

```
( :task Make-2Pile  
  :parameters ( ?a ?b )  
  :precondition  
  ( and )  
  :postcondition  
  ( and (on-table ?b) (on ?a ?b) (clear ?a) ) )
```

Figure 5: An example annotated task in the BLOCKS-WORLD domain.

Tareas anotadas con pre y post-condiciones

Conjunto de métodos preexistente (para que sea incremental)

Ojo que no son acciones, son tareas de alto nivel para las que se especifican condiciones que deben cumplirse antes de ejecutar y una vez ejecutada la tarea.

En el ejemplo vamos a asumir que como entrada tiene tareas para conseguir pilas de 1, 2 o 3 bloques.

Son necesarias para guiar el proceso de aprendizaje.

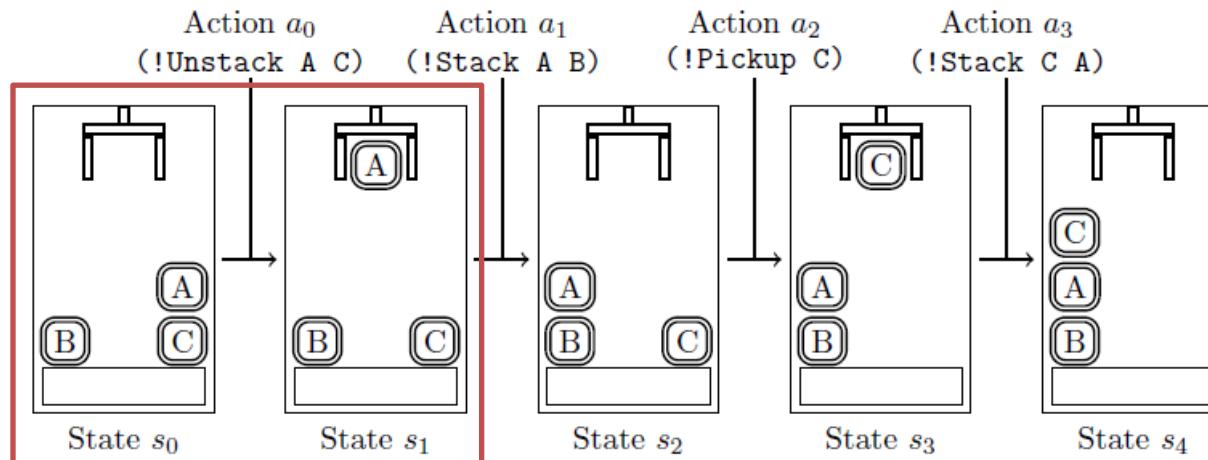
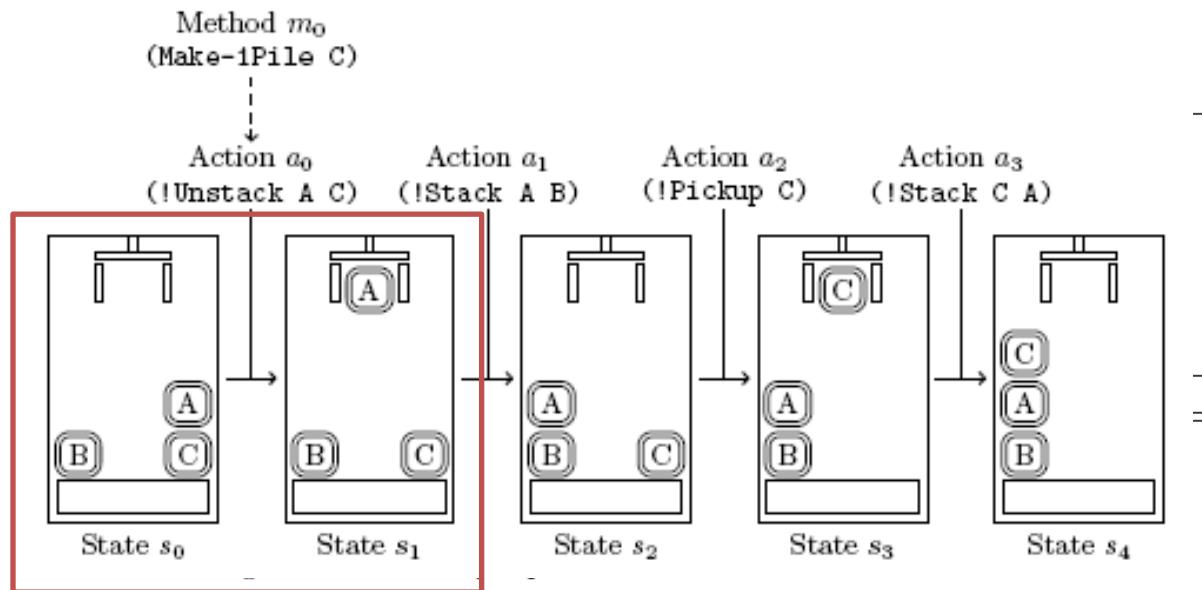
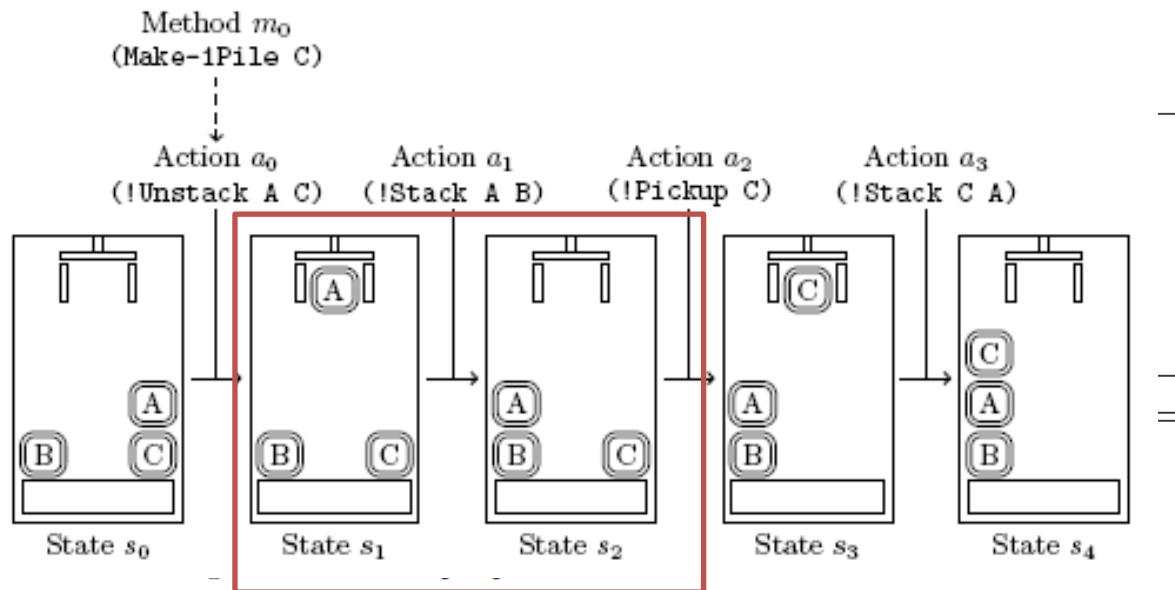


Figure 6: An example plan in the BLOCKS-WORLD domain.

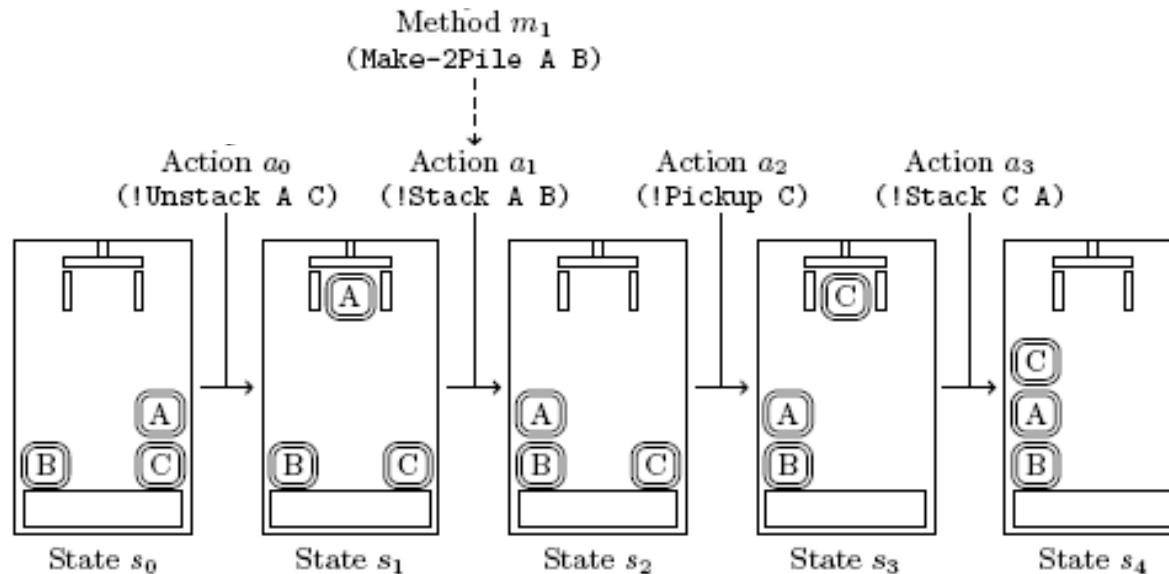
- Considera el subplan contenido sólo < $a_0$ >.
- ¿Hay alguna tarea anotada cuyas precondiciones son satisfechas en  $s_0$  y cuyas postcondiciones son satisfechas en  $s_1$  pero no en  $s_0$ ?
- Sí la hay.



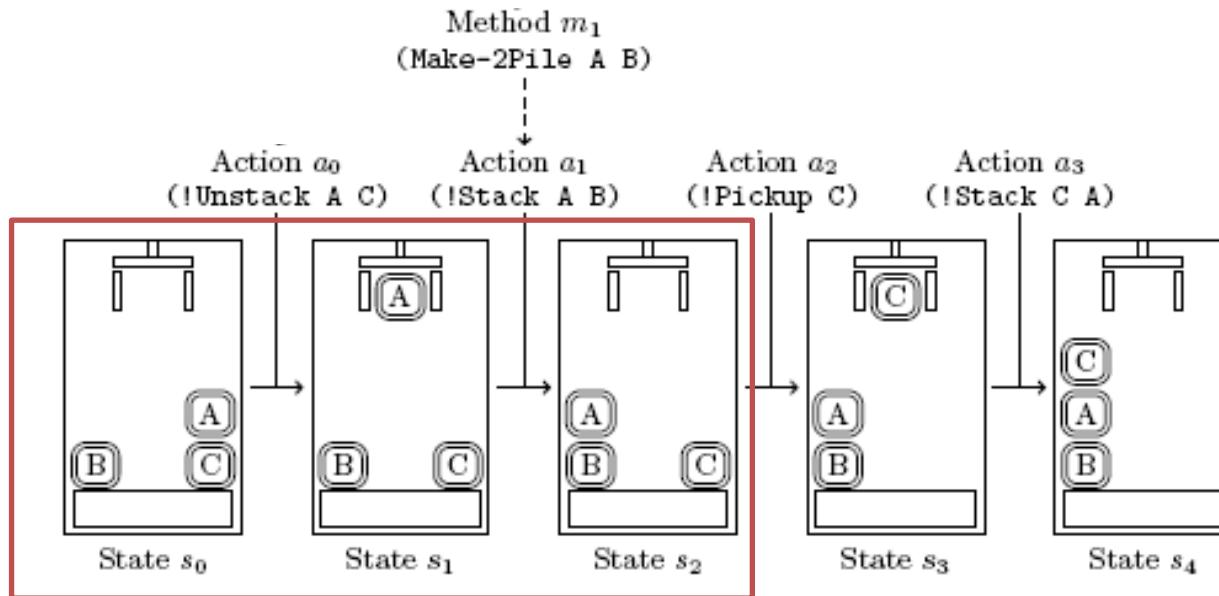
- Considera el subplan contenido sólo  $\langle a_0 \rangle$ .
- ¿Hay alguna tarea anotada cuyas precondiciones son satisfechas en  $s_0$  y cuyas postcondiciones son satisfechas en  $s_1$  pero no en  $s_0$ ?
- Sí la hay (asumir que se ha dado como entrada).
- HTN-maker aprende el método  $m_0$  para explicar cómo se puede hacer una pila de tamaño 1.
  - $m_0$  tiene una sola tarea ( $a_0$ ) y como precondiciones las propias de la acción  $a_0$ .



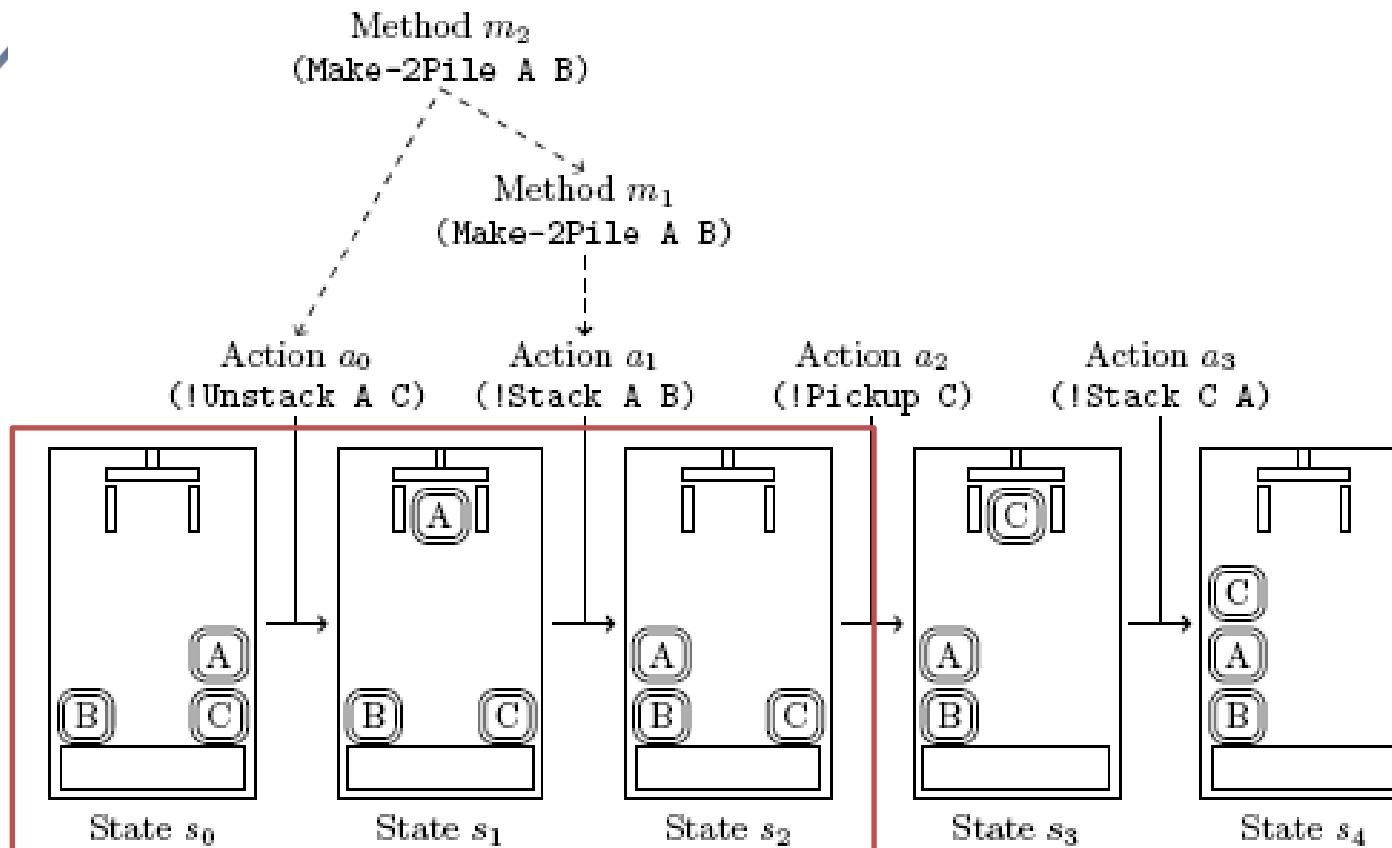
- Considera el subplan contenido sólo  $\langle a_1 \rangle$ .
- Detecta que  $s_2$  empareja con las postcondiciones de  $(make\_2Pile A B)$



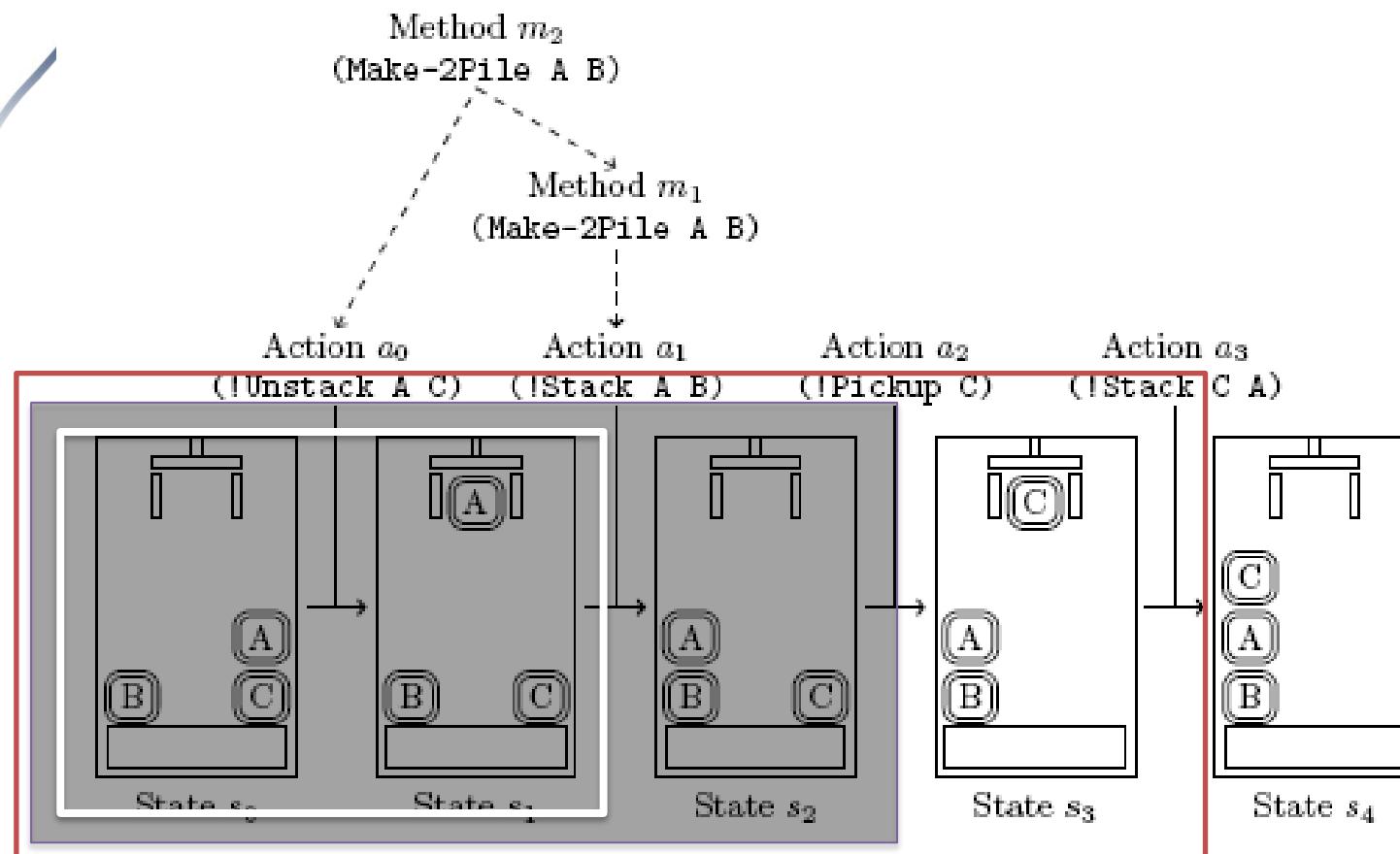
- Considera el subplan contenido sólo < $a_1$ >.
- Detecta que  $s_2$  empareja con las postcondiciones de (make\_2Pile A B)
- Aprende otro nuevo método  $m_1$



- Considera el subplan contenido sólo  $\langle a_0, a_1 \rangle$ .
- Detecta que  $s_2$  empareja con las postcondiciones de  $(make\_2Pile A B)$



- Considera el subplan contenido sólo  $\langle a_0, a_1 \rangle$ .
- Detecta que  $s_2$  empareja con las postcondiciones de  $(make\_2Pile A B)$
- Aprende otro nuevo método  $m_2$  y observar que es recursivo.



- Luego considera  $\langle a_2 \rangle$ ,  $\langle a_1, a_2 \rangle$ ,  $\langle a_0, a_1, a_2 \rangle$ , pero no encuentra ninguna tarea de las anotadas que pueda encajar con esos subplanes.

Method  $m_6$   
 (Make-3Pile C A B)

Method  $m_5$   
 (Make-3Pile C A B)

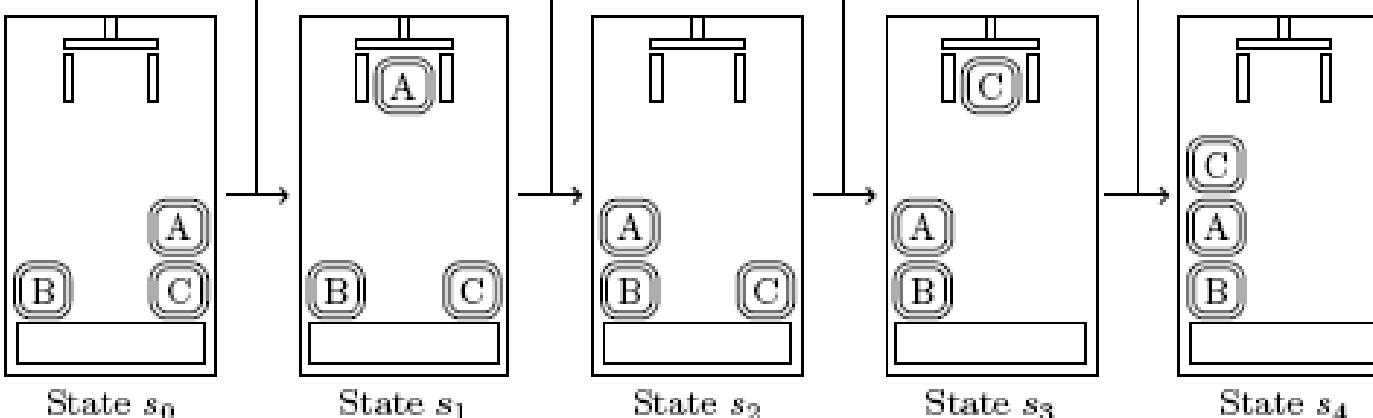
Method  $m_4$   
 (Make-3Pile C A B)

Method  $m_3$   
 (Make-3Pile C A B)

Action  $a_0$   
 (!Unstack A C)  
 Action  $a_1$   
 (!Stack A B)  
 Action  $a_2$   
 (!Pickup C)  
 Action  $a_3$   
 (!Stack C A)

A3  
 A2 a3  
 A1 a2 a3  
 A0 a1 a2 a3

Finalmente para estos otros subplanes encuentra otro conjunto de métodos en un proceso análogo al anterior.



- HTN maker repite este proceso para cada traza.
- Aplica posteriormente un proceso de generalización para convertir objetos instanciados en variables y reducir repeticiones.
- Obtiene como resultado un dominio de planificación jerárquico.
- Los dominios jerárquicos obtenidos no se parecen a los que crearía un humano, pero permiten establecer estrategias jerárquicas automáticamente para resolver problemas.