ARQUITECTURAS PARA AGENTES

Arquitecturas para Agentes

- Un *agente* es un sistema computacional capaz de realizar *acciones autónomas* y *flexibles* …

- Hechos que uno debe enfrentar para construir sistemas basados en agentes …

- Tres tipos de *arquitecturas de agentes*:
  - Simbólica/lógica
    - Razonamiento Deductivo
    - Razonamiento Práctico (BDI)
  - Reactiva
  - Híbrida

Arquitecturas para Agentes

- Originalmente (1956 -1985), la mayoría de los agentes diseñados por la IA eran agentes razonadores simbólicos
- Se proponía que los agentes usaran razonamiento lógico explícito para decidir qué hacer
- Problemas con el razonamiento simbólico llevaron a un movimiento en contra de eso (1985 -) llamado el movimiento de agentes reactivos
- Desde 1990 al presente, aparecieron una serie de alternativas: arquitecturas *híbridas*, que tratan de combinar lo mejor de las arquitecturas de razonamiento y reactivas

Agentes Razonadores Simbólicos

- Definimos un agente deductive como una arquitectura que:

1. Contiene un modelo que representa explícita y simbólicamente el mundo

Agentes Razonadores Simbólicos

- Definimos un agente deductivo como una arquitectura que:

  2. Toma decisiones (p. ej. qué acciones ejecutar) a través del razonamiento simbólico

Vaso

Jarro con agua

Tomar agua

Agentes Razonadores Simbólicos

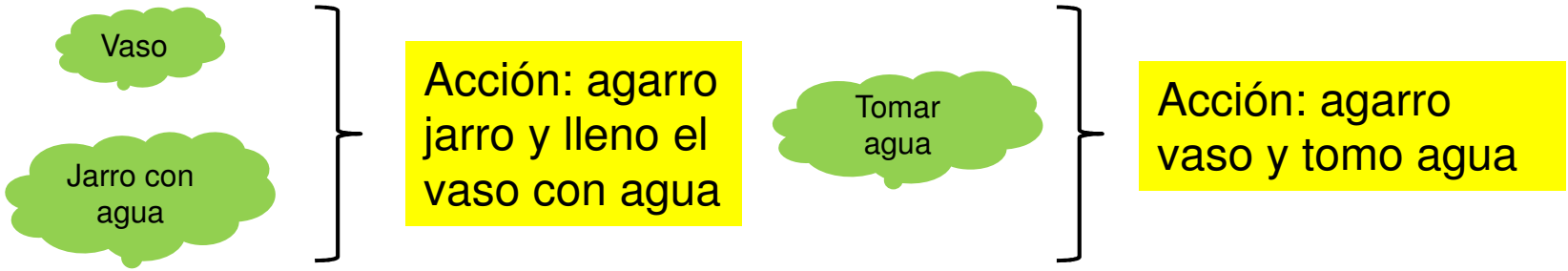
- Si intentamos construir un agente de esa forma, debemos resolver varios problemas:

1. *El problema de transducción*:
   trasladar el mundo real a una descripción simbólica adecuada y precisa en un lapso de tiempo adecuado para que sea útil.
   Ejemplos: visión, reconocimiento de voz, aprendizaje

Agentes Razonadores Simbólicos

- Si intentamos construir un agente de esa forma, debemos resolver varios problemas:

2. *El problema de representación/razonamiento*:
cómo representar simbólicamente información de entidades y procesos complejos del mundo real, y cómo hacer que los agentes razonen usando esa información en un lapso de tiempo adecuado para que sea útil. Ejemplos: representación del conocimiento, razonamiento automático, planificación automática



Agentes Razonadores Simbólicos

- ¿Cómo puede un agente decidir qué hacer usando deducción?

- La idea básica es usar lógica para codificar una teoría que estipule la mejor acción a realizar en una situación particular

- Sea:
  - ρ esa teoría (típicamente un conjunto de reglas)
  - Δ una BBDD lógica que describe el estado actual del mundo
  - Ac el conjunto de acciones que un agente puede ejecutar
  - $\Delta \vdash_{\rho} \phi$ significa que ϕ puede ser deducido de Δ usando ρ

Agentes Razonadores Simbólicos

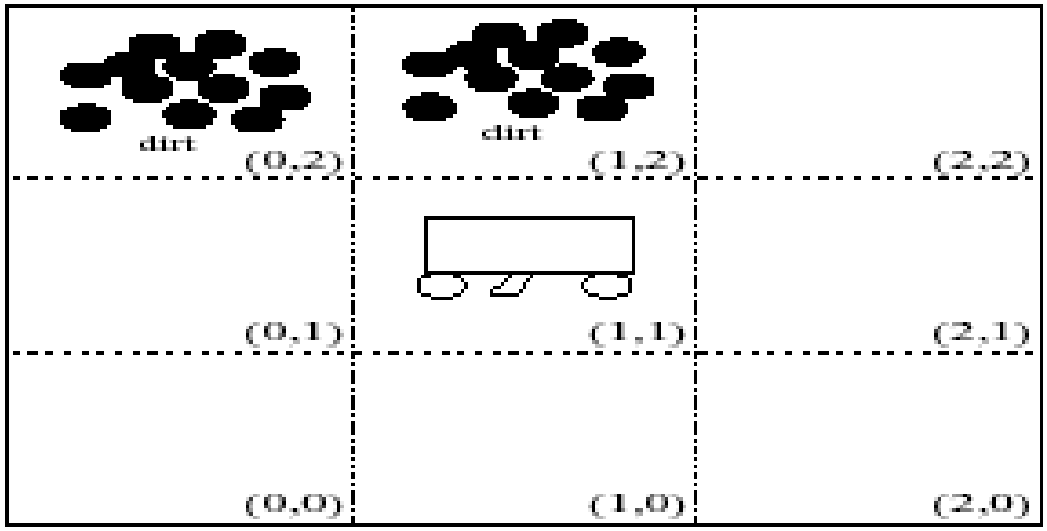
/* *tratar de encontrar una acción prescrita* */
Para cada $a \in Ac$ hacer
    si $\Delta \vdash_{\rho} Do(a)$ entonces
        retornar a
    fin decisión
Fin bucle

/* *tratar de encontrar una acción no excluida* */
Para cada $a \in Ac$ hacer
    si $\Delta \not\vdash_{\rho} \neg Do(a)$ entonces
        retornar a
    fin decisión
Fin bucle

retornar $null$ /* *no se encontró una acción* */

Agentes Razonadores Simbólicos

- Un ejemplo: The Vacuum World
- Objetivo es hacer que el robot limpie toda la mugre



Agentes Razonadores Simbólicos

- Use 3 predicados del dominio para resolver el problema :

$In(x, y)$	agente está en (x, y)
$Dirt(x, y)$	hay mugre en (x, y)
$Facing(d)$	el agente está en dirección d

- Posibles acciones:

$$Ac = \{turn, forward, suck\}$$

P.S. *turn* significa vire a la derecha

Agentes Razonadores Simbólicos

- Reglas ρ para determinar qué hacer:

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward)$$
$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward)$$
$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn)$$
$$In(0,2) \wedge Facing(east) \longrightarrow Do(forward)$$

In(x,y) ^ Dirt(x,y) → Do(suck)

- …y así sucesivamente!

- Usando estas reglas (+ otras obvias), comenzando en $(0, 0)$ el robot limpiará la mugre

Agentes Razonadores Simbólicos

- Problemas:
  - ¿Cómo convertir la entrada de una video cámara en $Dirt(0, 1)$?
  - La toma de decisiones asume un ambiente estático: racionalidad calculativa
- Normalmente eso involucra resolver problemas NP completos
- Soluciones típicas:
  - Usar una lógica más débil
  - Usar representaciones simbólicas no lógicas
  - Mover el énfasis del razonamiento de *run time* a *design time*
- Veremos un ejemplo de uno de dichos enfoques

Agentes Razonadores Prácticos

- El razonamiento práctico en un humano consiste en 2 actividades:
  - *deliberación*
    decidir *qué situación (estado) queremos lograr*

  - *razonamiento medios-fines (means-ends)*
    decidir *cómo* lograr el estado deseado

- Las salidas de la deliberación son *intenciones*

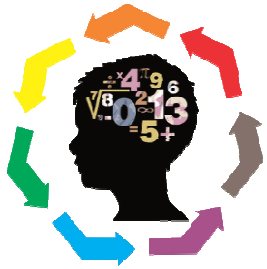
Intenciones en Razonamiento Práctico

- Note que las intenciones son más fuertes que los deseos:

  "Mi ***deseo*** de jugar basketball esta tarde es meramente una influencia potencial de mi conducta para esta tarde. Debe enfrentar mis otros ***deseos*** relevantes [. . .] antes de establecer que voy a hacer. En contraste, una vez que yo ***intento*** jugar basketball esta tarde, la cuestión está resuelta: normalmente no necesito seguir considerando las ventajas y desventajas. Cuando llega la tarde, normalmente ejecuto mis ***intenciones***." (Bratman, 1990)

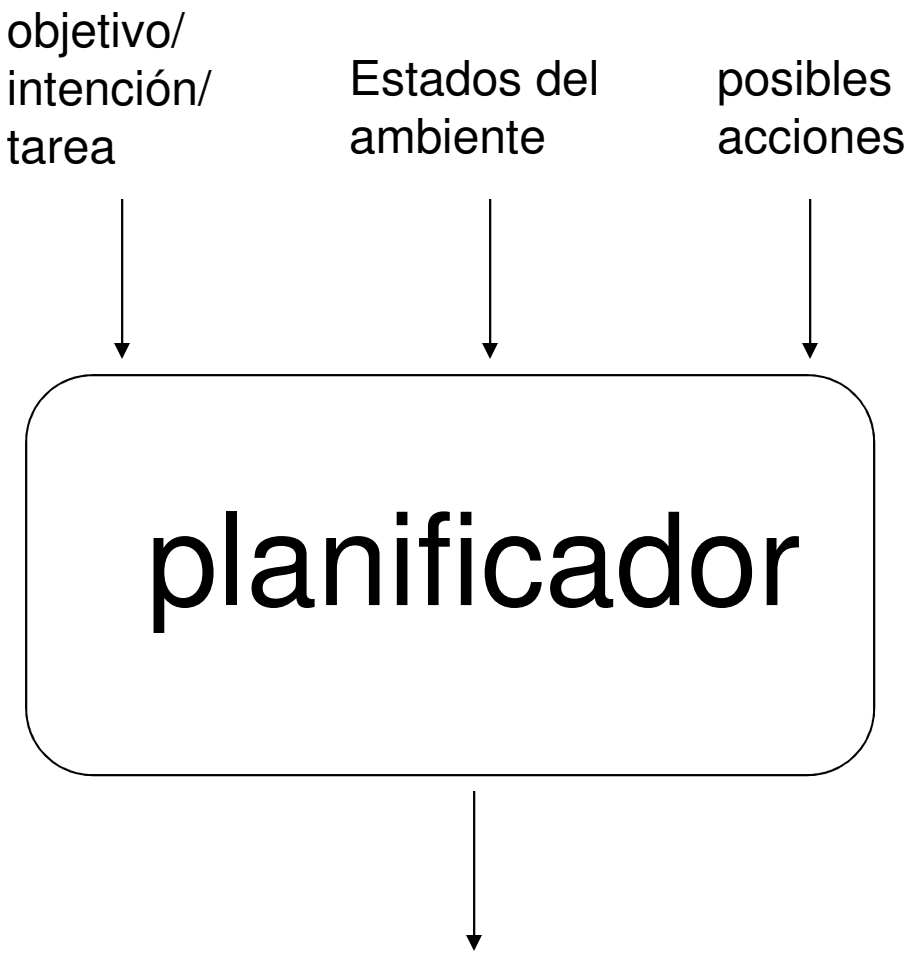
¿Qúe es Razonamiento Medios-Fin?

- La idea básica es dar a un agente:
  - representación del objetivo/intención a lograr
  - representación de las acciones que puede ejecutar
  - representación del ambiente



Y dejar que genere un *plan* para lograr el objetivo

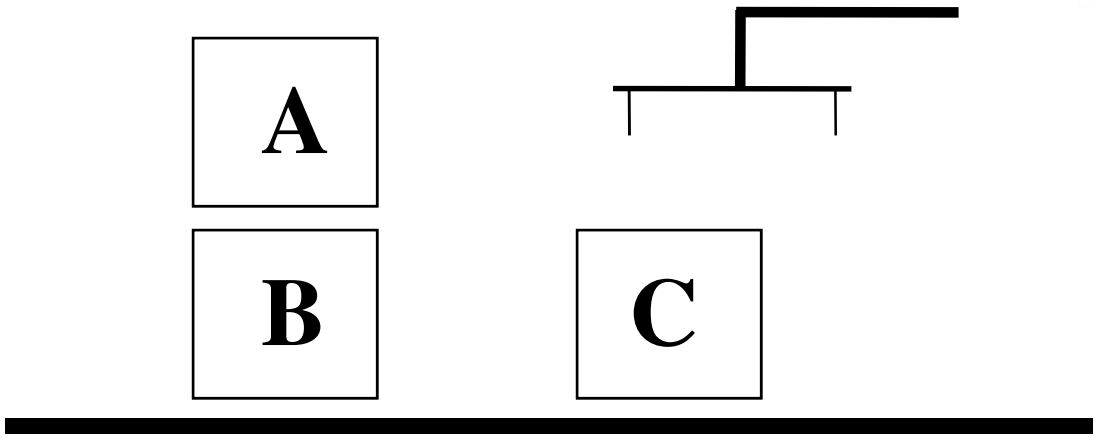
- Esencialmente, esto es *programación automática*


objetivo/
intención/
tarea
Estados del
ambiente
posibles
acciones
planificador
plan para lograr objetivo

Planificación

- Pregunta: Cómo representamos. . .
  - Objetivo a lograr
  - Estados del ambiente
  - Acciones disponibles para ser ejecutadas
  - El plan

The Blocks World



- Ilustraremos eso usando el "*blocks world*"
- Contiene un brazo robótico, 3 bloques (A, B, y C) de igual tamaño, y una mesa

Ontología del Blocks World

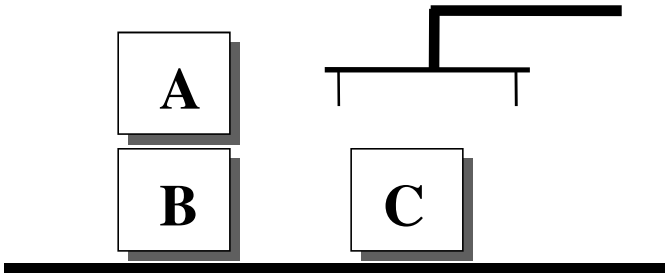
- Para representar este mundo, necesitamos una *ontología*

On(x, y)	objeto *x* sobre objeto *y*
OnTable(x)	objeto *x* está sobre la mesa
Clear(x)	nada está sobre objeto *x*
Holding(x)	brazo sostiene al objeto *x*
ArmEmpty	brazo no está sosteniendo algo

The Blocks World

- Aquí hay una representación de la situación anterior:

Clear(A)
On(A, B)
OnTable(B)
OnTable(C)
ArmEmpty

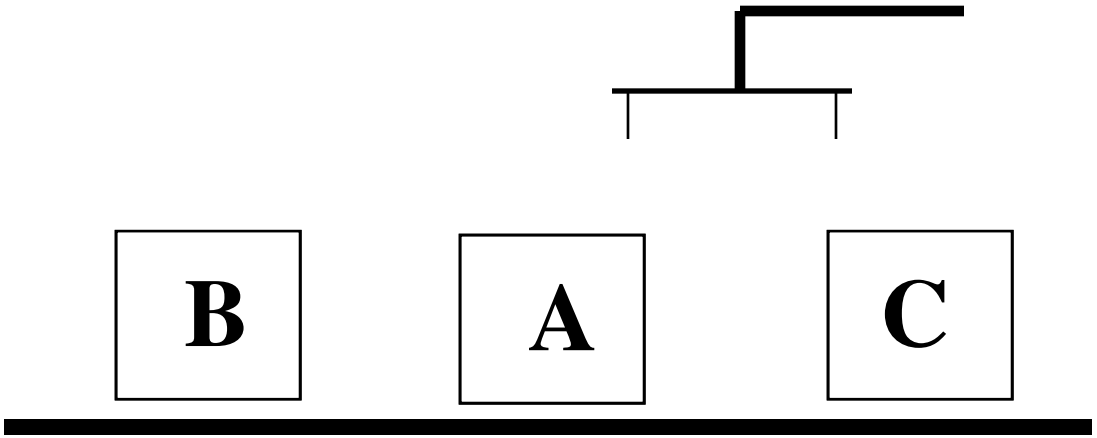


- Asumir que cualquier cosa no estipulada es falsa: *suposición de mundo cerrado*.

The Blocks World

- Un *objetivo* *es representado por un conjunto de fórmulas*
- Aquí hay un objetivo:

$$OnTable(A) \wedge OnTable(B) \wedge OnTable(C)$$

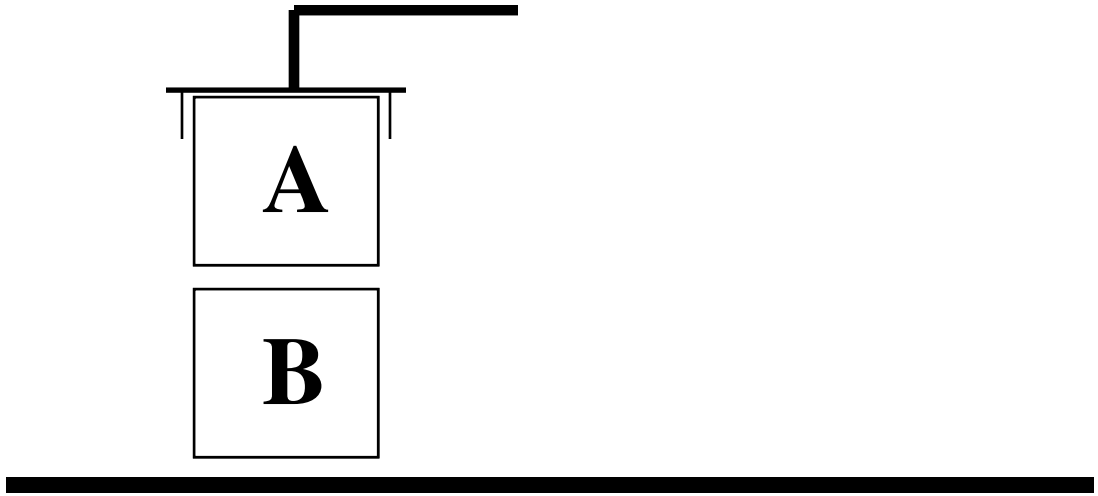


The Blocks World

- Las *acciones* son representadas de la siguiente forma:
- Cada acción tiene:
  - un *nombre*
    que puede tener argumentos
  - Una *lista de pre-condiciones*
    lista de hechos que debe ser verdadera para poder ejecutar una acción
  - una *lista de borrado (delete list)*
    lista de hechos que no son verdaderos después que la acción se ejecutó
  - Una *lista de agregación (add list)*
    lista de hechos que se vuelven verdaderos al ejecutar la acción

  Cada uno de ellos puede contener *variables*

Operadores del Blocks World



- Ejemplo 1:
  La acción *stack* ocurre cuando el brazo pone el objeto x *que tiene asido encima del* objeto y.

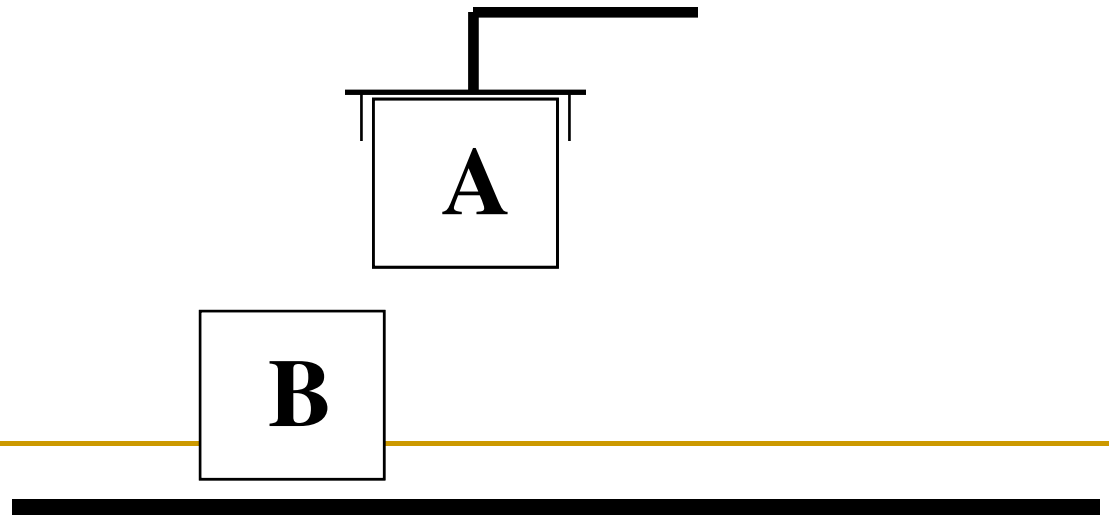
	$Stack(x, y)$
pre	$Clear(y) \wedge Holding(x)$
del	$Clear(y) \wedge Holding(x)$
add	$ArmEmpty \wedge On(x, y)$

Operadores del Blocks World

- Ejemplo 2:
  La acción *unstack* ocurre cuando el brazo toma un objeto x que está encima de otro objeto y.

	$UnStack(x, y)$
pre	$On(x, y) \wedge Clear(x) \wedge ArmEmpty$
del	$On(x, y) \wedge ArmEmpty$
add	$Holding(x) \wedge Clear(y)$

- Stack y UnStack son acciones *inversas*.



Operadores del Blocks World

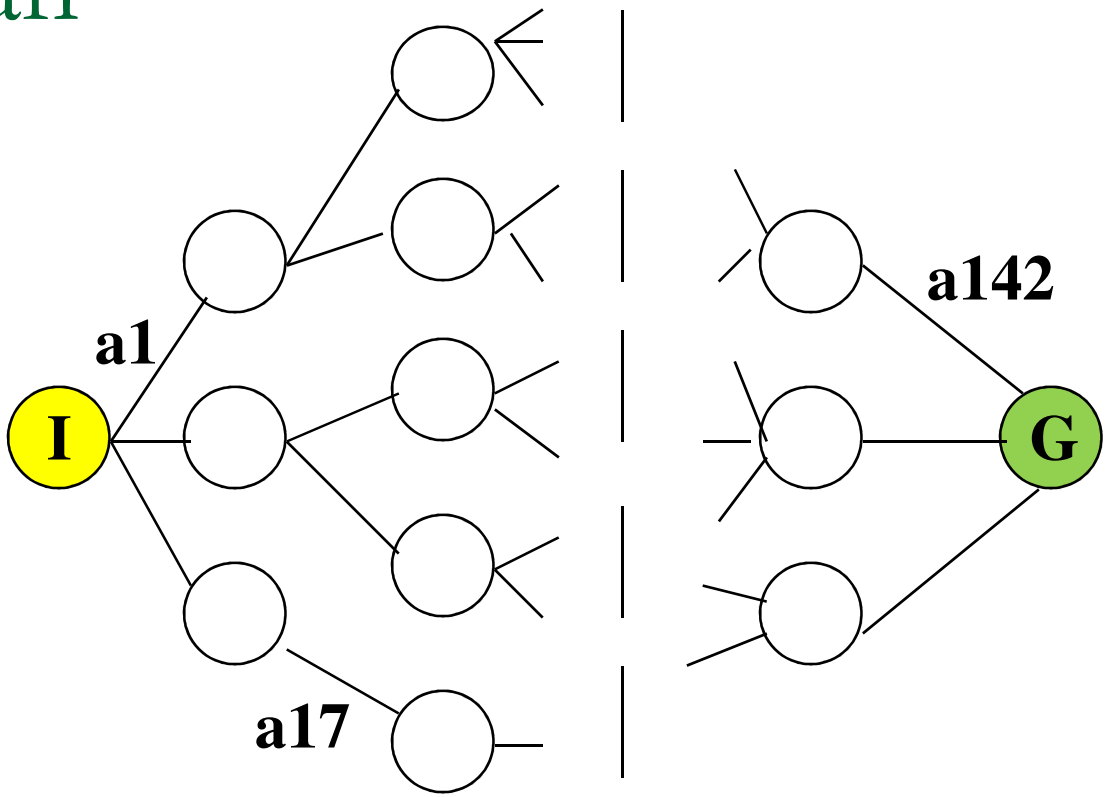
- Ejemplo 3:
  La acción *pickup* ocurre cuando el brazo levanta un objeto x desde encima de la mesa.

	$Pickup(x)$
pre	$Clear(x) \wedge OnTable(x) \wedge ArmEmpty$
del	$OnTable(x) \wedge ArmEmpty$
add	$Holding(x)$

- Ejemplo 4:
  La acción *putdown* ocurre cuando el brazo pone un objeto x sobre la mesa.

	$Putdown(x)$
pre	$Holding(x)$
del	$Holding(x)$
add	$Clear(x) \wedge OnTable(x) \wedge ArmEmpty$

Un Plan



- ¿Qué es un plan?

  Una secuencia (lista) de acciones, con variables reemplazadas por constantes.

La idea básica para lograr objetivos

- Poner el objetivo en el stack de objetivos:

- Considerando el objetivo 1, poner sobre él sus subobjetivos:

- Tratar de lograr subobjetivo GoalS1-2, y continuar…

Implementación de Agentes Razonadores Prácticos: BDI

- 1era pasada en la implementación de un agente razonador práctico:

```
Agent Control Loop Version 1
1. while true
2.      observe the world;
3.      update internal world model;
4.      deliberate about what intention to achieve next;
5.      use means-ends reasoning to get a plan for the intention;
6.      execute the plan
7. end while
```

- (No analizaremos (2) o (3))

Implementación de Agentes RP

- Problema: deliberación y razonamiento medios-fin no son instantáneos: Tienen un *costo en tiempo*.

- Suponga que el agente parte deliberando en t_0, comienza a razonar m-f en t_1, y a ejecutar el plan en t_2. Tiempo de deliberación es:

$$t_{deliberación} = t_1 - t_0$$

- Y el tiempo para razonar m-f es:

$$t_{mf} = t_2 - t_1$$

Implementación de Agentes RP

- Suponga que la deliberación es óptima, o sea, que la intención seleccionada es lo mejor para el agente (Maximiza la utilidad esperada)

- En el tiempo t_1, el agente ha seleccionado una intención que habría sido óptima si hubiera sido lograda en t_0.

  Pero a menos que $t_{deliberación}$ sea muy pequeño, el agente corre el riesgo de haber seleccionado una intención que no sea óptima cuando el agente la vaya a tratar de lograr.

- Esto se llama *racionalidad calculativa*.

- La deliberación es solo la mitad del problema: el agente debe todavía determinar *cómo* lograr la intención.

Implementación de Agentes RP

- Por eso, este agente tendrá comportamiento óptimo global en las siguientes situaciones:

1. Cuando deliberación y razonamiento m-f toma un tiempo muy pequeño; o

2. Cuando el mundo permanece igual mientras el agente delibera y razona m-f, de modo que las suposiciones respecto a la intención a lograr y la forma de hacerlo permanezcan válidas hasta que el agente haya completado la deliberación y razonamiento m-f; o

3. Cuando una intención que es óptima si se logra en t_0 (instante en que se observa el mundo) permanece óptima hasta t_2 (instante en que el agente encuentra un curso de acción para lograr la intención).

Implementación de Agentes RP

- Formalicemos el algoritmo (BI):

```
Agent Control Loop Version 2
1.   B := B0; /* initial beliefs */
2.   while true do
3.        get next percept ρ;
4.        B := brf(B, ρ);
5.        I := deliberate(B);
6.        π := plan(B, I);
7.        execute(π)
8.   end while
```

Deliberación

- ¿Cómo delibera un agente?
  - Comienza por tratar de entender qué opciones se encuentran disponibles
  - *Elige entre ellas*, y se *compromete* con algunas de ellas
- Las opciones elegidas se convierten en *intenciones*

Deliberación

- La función de deliberación se puede descomponer en dos distintos componentes:

  - *Generación de opciones*
    en la cual el agente genera un conjunto de posibles acciones alternativas;
    Una función representa esto (*options)*, que toma las creencias e intenciones actuales del agente y de ellas determina un conjunto de opciones (= *deseos*)

  - *filtrado*
    en el cual el agente elige entre alternativas que compiten y se compromete a lograrlas.
    Para seleccionar entre alternativas, el agente usa una función *filter*.

Deliberación

Formalicemos el algoritmo (BDI):

```
Agent Control Loop Version 3
1.
2.   B := B0;
3.   I := I0;
4.   while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       execute(π)
11.  end while
```

Ejercicio práctico: agente BDI

Macroespecificar como agente BDI el: VirtualConveyor © Fetch Robotics

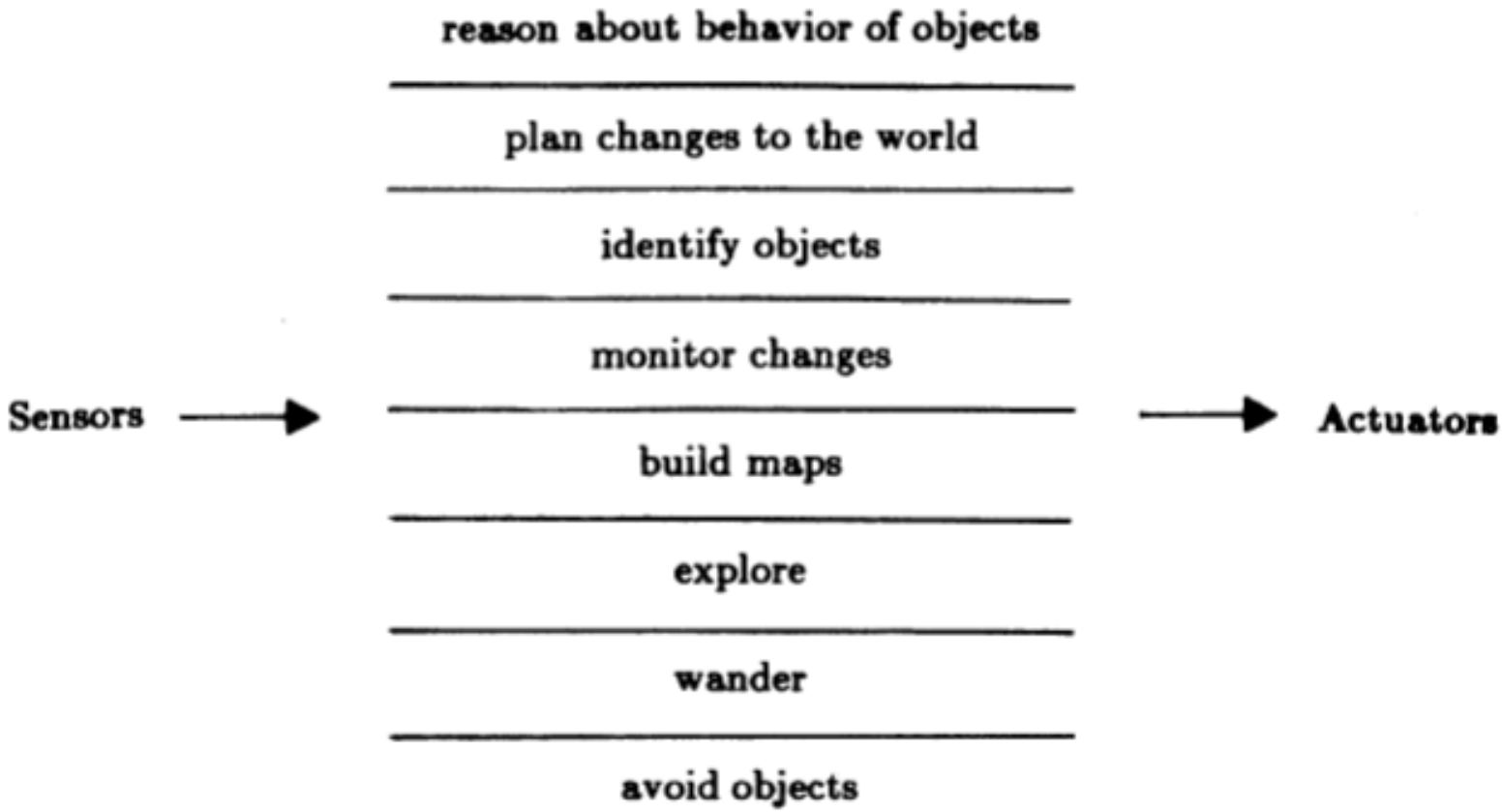
- ¿Qué serían los *B, D, I* y B_0 e I_0 ?
- ¿Cuáles serían los ρ ?
- ¿Qué deberían hacer las funciones *brf*, *options*, *filter* y *plan* ?

```
Agent Control Loop Version 3
1.
2.   B := B0;
3.   I := I0;
4.   while true do
5.        get next percept ρ;
6.        B := brf(B, ρ);
7.        D := options(B, I);
8.        I := filter(B, D, I);
9.        π := plan(B, I);
10.       execute(π)
11.  end while
```

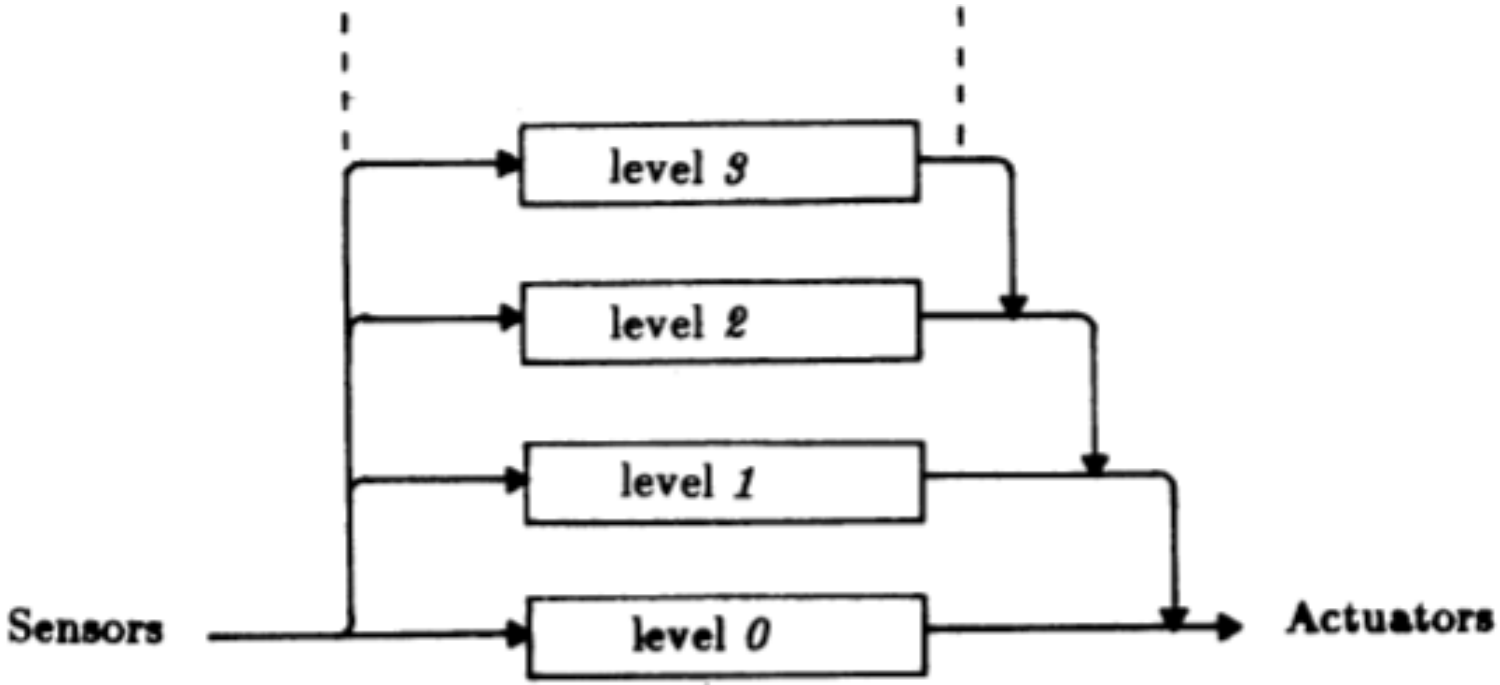
Agentes Reactivos

- Una arquitectura reactiva es una jerarquía de *comportamientos* para realizar una tarea
- Cada comportamiento es una estructura simple parecida a reglas
- Cada comportamiento "compite" con otros para controlar al agente
- Capas inferiores representan comportamientos más primitivos (tal como evitar obstáculos), y tienen precedencia sobre las capas que están más arriba en la jerarquía
- Los sistemas resultantes son extremadamente simples en cuanto a la cantidad de cómputos que deben realizar

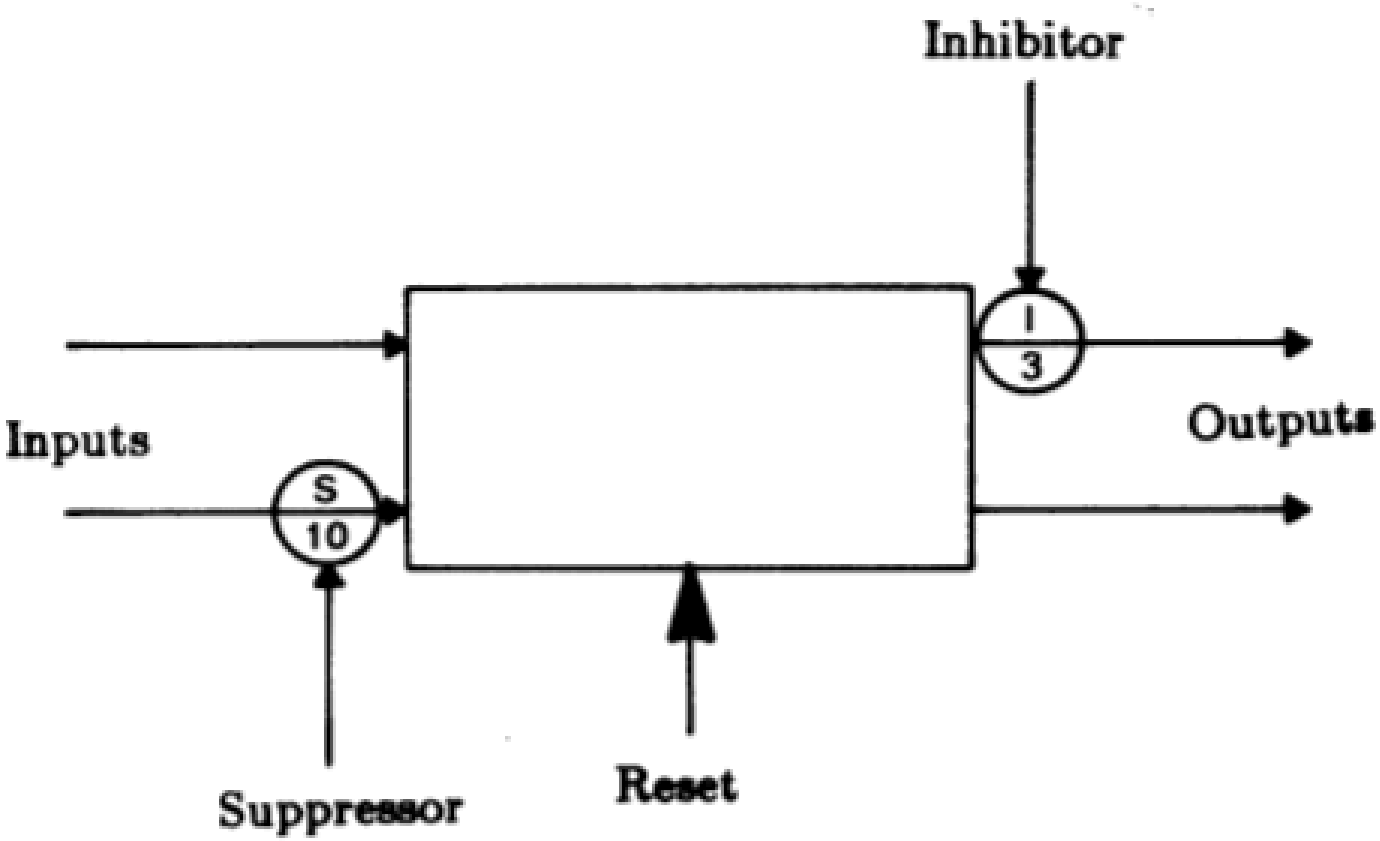
Una Descomposición de un Sistema de Control de un Robot Móvil basada en Comportamientos para efectuar una Tarea



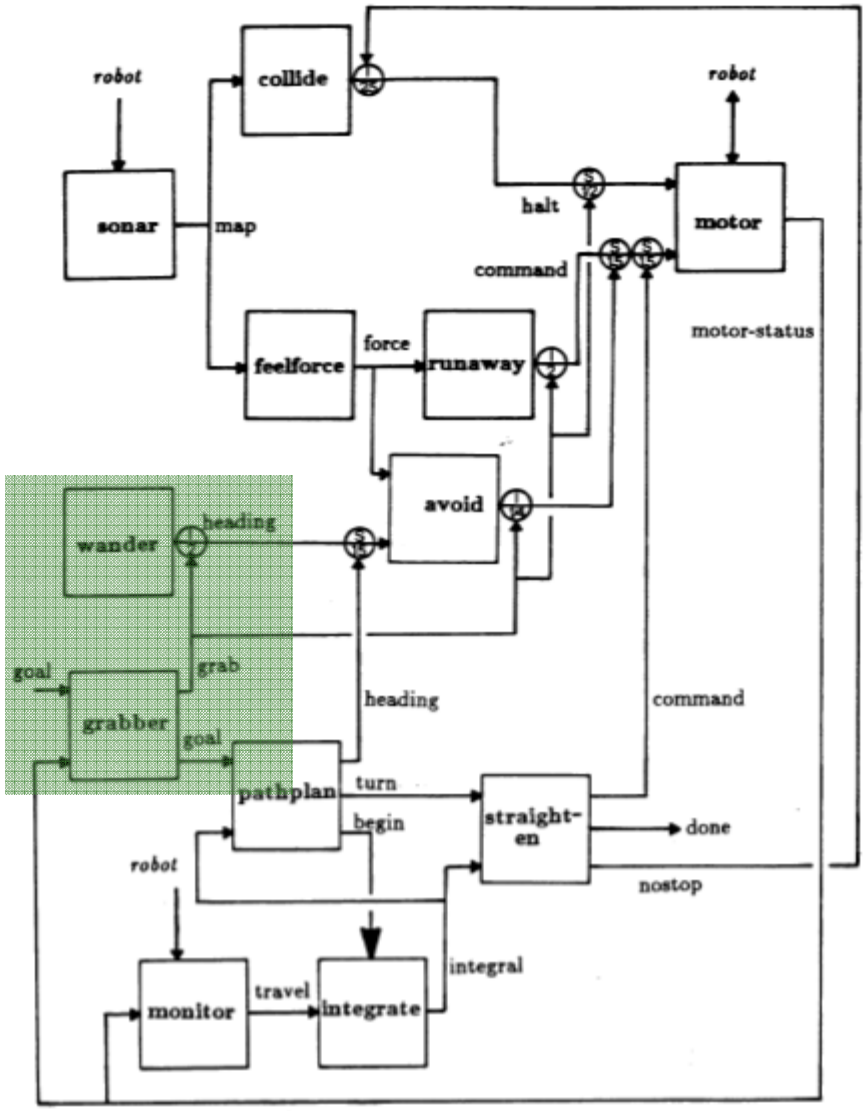
Control por Capas en la Arquitectura de Subsumisión

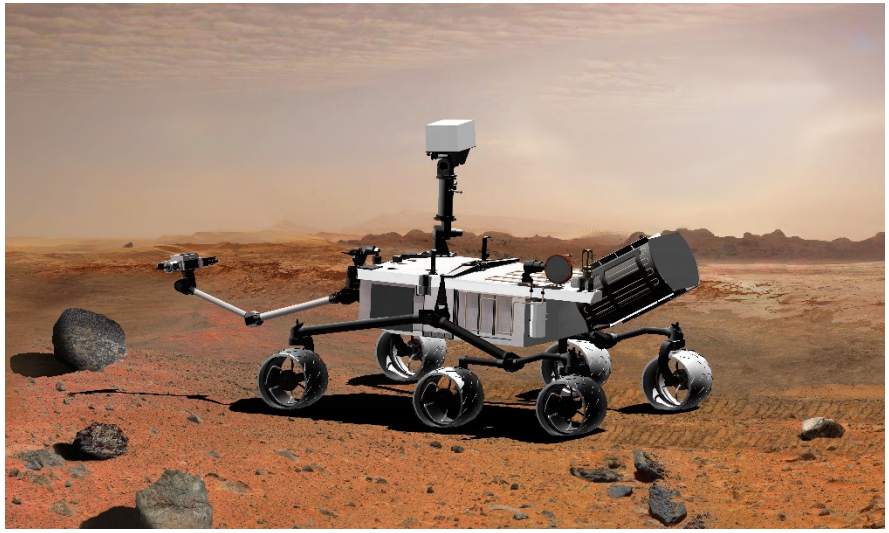


Esquema de un Módulo

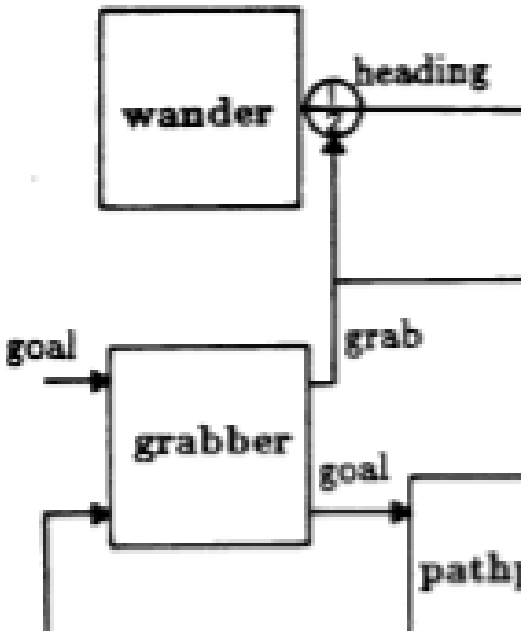


Niveles de un Sistema de Control





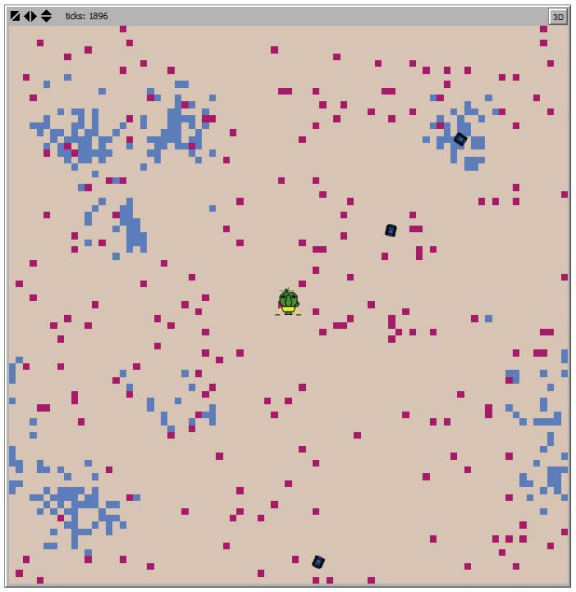
Niveles de un Sistema de Control



El Explorador de Marte de Steels

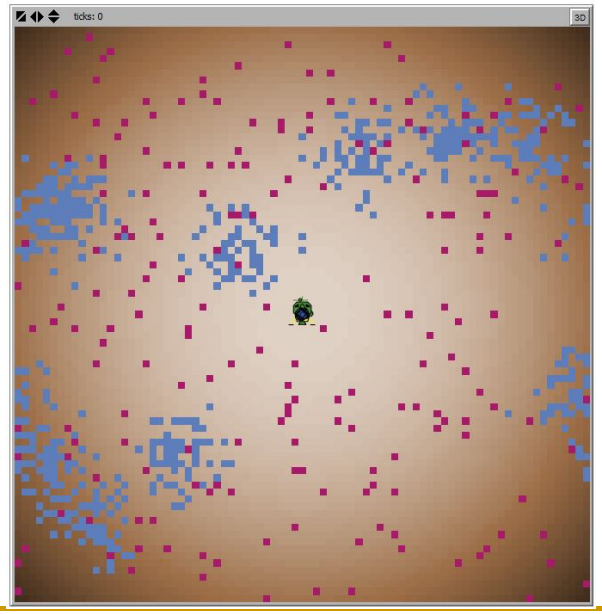
- El explorador de Marte de Steels logra un rendimiento casi óptimo en el dominio simulado de “recoger piedras en Marte”:

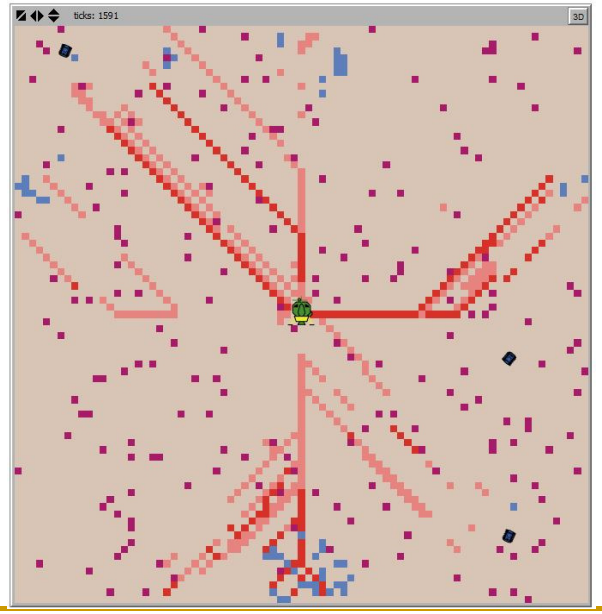
- El objetivo es explorar un planeta distante y recoger muestras de piedras preciosas. La localización de las piedras es desconocida, pero se sabe que tienden a estar agrupadas. La desigualdad del terreno no permite que los robots se comuniquen entre si



El Explorador de Marte de Steels

- La nave madre emite una señal de radio
- Los robots pueden sensar esa señal y dirigirse a la nave navegando la gradiente positiva de la señal
- Cada robot lleva “migajas radioactivas”, que puede dejar caer, recoger y detectar al pasar cerca de ellas





El Explorador de Marte: No Cooperador

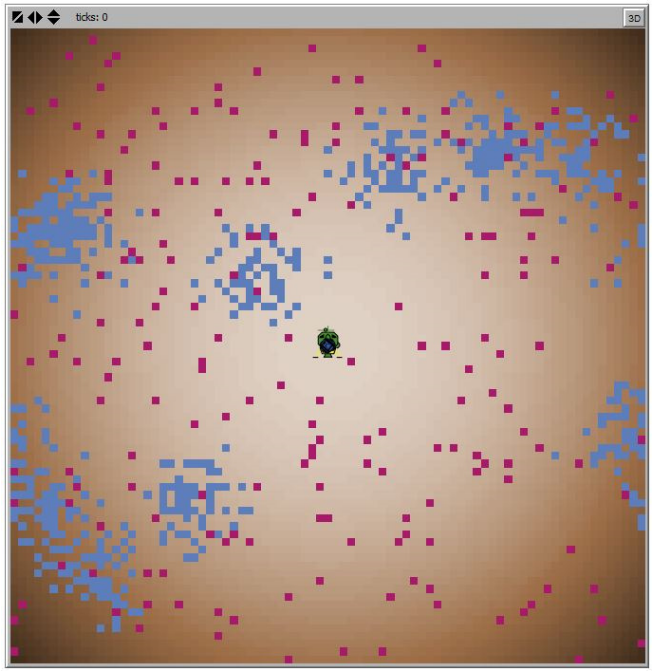
while *(alcanzar objetivo)*

Seleccione par condición-acción con precedencia $1 \prec 2 \prec 3 \prec 4 \prec 5$

1: ***if*** detecta obstáculo ***then*** cambie dirección
2: ***if*** lleva muestras ***and*** en la base ***then*** descargue muestras
3: ***if*** tiene muestras ***and not*** en la base ***then*** navegue gradiente positivo
4: ***if*** detecta una muestra ***then*** recójala
5: ***if true then*** muévase aleatoriamente

Ejecute acción seleccionada

end



El Explorador de Marte: Cooperador

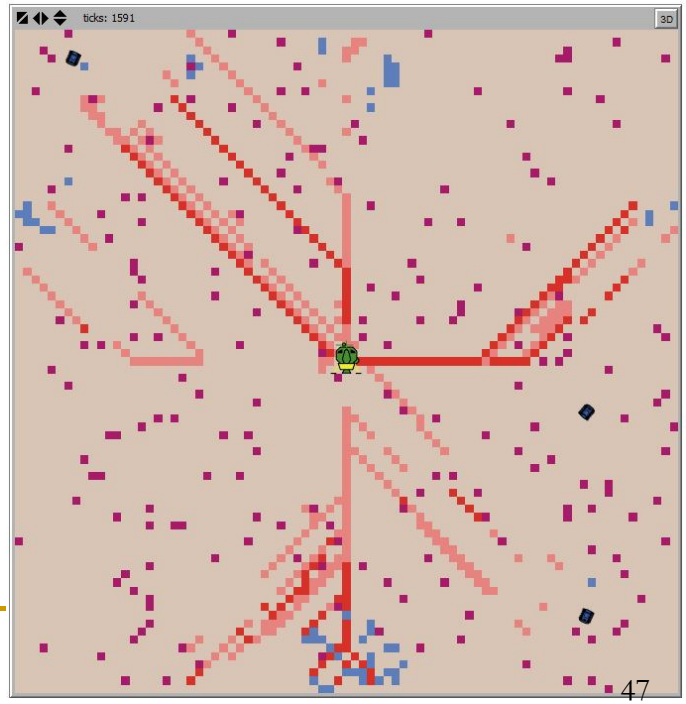
while *(alcanzar objetivo)*

Seleccione par condición-acción con precedencia $1 \prec 2 \prec 3 \prec 4 \prec 5 \prec 6$

1: ***if*** detecta obstáculo ***then*** cambie dirección
2: ***if*** lleva muestras ***and*** en la base ***then*** descargue muestras
3: ***if*** tiene muestras ***and not*** en la base ***then*** navegue gradiente positivo ***and*** suelte 2 migajas
4: ***if*** detecta una muestra ***then*** recójala
5: ***if*** sensa migaja ***then*** recoja 1 migaja ***and*** navegue gradiente negativa
6: ***if true then*** muévase aleatoriamente

Ejecute acción seleccionada

end



Ejercicio práctico: agente reactivo Subsumisión

Macroespecificar como agente Subsumisión el: VirtualConveyor © Fetch Robotics

- Defina el objetivo
- Defina qué pares acción-condición deben existir (tal vez sería más fácil listar acciones y condiciones y luego parearlas)
- Cree la jerarquía de los pares acción-condición

while *(alcanzar objetivo)*

Seleccione par condición-acción con precedencia 1 ≺ 2 ≺ 3 ≺ 4 ≺ .. ≺ n

1: ***if*** c1 ***then*** a1
2: ***if*** c2 ***and*** c3 ***then*** a2
3: ***if*** c2 ***and not*** a3 ***then*** a1 ***and*** a2
4: ***if*** c4 ***or*** c5 ***then*** a4
5: ***if true then*** a6
n:

Ejecute acción seleccionada

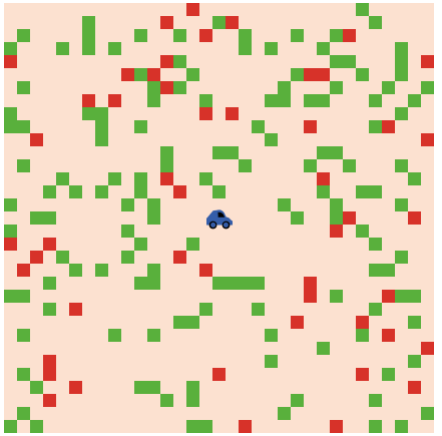
end

Limitaciones de Agentes Reactivos

- Agentes sin un modelo del ambiente deben tener disponible suficiente información del ambiente local
- Si las decisiones son basadas en el ambiente *local*, ¿Cómo toma en cuenta información *no local?* (tiene un enfoque corto placista)
- Es difícil construir agentes reactivos que puedan aprender
- Como el comportamiento emerge de las interacciones de los componentes y el ambiente, es difícil saber como diseñar los agentes (no existe una metodología)
- Es difícil diseñar agentes con un gran número de comportamientos (la dinámica de las interacciones se vuelve muy complicada de entender)

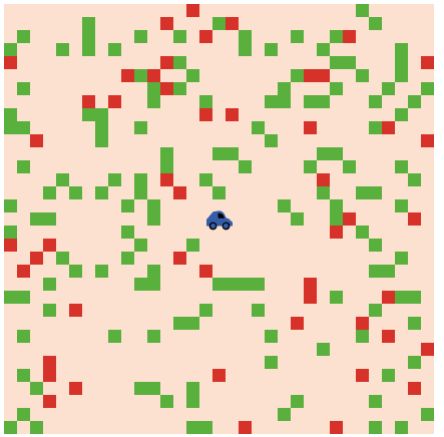
Ejercicio práctico: entender interacciones en agentes reactivos

1. Abra Netlogo v. 5.3.1
2. Desde Netlogo abra el archivo: Subsumption_arq_Real_Situation_Netlogo_v531_1
3. Oprima *Inicializar* y luego *Simular*
4. Aprecie qué pasa con la recolección de muestras (vea el "*mundo*")
5. Oprima nuevamente *Simular*
6. Vea qué pasó en ventana del "*mundo*"
7. Repita acciones 3 al 6 varias veces
8. ¿Cuál es el problema y por qué sucede?



Ejercicio práctico: entender interacciones en agentes reactivos

1. Ahora abra el archivo: Subsumption_arq_Real_Situation_Netlogo_v531_2
2. Oprima *Inicializar* y luego *Simular*
3. Aprecie qué pasa con la recolección de muestras (vea el "*mundo*")
4. Oprima nuevamente *Simular*
5. Vea qué pasó en ventana del "*mundo*"
6. Repita acciones 2 al 5 varias veces
7. ¿Existe ahora el problema anterior?



Ejercicio práctico: entender interacciones en agentes reactivos

Subsumption_arq_Real_Situation_Netlogo_v531_1

while *(NO recoja todas las muestras)*

Seleccione par condición-acción con precedencia $1 \prec 2 \prec 3$

1: ***if*** detecta obstáculo ***then*** evada obstáculo
2: ***if*** detecta una muestra ***then*** recójala
3: ***if*** random 100 < Porcentaje_movimiento_aleatorio ***then*** muévase aleatoriamente ***else*** muévase a la derecha

Ejecute acción seleccionada

end

Subsumption_arq_Real_Situation_Netlogo_v531_2

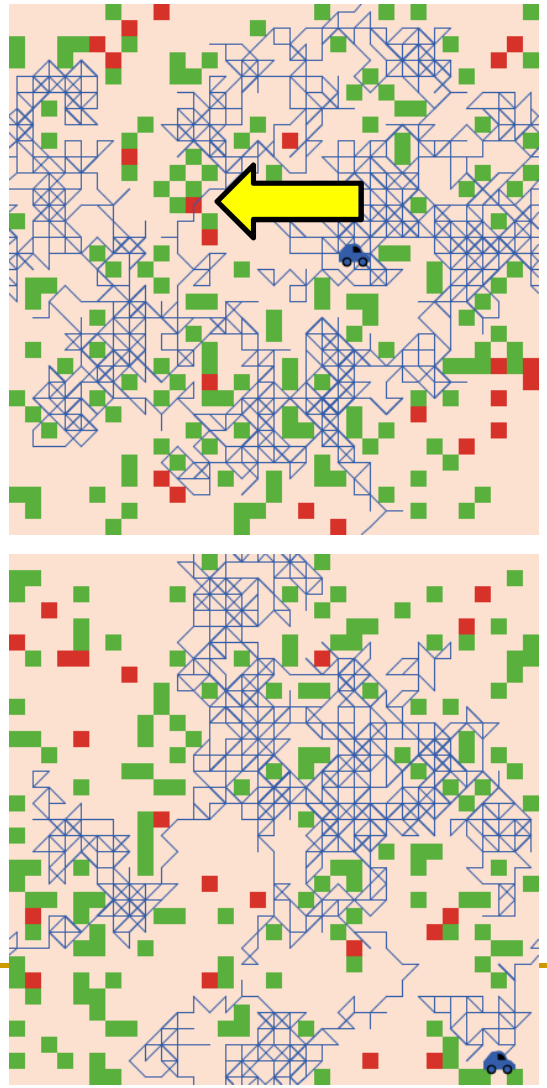
while *(NO recoja todas las muestras)*

Seleccione par condición-acción con precedencia $1 \prec 2 \prec 3$

1: ***if*** detecta una muestra ***then*** recójala
2: ***if*** detecta obstáculo ***then*** evada obstáculo
3: ***if*** random 100 < Porcentaje_movimiento_aleatorio ***then*** muévase aleatoriamente ***else*** muévase a la derecha

Ejecute acción seleccionada

end



Ejercicio práctico: entender interacciones en agentes reactivos

- Ahora abra el archivo: Mars_Explorer_2_Con_Obstaculos_v531
- Corra las siguientes condiciones por 5 veces, anotando el nro. de *ticks* (No of ticks since start simulation) que necesitan los exploradores para recoger las muestras:
  - Cooperative_Agents? Off, no_rocks_to_carry_explorer 1
  - Cooperative_Agents? On, no_rocks_to_carry_explorer 1
  - Cooperative_Agents? Off, no_rocks_to_carry_explorer 10
  - Cooperative_Agents? On, no_rocks_to_carry_explorer 10
- Para eso, establezca las condiciones y oprima *SETUP* y luego *GO SIM*

Ejercicio práctico: entender interacciones en agentes reactivos

- Usando los 5 valores de *ticks* para cada condición, calcular el promedio para cada condición, rellenando la siguiente tabla:

		Nro. de rocas que puede llevar el explorador	
Agentes		**1**	**10**
	Cooperadores	$\overline{t_{1C}}$	$\overline{t_{10C}}$
	No cooperadores	$\overline{t_{1NC}}$	$\overline{t_{10NC}}$

- ¿Qué puede decir de los resultados de dicha tabla?

Arquitecturas Híbridas

- Para muchas situaciones un enfoque completamente deliberativo o reactivo es inadecuado para construir agentes
- Una arquitectura híbrida puede mezclar ambos enfoques
- Se contruye un agente con dos o más subsistemas:
  - Uno deliberativo que desarrolle planes y tome decisiones en la forma propuesta por BDI
  - Uno reactivo, que sea capaz de reaccionar a eventos sin usar un razonamiento complejo
  - Frecuentemente, al componente reactivo se le da algún tipo de precedencia sobre el deliberativo