

Inteligencia Artificial

Agentes Lógicos



Universidad
Rey Juan Carlos

Esta presentación referencia el capítulo 7 del libro de texto: “Artificial Intelligence a Modern Approach”. Stuart Russel, Peter Norving.

Se diseñarán Agentes que pueden:

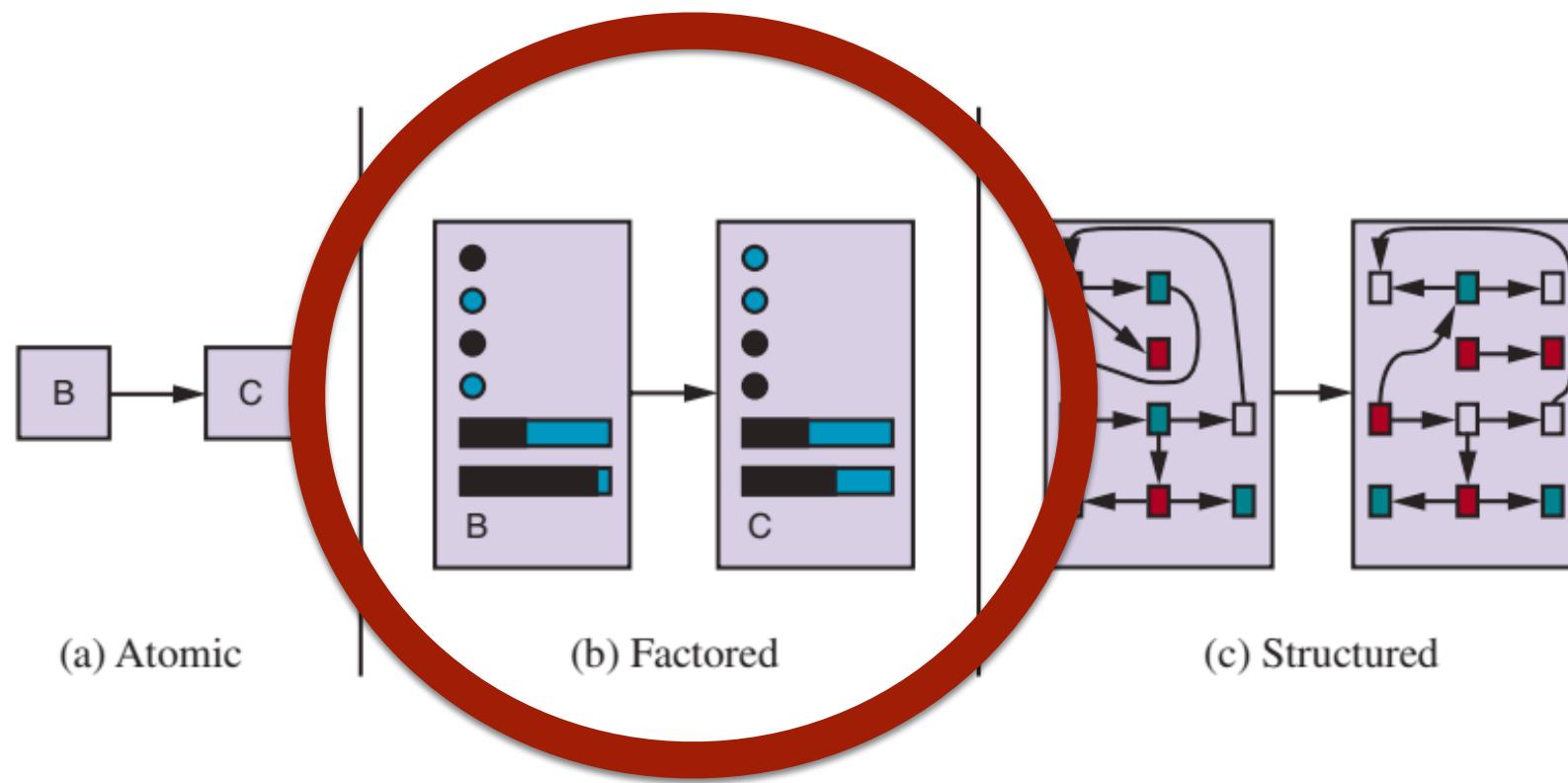
- Formar representaciones de un mundo complejo.
- Utilizar un proceso de inferencia para derivar nuevas representaciones sobre el mundo.
- Utilizar estas nuevas representaciones para deducir qué hacer.

Agentes basados en conocimiento:

- Los humanos razonamos sobre representaciones internas del conocimiento para hacer cosas.
- En IA, esta aproximación se materializa en **agentes basados en conocimiento (Knowledge-based agents)** que utilizan un proceso de razonamiento sobre una representación interna del conocimiento para decidir sobre los próximos pasos a dar
- Los **agentes de resolución de problemas** vistos hasta ahora tenían un **conocimiento muy limitado** :
 - Conocen :
 - Acciones disponibles
 - El resultado de las acciones
 - No conocen:
 - Hechos generales del mundo. Por ejemplo que en el 8-puzzle no puede haber dos números en la misma posición, etc. Que en el caso de buscador de rutas que no puede haber distancias negativas, etc.

¿Cómo representa el agente el estado del entorno?

- Representación de los estados
 - Atómica: Los estados son indivisibles y no tienen estructura interna (Search, HMM, MDP)
 - Factorizado: Cada estado tienen variables o atributos (Planning, Lógica proposicional, Machine Learning, BN)
 $\hookrightarrow a \leftarrow b, b \leftarrow c \rightarrow a \leftarrow c$
 - Estructurado: Cada estado describe objetos y las relaciones entre ellos (FOL, lenguaje natural)



En este tema desarrollamos la lógica como una clase general de representaciones para apoyar a los agentes basados en el conocimiento.

- **Los agentes basados en el conocimiento:**

1. pueden **aceptar nuevas tareas** en forma de metas descritas explícitamente
2. pueden ser **competentes rápidamente** si se les dice o aprenden nuevos conocimientos sobre el entorno
3. pueden **adaptarse a los cambios del entorno** actualizando los conocimientos pertinentes

- **Base de conocimiento (KB - knowledge base)**: El componente central
- **KB formado por sentencias** escritas en un **lenguaje de representación del conocimiento** y representan aserciones sobre el mundo.
- **Axiomas** : Cuando las sentencias no son derivadas de otras.
- Se pueden añadir (**TELL**) or preguntar (**ASK**) sentencias a la **KB**:
 - **TELL** : Añade nuevas sentencias a la KB
 - **ASK** : pregunta sobre lo que es conocido por la KB
- Las respuestas dadas en el proceso de ASK deben seguirse necesariamente de lo que se le ha dicho al sistema “TELL” no tiene que inventarlas.
- Ambas operaciones pueden incluir **Inferencias**- derivar nuevas sentencias de las ya existentes
- El Agente puede tener un conocimiento previo del mundo (**Background knowledge**)

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
            t, a counter, initially 0, indicating time
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Puede contener conocimiento previo del mundo

Construye una sentencia afirmando que el agente
Tuvo una percepción en un momento determinado

Construye sentencia para
Preguntar sobre la acción a tomar en el tiempo t

Construye una sentencia que asevera que la acción
Fue ejecutada

Agentes Basados en Conocimiento

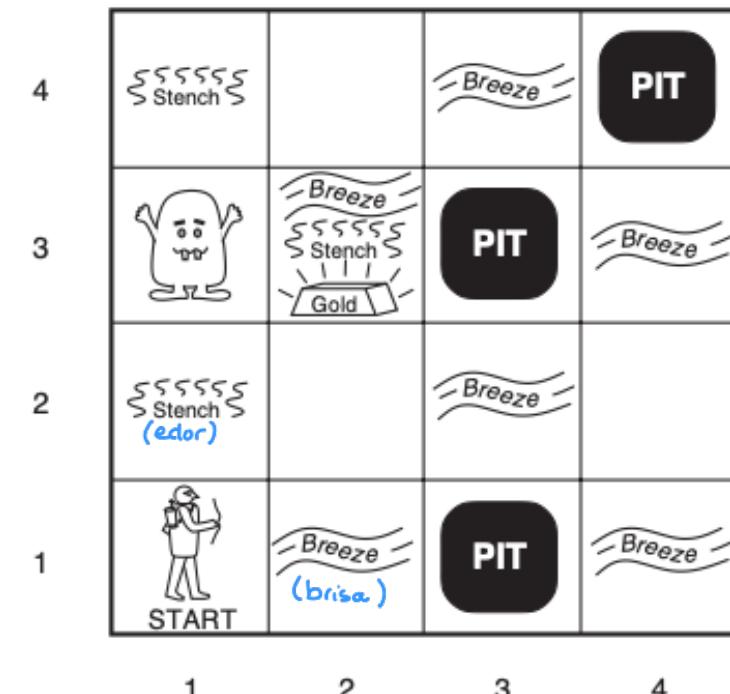
- Dos niveles independientes:
 - **Nivel de conocimiento:** Especificación de qué sabe el agente y cuáles son sus objetivos, a fin de fijar su comportamiento.
 - **Nivel de implementación:** Los detalles de cómo se codifican las localizaciones, o detalles de codificación del sistema.
- Tipo de aproximación:
 - **Declarativa:** Se insertan sentencias en la base de conocimiento hasta que el agente sabe cómo operar en su entorno. (*objetivo*) (*ve a la cafetería*)
 - **Procedural:** Los comportamientos se especifican en código fuente. (*cómo llega a ese objetivo*) (*sal, guia a la ieq...*)
 - La aproximación **híbrida** producen agentes más exitosos.



El Mundo de Wumpus

Descripción PEAS

- Medida de Rendimiento:**
 - +1000 por salir de la cueva con el oro
 - 1000 por caer a un pozo o ser comido por Wumpus
 - 1 por cada movimiento
 - 10 por usar el arco
 - Juego termina cuando el agente muere o cuando sale fuera de la cueva
- Entorno:**
 - un grid de 4x4.
 - [1, 1] es la entrada agente mirando al este.
 - Los elementos (wumpus y oro) están aleatoriamente distribuidos
 - Puede haber un pozo en cualquier estancia con probabilidad 0.2 excepto en 1,1.
 - Solo 1 flecha
- Actuadores:**
 - {Forward, TurnLeft, TurnRight, Grab, Shoot, Climb}
- Sensores:**
 - [Stench, Breeze, Glitter, Bump, Scream]



Observable	Agents	Deterministic	Episodic	Static	Discrete
Partially	Single	Deterministic	Sequential	Static	Discrete

↑
Conoces lo que
tienes a tu alrededor

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	2,1	3,1	4,1
A	OK		
OK			

- **Acción:** (inicial)
- **Percepción:** [None, None, None, None, None]

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1	2,1 A B OK	3,1 P?	4,1
V			
OK			

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus

- **Acción:** F
- **Percepción:** [None, Breeze, None, None, None]

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

- **Acción:** (L, L, F) (R, F)
- **Percepción:** [Stench, None, None, None, None]

Sacas una inferencia

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus

- **Acción:** (R, F), (L, F)
- **Percepción:** [Stench, Breeze, Glitter, None, None]



Lógica

- Conceptos fundamentales de representación lógica y razonamiento
- **Sintaxis y semántica (semiótica)**
 - La sintaxis define **cómo se forman las sentencias/axiomas - Sintaxis**
 - La semántica define **si una sentencia es cierta en un modelo o posible mundo - Semántica**

INIT:

$R_1 \top P_{11}$

BIRDS:

$R_2 \quad B_{11} \Leftrightarrow (P_{12} \vee P_{21})$

$R_3 \quad B_{21} \Leftrightarrow (P_{11} \vee P_{22} \vee P_{31})$

PERCEPTS :

$R_4 \top B_{11}$

$R_5 \quad B_{21}$

Si cualquier conclusión queremos que sea verdadera, se tiene que cumplir todo lo anterior.

base de conocimiento informal.

Si una sentencia α es verdadera en un modelo m , decimos

- m **satisface** α
- m **es un modelo de** α

$x + y = 4$	$m_1 = \{x : 0, y : 4\}$ $m_2 = \{x : 1, y : 3\}$ $m_3 = \{x : 2, y : 2\}$ $m_4 = \{x : 3, y : 1\}$ $m_5 = \{x : 4, y : 0\}$ $m_6 = \{x : 8, y : 5\}$	esto es modelo de esto

$M(\alpha)$

$M(\alpha)$ es el conjunto de todos los modelos de α

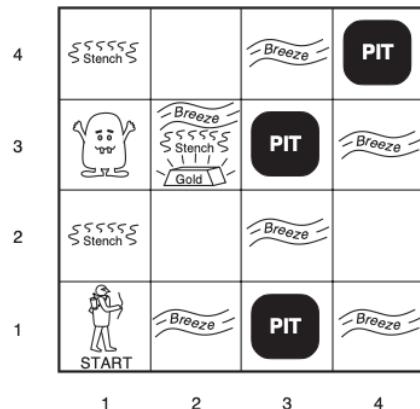
proposiciones
modelos de
conocimiento

α es más restrictiva que β

$\alpha \models \beta$ significa que la sentencia α **implica** β

$\alpha \models \beta$ es verdadero si y solo si $M(\alpha) \subseteq M(\beta)$

semánticamente verdaderas

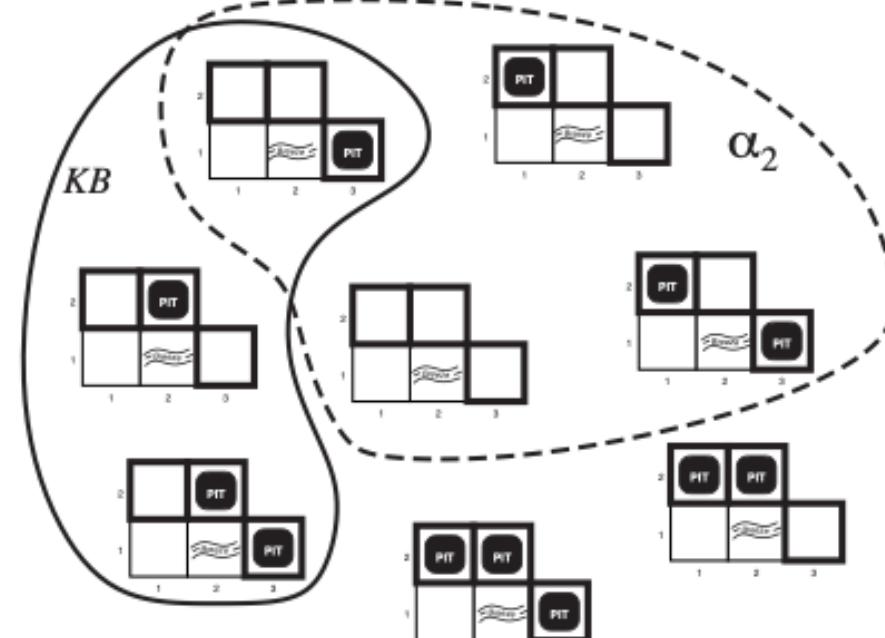


Modelos de la presencia de un pozo en
[1, 2], [2, 2], [3, 1]

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

La línea punteada es $M(\alpha_2)$, α_2 = "no hay pozo en [2, 2]"

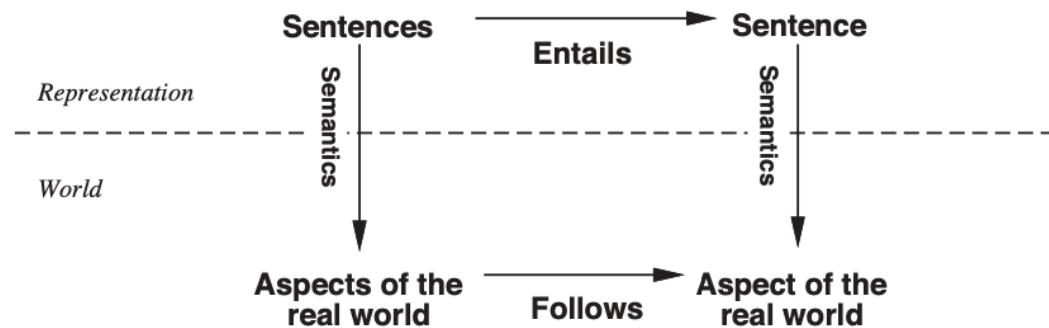


$KB \not\models \alpha_2$!!!!!

↑ no se infiere
lógicamente

porque no influye solo
eso lógicamente.
"el resto de reglas influye".

Si KB es verdadero en el mundo real, entonces cualquier sentencia α derivada de KB mediante un procedimiento de inferencia sólido también es cierto en el mundo real.



Se puede resumir esto en : El conjunto de todas las consecuencias derivadas de la KB es como un pajar y α es como una aguja. Una implicación $\alpha \models \beta$ es como la aguja en el pajar, y la inferencia $KB \vdash_i \alpha$ es encontrarla

- Un algoritmo de **inferencia** i trata de derivar sentencias de la KB ,

$$KB \vdash_i \alpha$$

↑ lógicamente
inferible.

- Si deriva solo sentencias implícitas, es **sólido** (sound / truth-preserving)
- Si puede derivar cualquier sentencia que está implícita, es **completo**

- El ejemplo anterior muestra el proceso de inferencia/deducción lógica. $KB \vdash_i \alpha$
- El algoritmo usado se llama **model checking**:
 - Se enumeran todas los posibles modelos
 - Se identifica el conjunto KB
 - Se chequea que α es cierto para todos los modelos en los que KB es cierto,

$$M(KB) \subseteq M(\alpha)$$

\uparrow

*α se tiene que cumplir
en base de los
conocimientos (KB)*

\models consecutor/deductor semántico (Semántica)
 \vdash Ductor lógico (Sintaxis)

$\alpha \models \beta$ β es consecuencia lógica de α

$KB \vdash_i \alpha$ α es formalmente deducible de KB

Deducción lógica:

- 1. $p \rightarrow (q \rightarrow r)$
- 2. $p \rightarrow q$
- 3. p
- 4. $q \rightarrow r$ mp(2,3)
- 5. q
- 6. r mp(4,5)

Ejemplo:

$i \vdash_1 p \rightarrow r ?$

- | | |
|----------------|---|
| $p \text{ CN}$ | - 1 $\neg p \rightarrow q$ |
| $q \text{ PA}$ | - 2 $q \rightarrow r$ |
| $r \wedge p_o$ | - 3 $\neg p$ |
| | - 4 q m 1,3 |
| | - 5 r m 2,4 |
| | - 6 $\neg p \rightarrow r \dashv \odot 3,5$ |

l teorema
de deducción

Lógica Proposicional

Sintaxis

La sintaxis de la lógica proposicional define las sentencias permitidas

- **Símbolos proposicionales:** $P, Q, R, W_{1,3}, North$
- *True, False*
- **Sentencias complejas** usando paréntesis o **conectores lógicos**

Sentence → *AtomicSentence* | *ComplexSentence*

AtomicSentence → *True* | *False* | *P* | *Q* | *R* | ...

ComplexSentence → (*Sentence*) | [*Sentence*]
| $\neg Sentence$
| *Sentence* \wedge *Sentence*
| *Sentence* \vee *Sentence*
| *Sentence* \Rightarrow *Sentence*
| *Sentence* \Leftrightarrow *Sentence*

OPERATOR PRECEDENCE : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$



Semántica

La semántica define las reglas para determinar la verdad de una sentencia con respecto a un modelo en particular.

- Un modelo define el valor de verdad para cada uno de los símbolos proposicionales
 - Un posible modelo m_1 (Hay $2^3 = 8$ posibles modelos)

$$m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\} \quad \begin{matrix} \text{AND} \\ \text{OR} \end{matrix} \quad \begin{matrix} \text{Pentance Q} \\ \downarrow \end{matrix}$$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) \quad true \wedge (false \vee true) = true$$

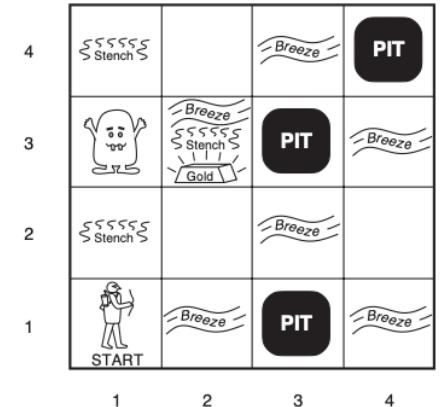
$\oplus \Rightarrow \Leftrightarrow$

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$



Una base de conocimiento simple

- $P_{x,y}$ es verdadero si hay un pozo en $[x, y]$.
- $W_{x,y}$ es verdadero si hay un wumpus en $[x, y]$, vivo o muerto.
- $B_{x,y}$ es verdadero si el agente percibe una brisa en $[x, y]$.
- $S_{x,y}$ es verdadera si el agente percibe un hedor en $[x, y]$.



No puede haber pozo en la casilla de salida

$$R_1 : \neg P_{1,1}$$

Por cada casilla, establecemos la condición de que haya brisa

$$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Las percepciones que llevamos hasta el momento

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

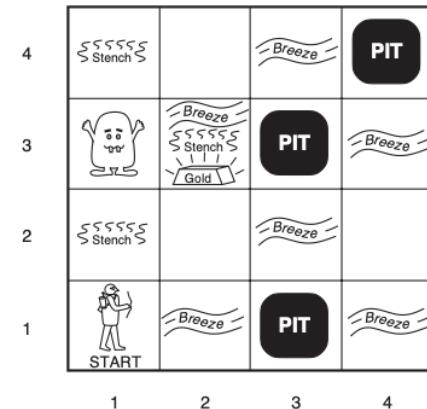
base inicial de conocimiento.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1	2,1 A	3,1 P?	4,1
V OK	B OK		

Un proceso de inferencia simple

- El objetivo es decidir $KB \models \alpha$ para algún α
- ¿si α es $\neg P_{1,2}$? $P_{1,2}$ es falso pq KB es falso (a su misma altura).
- Veamos los símbolos relevantes $B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$

$\neg P_{1,2}$ es infonble.



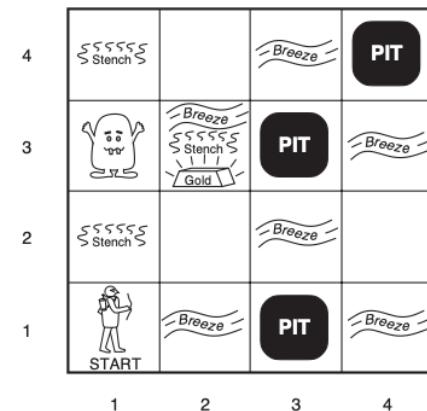
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
:	:	:	:	:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	false	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	true	true	true	true	true	true	<u>true</u>
false	true	false	false	true	false	false	true	false	false	true	true	false
:	:	:	:	:	:	:	:	:	:	:	:	:
true	false	true	true	false	true	false						

Actividad de clase

Supón que el agente ha progresado hasta el punto que se muestra en la figura de abajo, sin haber percibido nada en [1,1], una brisa en [2,1] y un hedor en [1,2], y ahora se ocupa del contenido de [1,3], [2,2] y [3,1]. Cada uno de estos puede contener un hoyo y, como máximo, uno puede contener un wumpus. Siguiendo el ejemplo de la trampa 15 y 16, construye el conjunto de mundos posibles (deberías encontrar 32 de ellos). Marca los mundos en los que la KB es verdadera y aquellos en los que cada una de las siguientes sentencias es *true*:

- $\alpha_2 = \text{"No hay hoyo en [2, 2]"}.$
- $\alpha_3 = \text{"No hay hoyo en [1, 3]"}.$

Por tanto, demuestra que $KB \models \alpha_2$ y $KB \models \alpha_3$.



1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

Un proceso de inferencia simple

(el algoritmo que te permite hacer las tablas de verdad)

- TT-ENTAILS? realiza una enumeración recursiva de un espacio finito de asignaciones a símbolos.
- **Sólido y completo**
- $O(2^n)$

4			PIT
3			PIT
2			
1	START		PIT
			3 4

function TT-ENTAILS?(*KB*, α) **returns** true or false

inputs: *KB*, the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

symbols \leftarrow a list of the proposition symbols in *KB* and α

return TT-CHECK-ALL(*KB*, α , *symbols*, { })

function TT-CHECK-ALL(*KB*, α , *symbols*, *model*) **returns** true or false

if EMPTY?(*symbols*) **then**

if PL-TRUE?(*KB*, *model*) **then return** PL-TRUE?(α , *model*)

else return true // when *KB* is false, always return true

else do

P \leftarrow FIRST(*symbols*)

rest \leftarrow REST(*symbols*)

return (TT-CHECK-ALL(*KB*, α , *rest*, *model* \cup {*P* = true})

and

 TT-CHECK-ALL(*KB*, α , *rest*, *model* \cup {*P* = false }))



Demostración de Teorema Proposicional

- Hasta ahora hemos determinado la implicación mediante **model checking**.
- Ahora mostramos cómo se puede realizar la implicación por **theorem proving**

Aplicar reglas de inferencia directamente a las sentencias en nuestra base de conocimiento para construir una prueba de la sentencia deseada sin consultar modelos.

Conceptos

- **Equivalencia lógica:** dos axiomas α y β son lógicamente equivalentes si son verdaderos en el mismo conjunto de modelos

si solo si.

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \text{ contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge$$

$$\alpha \equiv \beta \text{ si y solo si } \alpha \models \beta \text{ y } \beta \models \alpha$$

- Hasta ahora hemos determinado la implicación mediante **model checking**.
- Ahora mostramos cómo se puede realizar la implicación por **theorem proving**

Aplicar reglas de inferencia directamente a las sentencias en nuestra base de conocimiento para construir una prueba de la sentencia deseada sin consultar modelos.

Base conocimiento + combinación

↳ deduce modelo.

Conceptos

Validez: una sentencia es válida (es una tautología) si es verdadero en *todos* los modelos. Por ejemplo $\alpha \vee \neg\alpha$ es válida. Cada sentencia válida es equivalente a Verdadero.

Teorema de deducción : Para cualquier $\alpha, \beta, \alpha \models \beta$ si y solo si $\alpha \Rightarrow \beta$ es válido. Mirar tabla para comprobarlo.

es deducible lógicamente de α

Satisfacibilidad (SAT Problem): una sentencia es satisfacible si es verdadero en *algún* modelo

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	false	false	false	true	false	false						
true	false	true	true	false	true	false						

α es válido si, y solo si, $\neg\alpha$ es no satisfacible
 α es satisfacible si, y solo si, $\neg\alpha$ no es válido

$\alpha \models \beta$ si y solo si $(\alpha \wedge \neg\beta)$ es no satisfacible.
Prueba por contradicción o reducción al absurdo.

SAT problem — NP-completo

Inferencia y demostraciones

- **Demostración:** una cadena de conclusiones que conduce al objetivo deseado
- **Reglas de inferencia** que pueden aplicarse para derivar una prueba

Modus ponens

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

Dado $\alpha \Rightarrow \beta$ y α , entonces se puede inferir β

Y-Eliminación

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

Eliminación bicondicional

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

$$\frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Resolución a mano ($\neg P_{1,2}$)

$\frac{\alpha \wedge \beta}{\alpha}$	And-Elimination
$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$	Modus ponens
$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of \wedge
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of \vee
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of \wedge
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of \vee
$\neg(\neg \alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of \wedge over \vee
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of \vee over \wedge

$R_4 : \neg B_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

$R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

$R_8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$ Con R8 y R4

$R_9 : \neg(P_{1,2} \vee P_{2,1})$

$R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$

4			
3		 	
2			
1	 START		
	1	2	3
			4

Algoritmo 1: Resolución por búsqueda

- Podemos aplicar cualquier algoritmo de resolución por búsqueda del tema anterior:
 - ESTADO INICIAL: la base de conocimientos inicial.
 - ACCIONES: el conjunto de acciones consta de todas las reglas de inferencia aplicadas a todas las oraciones que coinciden con la mitad superior de la regla de inferencia.
 - RESULTADO: el resultado de una acción es agregar el axioma en la mitad inferior de la regla de inferencia.
 - OBJETIVO: el objetivo es un estado que contiene el axioma que estamos tratando de demostrar.
- Es más eficiente que model-checking, y puede ignorar proposiciones irrelevantes
- Un sistema lógico es **monotónico**, ya que el conjunto de axiomas implicadas solo puede aumentar a medida que se agrega información a la base de conocimientos

Si $KB \models \alpha$, entonces $KB \wedge \beta \models \alpha$

Algoritmo 2: Prueba por resolución

- Aunque las reglas de inferencia sean sólidas, los algoritmos de inferencia (búsqueda iterativa en profundidad, por ejemplo) pueden no ser completos si las reglas disponibles no son adecuadas.
- **Resolución** es una única regla de inferencia que produce un algoritmo de inferencia **completo** cuando se combina con cualquier algoritmo de búsqueda completo.
- Se aplica a cláusulas: una disyunción de literales

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k} \quad \text{Donde } l_i \text{ y } m \text{ son complementarios}$$

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

- Un demostrador de teoremas basado en resolución puede, para cualquier enunciado α y β en lógica proposicional, decidir si $\alpha \models \beta$. Veamos cómo:

Conjunctive Normal Form (CNF)

- Ya que resolución solo se aplica a cláusulas (disyunciones de literales), es relevante que todo el conocimiento sea expresado como cláusulas.
- *Cada sentencia de lógica proposicional es lógicamente equivalente a una conjunción de cláusulas*
- Si una sentencia cumple esto, está en CNF
- Mirar Figura 7.12 del libro para ver una gramática

$$\frac{\alpha \wedge \beta}{\alpha} \quad \text{And-Elimination}$$

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta} \quad \text{Modus ponens}$$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

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

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

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

↑ OR ↑ AND (γ)



Algoritmo de resolución

- Se basa en el principio de prueba por contradicción

Probamos $KB \models \alpha$ probando que $(KB \wedge \neg\alpha)$ es insatisfacible

function PL-RESOLUTION(KB, α) **returns** true or false

inputs: KB , the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

$new \leftarrow \{\}$

loop do

for each pair of clauses C_i, C_j **in** $clauses$ **do**

$resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)

if $resolvents$ contains the empty clause **then return** true

$new \leftarrow new \cup resolvents$

if $new \subseteq clauses$ **then return** false

$clauses \leftarrow clauses \cup new$

PL-RESOLVE devuelve el conjunto de todas las posibles cláusulas obtenidas resolviendo las dos entradas,

$$KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

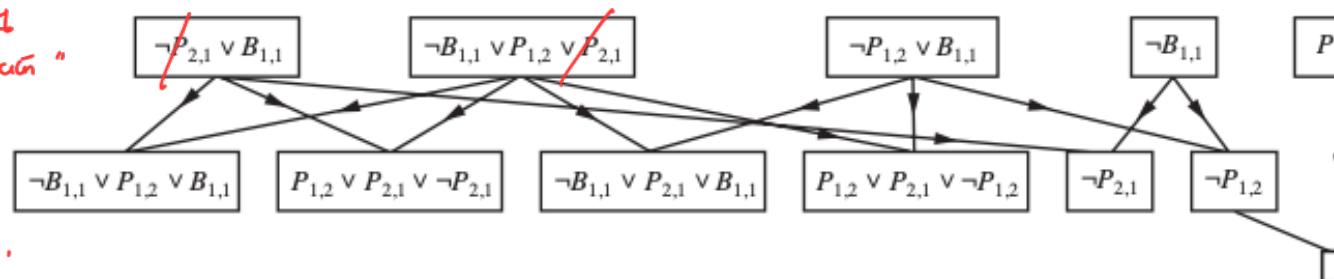
Vamos descartando en cada iteración

"Se elimina 1 en cada iteración"

Si ocurre termina y si no llegas a nada, no se puede concluir.

- Cada par que contiene literales complementarios se resuelve hasta que:

- no hay nuevas cláusulas que se puedan agregar, en cuyo caso $KB \not\models \alpha$; o,
- dos cláusulas resuelven producir la cláusula vacía, en cuyo caso $KB \models \alpha$



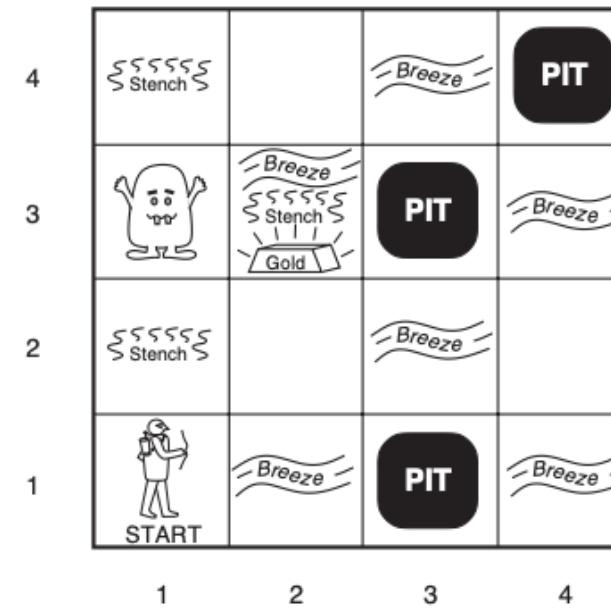
α es $\neg P_{1,2}$

$\neg\alpha$

Agentes basados en Lógica Proposicional

Objetivos

- Queremos que el agente deduzca, en la medida de lo posible, el estado del mundo dada la historia de percepciones disponibles.
- Es necesario escribir un modelo lógico de los efectos de las acciones
- La inferencia lógica tiene que aplicarse en el mundo de Wumpus
- Veremos que el agente puede hacer un seguimiento del mundo eficientemente sin tener que ir para atrás en el histórico de percepciones para cada inferencia.
- Finalmente veremos cómo el agente puede utilizar inferencias lógicas para construir planes que de forma garantizada llevarán a cabo sus objetivos, dado que la KB sea verdadera en el mundo dado.



El estado actual del mundo

- Un agente lógico opera deduciendo qué hacer a partir de una base de conocimiento de axiomas sobre el mundo.
- La base de conocimiento contiene:
 - Axiomas sobre cómo funciona el mundo
 - Sentencias que provienen de las percepciones del agente en un mundo en particular
- ¿Qué sabemos del estado actual del mundo? Sección 7.4.3 libro.
 - Que en la entrada de la cueva no hay pozo ni wumpus
 - $\neg P_{1,1}, \neg W_{1,1}$
 - Que si detecta brisa, en cualquier casilla, es porque está al lado de un pozo, y si detecta hedor es que está al lado del wumpus
 - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - $S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$
 - ...
 - Que solo hay exactamente un wumpus. Se expresa en dos pasos
 - Que al menos hay un Wumpus:
 - $W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,3} \vee W_{4,4}$
 - Que como máximo hay un Wumpus:
 - $\neg W_{1,1} \vee \neg W_{1,2}$
 - $\neg W_{1,1} \vee \neg W_{1,3}$
 - ...
 - $\neg W_{1,1} \vee \neg W_{1,3}$

	$\leq\leq\leq\leq\leq$ Stench		\approx Breeze	PIT
3		\approx Breeze	$\leq\leq\leq\leq$ Stench	PIT
2	$\leq\leq\leq\leq$ Stench		\approx Breeze	
1	 START	\approx Breeze	PIT	\approx Breeze
	1	2	3	4

El estado actual del mundo

- **Percepciones:**

Hedor en t = 4

$Stench^4$

- **Fluents:** aspectos del mundo que varían

$L_{1,1}^0 \quad FacingEast^0 \quad HaveArrow^0 \quad WumpusAlive^0$

- **Variables atemporales**

$$L_{x,y}^t \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$$

$$L_{x,y}^t \Rightarrow (Stench^t \Leftrightarrow S_{x,y})$$

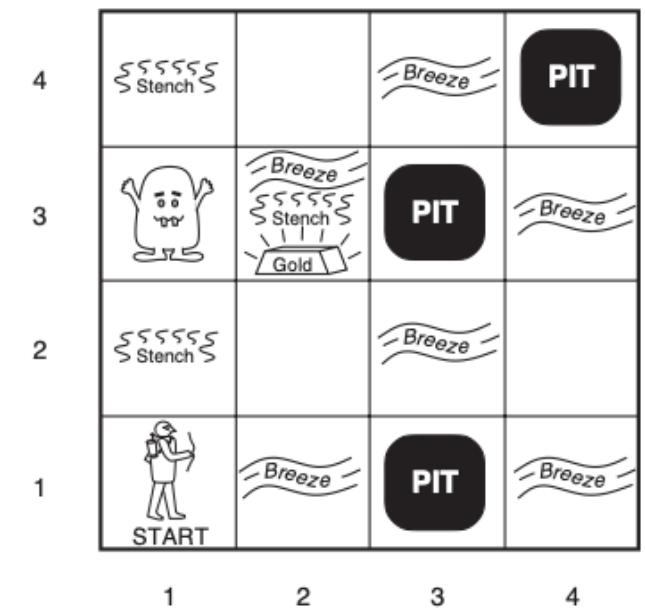
- **Axiomas de efectos**

$$L_{1,1}^0 \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L_{2,1}^1 \wedge \neg L_{1,1}^1)$$

- **Frame axioms**

$$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})$$

$$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$$



El estado actual del mundo

- **Percepciones:**

Hedor en t = 4

$Stench^4$

- **Fluents:** aspectos del mundo que varían

$L_{1,1}^0$ $FacingEast^0$ $HaveArrow^0$ $WumpusAlive^0$

- **Variables atemporales**

$$L_{x,y}^t \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$$

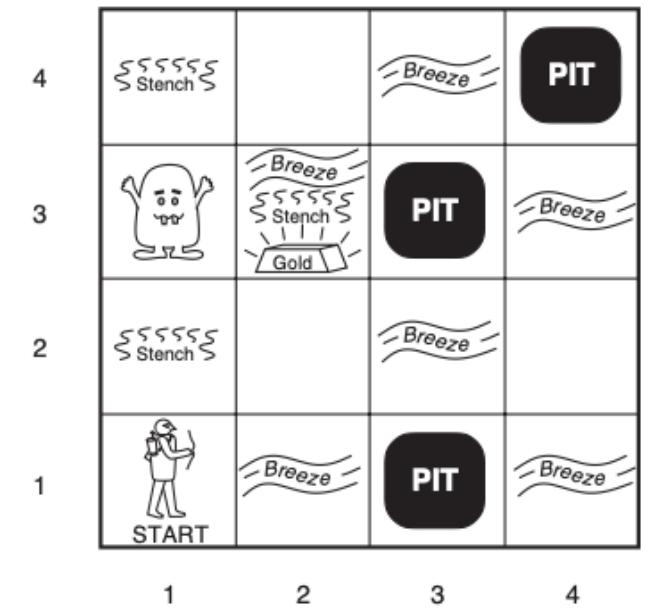
$$L_{x,y}^t \Rightarrow (Stench^t \Leftrightarrow S_{x,y})$$

- **Axiomas de estado sucesor**

1 $HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$

- **Axiomas de estado sucesor de localización del agente**

$$L_{1,1}^{t+1} \Leftrightarrow (L_{1,1}^t \wedge (\neg Forward^t \wedge Bump^{t+1}) \\ \vee (L_{1,2}^t \wedge (South^t \wedge Forward^t)) \\ \vee (L_{2,1}^t \wedge (West^t \wedge Forward^t)))$$



Ejemplo:

Percepción

Acción

$\neg Stench^0 \wedge \neg Breeze^0 \wedge \neg Glitter^0 \wedge \neg Bump^0 \wedge \neg Scream^0$	$Forward^0$
$\neg Stench^1 \wedge Breeze^1 \wedge \neg Glitter^1 \wedge \neg Bump^1 \wedge \neg Scream^1$	$TurnRight^1$
$\neg Stench^2 \wedge Breeze^2 \wedge \neg Glitter^2 \wedge \neg Bump^2 \wedge \neg Scream^2$	$TurnRight^2$
$\neg Stench^3 \wedge Breeze^3 \wedge \neg Glitter^3 \wedge \neg Bump^3 \wedge \neg Scream^3$	$Forward^3$
$\neg Stench^4 \wedge \neg Breeze^4 \wedge \neg Glitter^4 \wedge \neg Bump^4 \wedge \neg Scream^4$	$TurnRight^4$
$\neg Stench^5 \wedge \neg Breeze^5 \wedge \neg Glitter^5 \wedge \neg Bump^5 \wedge \neg Scream^5$	$Forward^5$
$Stench^6 \wedge \neg Breeze^6 \wedge \neg Glitter^6 \wedge \neg Bump^6 \wedge \neg Scream^6$	

¿Qué sabe el agente del mundo? ← *terminos que definir proposiciones que nos digan.*

$$\text{ASK}(KB, L_{1,2}^6) = \text{true}$$

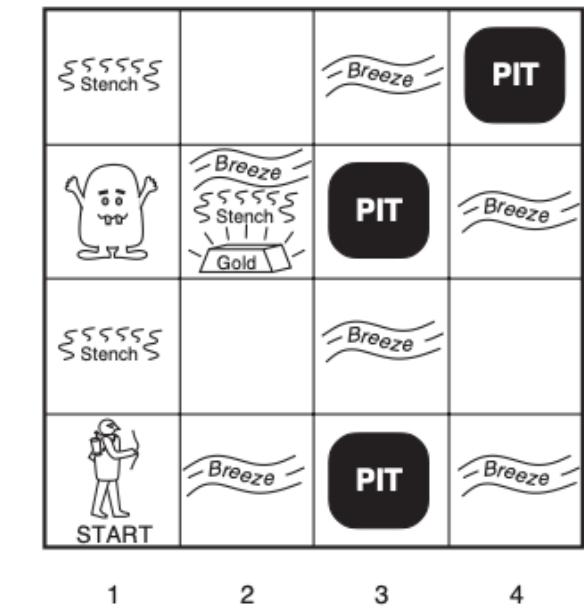
$$\text{ASK}(KB, W_{1,3}) = \text{true}$$

$$\text{ASK}(KB, P_{3,1}) = \text{true},$$

Axiomas a añadir a la KB para saber si es seguro moverse a una localización

$$OK_{x,y}^t \Leftrightarrow \neg P_{x,y} \wedge \neg (W_{x,y} \wedge WumpusAlive^t)$$

$$\text{ASK}(KB, OK_{2,2}^6) = \text{true}$$



function HYBRID-WUMPUS-AGENT(*percept*) returns an *action*

inputs: *percept*, a list, [*stench*,*breeze*,*glitter*,*bump*,*scream*]

persistent: *KB*, a knowledge base, initially the atemporal “wumpus physics”

t, a counter, initially 0, indicating time

plan, an action sequence, initially empty

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))

TELL the *KB* the temporal “physics” sentences for time *t*

safe $\leftarrow \{[x, y] : \text{ASK}(KB, OK_{x,y}^t) = \text{true}\}$

if $\text{ASK}(KB, \text{Glitter}^t) = \text{true}$ **then**

plan $\leftarrow [\text{Grab}] + \text{PLAN-ROUTE}(\text{current}, \{[1,1]\}, \text{safe}) + [\text{Climb}]$

if *plan* is empty **then**

unvisited $\leftarrow \{[x, y] : \text{ASK}(KB, L_{x,y}^{t'}) = \text{false} \text{ for all } t' \leq t\}$

plan $\leftarrow \text{PLAN-ROUTE}(\text{current}, \text{unvisited} \cap \text{safe}, \text{safe})$

if *plan* is empty and $\text{ASK}(KB, \text{HaveArrow}^t) = \text{true}$ **then**

possible_wumpus $\leftarrow \{[x, y] : \text{ASK}(KB, \neg W_{x,y}) = \text{false}\}$

plan $\leftarrow \text{PLAN-SHOT}(\text{current}, \text{possible_wumpus}, \text{safe})$

if *plan* is empty **then** // no choice but to take a risk

not_unsafe $\leftarrow \{[x, y] : \text{ASK}(KB, \neg OK_{x,y}^t) = \text{false}\}$

plan $\leftarrow \text{PLAN-ROUTE}(\text{current}, \text{unvisited} \cap \text{not_unsafe}, \text{safe})$

if *plan* is empty **then**

plan $\leftarrow \text{PLAN-ROUTE}(\text{current}, \{[1, 1]\}, \text{safe}) + [\text{Climb}]$

action $\leftarrow \text{POP}(\text{plan})$

TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))

t $\leftarrow t + 1$

return *action*

function PLAN-ROUTE(*current*,*goals*,*allowed*) returns an action sequence

inputs: *current*, the agent's current position

goals, a set of squares; try to plan a route to one of them

allowed, a set of squares that can form part of the route

problem $\leftarrow \text{ROUTE-PROBLEM}(\text{current}, \text{goals}, \text{allowed})$

return A*-GRAPH-SEARCH(*problem*)



Planificar usando inferencia proposicional

Se cumpla kb y que
se cumpla la proposición.

```

function SATPLAN(init, transition, goal,  $T_{\max}$ ) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
     $T_{\max}$ , an upper limit for plan length
    ← porque los axiomas se
    lo meten todos a la vez.
    y luego sacas deducciones.
  for  $t = 0$  to  $T_{\max}$  do
     $cnf \leftarrow \text{TRANSLATE-TO-SAT}(\textit{init}, \textit{transition}, \textit{goal}, t)$ 
     $model \leftarrow \text{SAT-SOLVER}(cnf)$ 
    if model is not null then
      return EXTRACT-SOLUTION(model)
    return failure
  
```

1. Construye una sentencia que contenga
 - a) $Init^0$ — Una colección de aserciones sobre el estado inicial
 - b) $Transition^1, \dots, Transition^t$ — Axiomas para todas las posibles acciones hasta t
 - c) La aserción de que se ha llegado al objetivo en t *Satisfactibilidad* *ayuda a depurar*
2. Presentar la sentencia a un SAT Solver en cada t *ayuda a depurar*
3. Si hay un modelo, extraer del modelo las acciones a las que se les ha asignado *true*

Ejercicios de lógica proposicional

1. ¿Cuáles son correctos?

- a. $\text{False} \models \text{True}$.
- b. $\text{True} \models \text{False}$.
- c. $(A \wedge B) \models (A \Leftrightarrow B)$.
- d. $A \Leftrightarrow B \models A \vee B$.
- e. $A \Leftrightarrow B \models \neg A \vee B$.
- f. $(A \wedge B) \Rightarrow C \models (A \Rightarrow C) \vee (B \Rightarrow C)$.
- g. $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$.
- h. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$.
- i. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B) \wedge (\neg D \vee E)$.
- j. $(A \vee B) \wedge \neg(A \Rightarrow B)$ is satisfiable.
- k. $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ is satisfiable.
- l. $(A \Leftrightarrow B) \Leftrightarrow C$ has the same number of models as $(A \Leftrightarrow B)$ for any fixed set of proposition symbols that includes A, B, C .

2. Prueba estas aserciones

- a. α is valid if and only if $\text{True} \models \alpha$.
- b. For any α , $\text{False} \models \alpha$.
- c. $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.
- d. $\alpha \equiv \beta$ if and only if the sentence $(\alpha \Leftrightarrow \beta)$ is valid.
- e. $\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable.

3. Usando la tabla de la transparencia 25, di si las sentencias son válidas, no satisfacibles o ninguna de ambas:

- a. $\text{Smoke} \Rightarrow \text{Smoke}$
- b. $\text{Smoke} \Rightarrow \text{Fire}$
- c. $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow (\neg\text{Smoke} \Rightarrow \neg\text{Fire})$
- d. $\text{Smoke} \vee \text{Fire} \vee \neg\text{Fire}$
- e. $((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire}) \Leftrightarrow ((\text{Smoke} \Rightarrow \text{Fire}) \vee (\text{Heat} \Rightarrow \text{Fire}))$
- f. $(\text{Smoke} \Rightarrow \text{Fire}) \Rightarrow ((\text{Smoke} \wedge \text{Heat}) \Rightarrow \text{Fire})$
- g. $\text{Big} \vee \text{Dumb} \vee (\text{Big} \Rightarrow \text{Dumb})$