

# Tema 3: Razonamiento Práctico (II)

Agentes Inteligentes (AIN)

---

---

*Autores: Vicent Botti, Carlos Carrascosa, Vicente Julián*

# Tema 2- Índice

2.1 Arquitecturas de agente.

2.2 Agentes de Razonamiento Deductivo.

    2.2.1 Más Problemas...

    2.2.2 Sistemas de Planificación (en general)

    2.2.3 El Mundo de Bloques

    2.2.4 AGENT0 y PLACA

    2.2.5 Concurrent METATEM

2.3 Agentes Reactivos

    2.3.1 Arquitectura de Subsunción

    2.3.2 Red de Comportamientos para Agentes Situados

2.4 Agentes Híbridos

    2.4.1 Arquitectura Capas Horizontales: TouringMachines

    2.4.2 Arquitectura Capas Verticales: InteRRaP

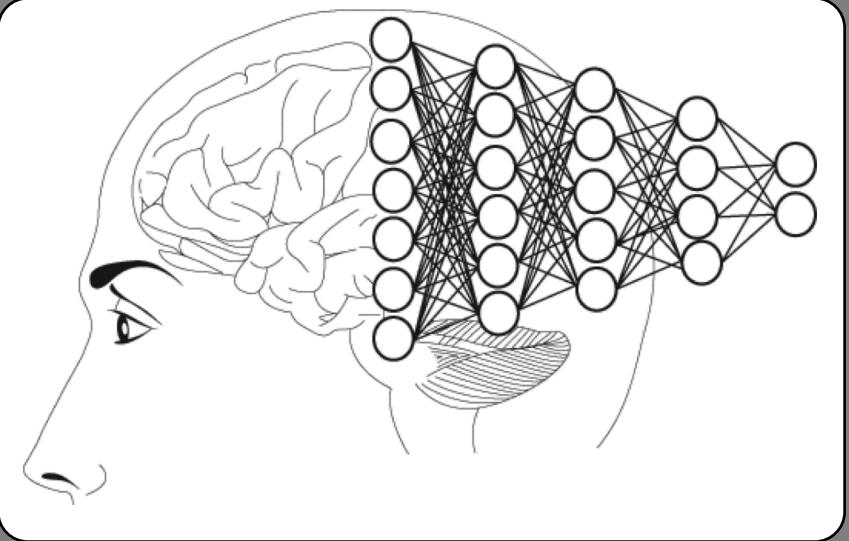
**2.5 Agentes de Razonamiento Práctico.**

**2.5.1 Arquitecturas BDI**

**2.5.2 Razonamiento Dirigido por el Objetivo**

**2.5.4 Implementando Agentes de Razonamiento Práctico: JASON**

2.6 Conclusión



# Razonamiento Práctico

Razonamiento práctico es razonamiento dirigido *por las acciones* - el proceso de descubrir qué hacer:

*El razonamiento práctico es una cuestión de sopesar consideraciones conflictivas por y contra las opciones que compiten, donde las consideraciones relevantes se proveen por lo que el agente desea/valora/le importa y lo que el agente cree. (Bratman)*

El Razonamiento Práctico se distingue del Razonamiento Teórico en que el razonamiento teórico es dirigido por las *creencias*.

# Arquitecturas BDI

- BDI - una teoría de razonamiento práctico - Bratman, 1988
- para *agentes con recursos*
- incluye
  - análisis de objetivos
  - sopesar diferentes alternativas
  - interacción entre estas dos formas de razonamiento
- Conceptos básicos
  - *Beliefs* = información que tiene el agente sobre el mundo
  - *Desires* = estado de los asuntos que el agente desearía alcanzar
  - *Intentions* = deseos (o acciones) que el agente se ha comprometido lograr

# Arquitecturas BDI

BDI particularmente convincente por:

*Componente Filosófico*: basado en una teoría de acciones racionales en humanos

*Arquitectura Software*: ha sido implementado y usado con éxito en un gran número de complejas aplicaciones

IRMA - Intelligent Resource-bounded Machine Architecture

PRS - Procedural Reasoning System

*Componente Lógico*: el modelo ha sido formalizado rigurosamente en una familia de lógicas BDI

Rao & Georgeff, Wooldrige

$(\text{Int } Ai \varphi) \rightarrow \neg (\text{Bel } Ai \varphi)$



# Razonamiento Práctico

Razonamiento Práctico Humano =

Deliberación

+

Razonamiento guiado por los objetivos

Deliberación:

Decidir **qué** estados se desean alcanzar (objetivos).

Razonamiento guiado por objetivos:

Decidir **cómo** lograr esos estados.



# Razonamiento Práctico

- Deliberación
  - El resultado de la deliberación son las *intenciones*.
- Razonamiento guiado por los objetivos
  - El resultado es un plan (secuencia de acciones)
- Arquitectura

# Intenciones en Razonamiento Práctico



Intenciones: los agentes necesitan determinar formas de lograrlas.

Si tengo una intención de I, esperarías que dedicase recursos a decidir cómo lograr I.

Intenciones: son como un *filtro* para adoptar otras intenciones que no entren en conflicto.

Si tengo una intención de I, no esperarías que adoptase una intención Y tal que I y Y sean mútuamente exclusivas.

Los agentes siguen el éxito de sus intenciones, y están inclinados a volver a intentarlas si fallan.

Si el primer intento de un agente de lograr I falla, y manteniéndose el resto de cosas igual, intentará un plan alternativo para lograr I.

# Intenciones en Razonamiento Práctico

Los agentes creen que sus intenciones son posibles.

Crean que hay alguna manera de lograr sus intenciones.

Los agentes no creen que no lograrán sus intenciones.

No sería racional por mi parte adoptar una intención de I, si creyese que I no es posible.

Los agentes creen que conseguirán sus intenciones, bajo ciertas circunstancias.

Si deseo I, entonces creo que bajo circunstancias normales conseguiré I. Normalmente no sería racional por mi parte creer que yo conseguiré siempre mis intenciones; las intenciones pueden fallar. Más aún, no tiene sentido que si creo que I es inevitable lo adopte como una intención.



# Intenciones en Razonamiento Práctico



Los agentes no tienen que planificar todos los efectos colaterales de sus intenciones.

Si yo creo  $I \rightarrow Y$  y yo planifico que  $I$ , no tengo que planificar necesariamente  $Y$  también. (Intenciones no son cerradas bajo la implicación).

Este problema se conoce como **efecto colateral** o problema del **todo en el mismo paquete** (*deal package*).

Yo puedo creer que ir al dentista tiene relacionado dolor, y puede que yo tenga intención de ir al dentista — pero esto no implica que yo tenga la intención de sufrir dolor! .

# Intenciones en Razonamiento Práctico

Hay que notar que las **intenciones** son mucho más fuertes que los meros **deseos**.

*Mi deseo de jugar baloncesto esta tarde es meramente un influenciador potencial de mi conducta esta tarde. Debe competir con mis otros deseos relevantes [...] antes de que se decida lo que haré. Por contra, una vez yo tengo la intención de jugar baloncesto esta tarde, el asunto está decidido: normalmente, no necesitaré seguir sopesando los pros y los contras, sino que procederé a ejecutar mis intenciones.* (Bratman, 1990)



# Agentes Planificadores

Desde ppios. de los 70s, la comunidad de planific. de IA ha estado interesada en el diseño de agentes.

**Planificación es esencialm. prog. autom:** el diseño de un curso de acción que logrará algún objetivo deseado.

Según la comunidad de IA simbólica algún tipo de sist. de planific. será un componente central de cualquier agente.

Se han propuesto muchos algoritmos de planificación y la teoría de planificación ha sido bien desarrollada (basado en los trabajos de Fikes & Nilsson).





# Razonamiento Dirigido por el Objetivo

Idea básica:  
dar a un agente

- representación de objetivo/intención a lograr
- representación de acciones que puede realizar
- representación del entorno
- y hacerle generar un **plan** para lograr el objetivo

Esencialmente esto es  
programación automática



# Razonamiento Dirigido por el Objetivo





# Planificación

Cómo representamos...

- objetivo a ser logrado
- estado del entorno
- acciones disponibles al agente
- el mismo plan

# Tema 2- Índice

2.1 Arquitecturas de agente.

2.2 Agentes de Razonamiento Deductivo.

    2.2.1 Más Problemas...

    2.2.2 Sistemas de Planificación (en general)

    2.2.3 El Mundo de Bloques

    2.2.4 AGENT0 y PLACA

    2.2.5 Concurrent METATEM

2.3 Agentes Reactivos

    2.3.1 Arquitectura de Subsunción

    2.3.2 Red de Comportamientos para Agentes Situados

2.4 Agentes Híbridos

    2.4.1 Arquitectura Capas Horizontales: TouringMachines

    2.4.2 Arquitectura Capas Verticales: InteRRaP

**2.5 Agentes de Razonamiento Práctico.**

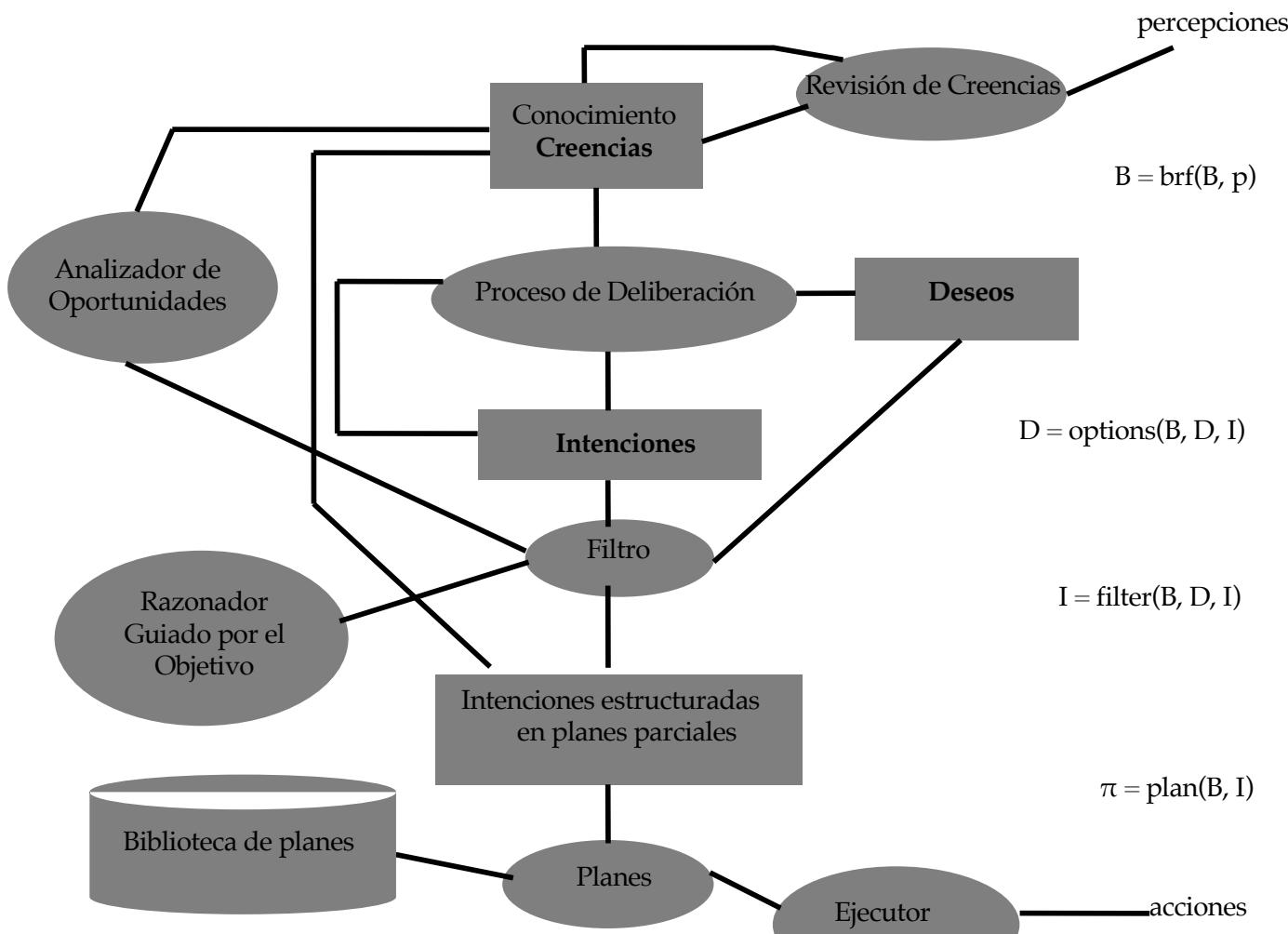
**2.5.1 Arquitecturas BDI**

**2.5.2 Razonamiento Dirigido por el Objetivo**

**2.5.4 Implementando Agentes de Razonamiento Práctico: JASON**

2.6 Conclusión

# Arquitectura BDI



# Agente de Razonamiento Práctico

Bucle de control del agente

*while true*

- ¿cuáles son las opciones (deseos)?
- ¿cómo elegir una opción?
- incl. filtrado

observe the world; opción elegida: intención ...

update internal world model;

deliberate about what intention to achieve next;

*intention;* use means-ends reasoning to get a plan for the

execute the plan

¿cuándo reconsiderar las intenciones?

*end while*

# Implementando Agentes de Razonamiento Práctico

**Problema:** los procesos de deliberación y razonamiento guiado por los objetivos no son instantáneos.

Tienen un coste temporal.

Si el agente comienza a deliberar en  $t_0$ , comienza el razonamiento guiado por objetivos en  $t_1$ , y comienza a ejecutar el plan en  $t_2$ . El tiempo para deliberar es

$$t_{\text{deliberate}} = t_1 - t_0$$

y el tiempo para razonamiento guiado por los objetivos es

$$t_{\text{me}} = t_2 - t_1$$

# Implementando Agentes de Razonamiento Práctico

- La deliberación es óptima si cuando selecciona alguna intención a lograr, ésta es la mejor opción para el agente (Maximiza su Utilidad Esperada).
- Así en el instante  $t_1$ , el agente ha elegido una intención a lograr que habría sido óptima si se hubiese logrado en  $t_0$ . Pero, a menos que  $t_{\text{deliberate}}$  sea muy pequeña, el agente corre el riesgo de que la intención seleccionada no sea ya óptima en el momento en el que el agente se fija en ella.
- Esto es racionalidad calculada.
- Deliberación es sólo la mitad del problema: el agente todavía tiene que determinar **cómo** lograr la intención.

# Implementando Agentes de Razonamiento Práctico

Así, este agente tendrá un comportamiento global óptimo en las siguientes circunstancias:

1. Cuando la deliberación y el razonamiento guiado por el objetivo toman muy poco tiempo; o
2. Cuando se garantiza que el mundo permanecerá estático mientras el agente está deliberando y realizando su razonamiento basado en objetivos, con lo que las suposiciones sobre la elección de qué intención lograr y qué plan usar para lograr la intención permanecen válidas hasta que el agente ha completado la deliberación y el razonamiento dirigido por los objetivos; o
3. Cuando una intención que es óptima si se logra en el instante  $t_0$  (instante en el que se observa el mundo) se garantiza que permanece siendo óptima hasta el instante  $t_2$  (instante en el que el agente ha encontrado un curso de acción para lograr la intención).



# Implementando Agentes de Razonamiento Práctico

Hagamos el algoritmo más formal:

```
Agent Control Loop Version 2
1.  $B := B_0$ ; /* initial beliefs */
2. while true do
3.     get next percept  $\rho$ ;
4.      $B := brf(B, \rho)$ ;
5.      $I := deliberate(B)$ ;
6.      $\pi := plan(B, I)$ ;
7.      $execute(\pi)$ 
8. end while
```

# Deliberación

¿Cómo delibera un agente?

- Comienza intentando entender qué opciones tiene disponibles
- Elige entre ellas, y se compromete con alguna

Opciones elegidas son entonces,  
**intenciones**



# Deliberación

La función *deliberar* puede ser descompuesta en 2 componentes funcionalmente distintos:

## *generación de opciones*

en la que el agente genera un conjunto de posibles alternativas;

Se representa la generación de opciones por una función, *opciones*, que a partir de las creencias e intenciones actuales del agente determina un conjunto de opciones (= *desires*)

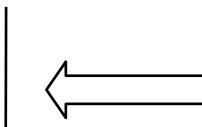
## *filtrado*

en el que el agente elige entre alternativas que compiten y se compromete a lograrlas.

Para seleccionar entre opciones que compiten, un agente usa una función de filtrado.

# Deliberación

Agent Control Loop Version 3

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$  |   
8.        $I := filter(B, D, I);$  |   
9.        $\pi := plan(B, I);$   
10.       $execute(\pi)$   
11. end while
```

# Deliberación

Función *Acción* ( $p : P$ ) :  $A$   
comienzo

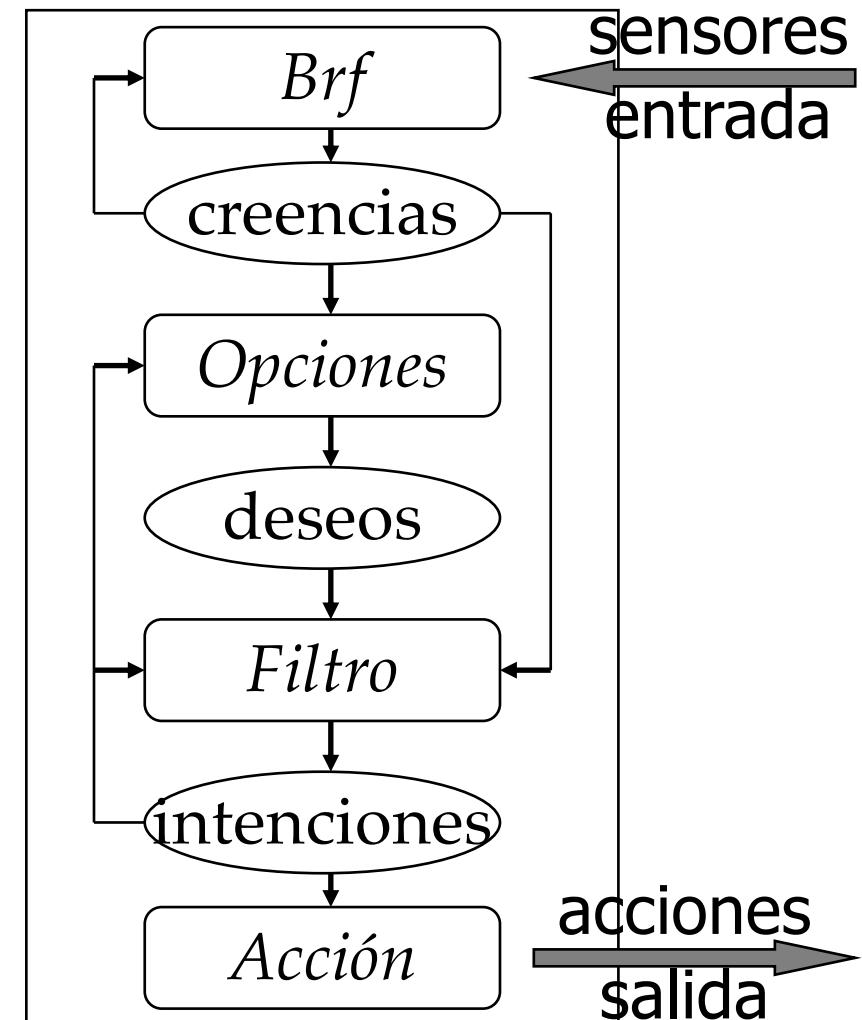
$B := Brf (B, p)$

$D := Opciones (D, I)$

$I := Filtro (B, D, I)$

devolver *Ejecutar* ( $I$ )

fin



# Estrategias de Compromiso

“Some time in the not-so-distant future, you are having trouble with your new household robot. You say “Willie, bring me a beer.” The robot replies “OK boss.” Twenty minutes later, you screech “Willie, why didn’t you bring me that beer?” It answers “Well, I intended to get you the beer, but I decided to do something else.” Miffed, you send the wise guy back to the manufacturer, complaining about a lack of commitment. After retrofitting, Willie is returned, marked “Model C: The Committed Assistant.” Again, you ask Willie to bring you a beer. Again, it accedes, replying “Sure thing.” Then you ask: “What kind of beer did you buy?” It answers: “Genessee.” You say “Never mind.” One minute later, Willie trundles over with a Genessee in its gripper. This time, you angrily return Willie for overcommitment. After still more tinkering, the manufacturer sends Willie back, promising no more problems with its commitments. So, being a somewhat trusting customer, you accept the rascal back into your household, but as a test, you ask it to bring you your last beer. Willie again accedes, saying “Yes, Sir.” (Its attitude problem seems to have been fixed.) The robot gets the beer and starts towards you. As it approaches, it lifts its arm, wheels around, deliberately smashes the bottle, and trundles off. Back at the plant, when interrogated by customer service as to why it had abandoned its commitments, the robot replies that according to its specifications, it kept its commitments as long as required — commitments must be dropped when fulfilled or impossible to achieve. By smashing the bottle, the commitment became unachievable.”

# Estrategias de Compromiso

Si una opción ha superado con éxito la función filtrado y es elegida por el agente como intención, *el agente se compromete con esa opción* (no será abandonada inmediatamente, implica una persistencia temporal de las intenciones)

*Estrategias de compromiso* más comunes:

- ***Compromiso Ciego***

Un agente comprometido ciegamente continuará manteniendo una intención hasta que crea que la ha logrado. *Compromiso Ciego* es también referido a veces como *Compromiso Fanático*.

- ***Compromiso Inquebrantable***

Un agente inquebrantable continuará manteniendo una intención hasta que crea que o bien ha logrado la intención, o bien ya no es posible lograrla.

- ***Compromiso Sin Prejuicios***

Un agente sin prejuicios mantendrá una intención mientras la crea posible.

# Estrategias de Compromiso

Un agente se compromete tanto

- a los *fines* (por ej., los deseos que quiere lograr),
- como a los *medios* (por ej., el mecanismo por el cual desea lograr sus objetivos).

Actualmente, nuestro bucle de control del agente está demasiado comprometido, tanto a medios como a fines.

Modificación: *replanificar* siempre que un plan vaya mal

## Agent Control Loop Version 4

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not empty( $\pi$ ) do  
11.           $\alpha := hd(\pi);$   
12.          execute( $\alpha$ );  
13.           $\pi := tail(\pi);$   
14.          get next percept  $\rho$ ;  
15.           $B := brf(B, \rho);$   
16.          if not sound( $\pi, I, B$ ) then  
17.               $\pi := plan(B, I)$   
18.          end-if  
19.      end-while  
20.  end-while
```

← *Reactividad,  
Replanificación*

“Compromiso Ciego”

# Estrategias de Compromiso

Todavía demasiado comprometido a las intenciones: Nunca para a considerar si sus intenciones son apropiadas o no.

Modificación: parar a determinar si las intenciones han tenido éxito o si son imposibles:

*(Compromiso Inquebrantable)*

## Agent Control Loop Version 5

```
2.    $B := B_0;$ 
3.    $I := I_0;$ 
4.   while true do
5.       get next percept  $\rho$ ;
6.        $B := brf(B, \rho);$ 
7.        $D := options(B, I);$ 
8.        $I := filter(B, D, I);$ 
9.        $\pi := plan(B, I);$ 
10.      while not ( $empty(\pi)$ 
11.          or  $succeeded(I, B)$ 
12.          or  $impossible(I, B)$ ) do
13.           $\alpha := hd(\pi);$ 
14.           $execute(\alpha);$ 
15.           $\pi := tail(\pi);$ 
16.          get next percept  $\rho$ ;
17.           $B := brf(B, \rho);$ 
18.          if not  $sound(\pi, I, B)$  then
19.               $\pi := plan(B, I)$ 
20.          end-if
21.      end-while
22.  end-while
```

Compromiso inquebrantable

*Abandonar intenciones que son imposibles o han tenido éxito*

← *Reactividad, Replanificación*

# Reconsideración de Intenciones

- Nuestro agente reconsidera sus intenciones cada vez que está en el bucle exterior, cuando:
  - ha *ejecutado completamente un plan* para lograr sus intenciones actuales; o
  - cree que *ha logrado sus intenciones actuales*; o
  - cree que sus *intenciones actuales ya no son posibles*.
- Esto permite a un agente *reconsiderar* sus intenciones
- Modificación: Reconsiderar intenciones después de ejecutar cada acción

*Compromiso de Mente Abierta*

## Agent Control Loop Version 6

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not ( $empty(\pi)$   
11.              or  $succeeded(I, B)$   
12.              or  $impossible(I, B)$ ) do  
13.           $\alpha := hd(\pi);$   
14.           $execute(\alpha);$   
15.           $\pi := tail(\pi);$   
16.          get next percept  $\rho$ ;  
17.           $B := brf(B, \rho);$   
18.           $D := options(B, I);$   
19.           $I := filter(B, D, I);$   
20.          if not  $sound(\pi, I, B)$  then  
21.               $\pi := plan(B, I)$   
22.          end-if  
23.      end-while  
24.  end-while
```

*Compromiso de  
Mente Abierta*

# Reconsideración de Intenciones

Pero la reconsideración de intenciones es muy *costosa*!

Un dilema:

- un agente que no para a reconsiderar sus intenciones muy a menudo continuará intentando lograr sus intenciones incluso después de que esté claro que no pueden ser logradas, o de que ya no haya ninguna razón para lograrlas
- un agente que *constantemente* reconsidera sus intenciones puede dedicarle realmente demasiado poco tiempo para lograrlas, y por lo tanto corre el riesgo de realmente no lograrlas nunca

Solución: incorporar un componente explícito de *control meta-nivel*, que decida si debe o no reconsiderarlas

## Agent Control Loop Version 7

```
1.  
2.    $B := B_0;$   
3.    $I := I_0;$   
4.   while true do  
5.       get next percept  $\rho$ ;  
6.        $B := brf(B, \rho);$   
7.        $D := options(B, I);$   
8.        $I := filter(B, D, I);$   
9.        $\pi := plan(B, I);$   
10.      while not (empty( $\pi$ )  
           or succeeded( $I, B$ )  
           or impossible( $I, B$ )) do  
11.           $\alpha := hd(\pi);$   
12.          execute( $\alpha$ );  
13.           $\pi := tail(\pi);$   
14.          get next percept  $\rho$ ;  
15.           $B := brf(B, \rho);$            Control meta-nivel  
16.          if reconsider( $I, B$ ) then ←  
17.               $D := options(B, I);$   
18.               $I := filter(B, D, I);$   
19.          end-if  
20.          if not sound( $\pi, I, B$ ) then  
21.               $\pi := plan(B, I)$   
22.          end-if  
23.      end-while  
24.  end-while
```

# Interacciones Posibles

Las interacciones posibles entre control meta-nivel y deliberación son:

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	<i>reconsider(...)</i> optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

# Reconsideración de Intenciones

Situación (1): el agente no eligió deliberar y, como consecuencia, no eligió cambiar intenciones. Más aún, si *hubiese* elegido deliberar, tampoco habría cambiado intenciones. En esta situación, la función reconsider(...) se comporta óptimamente.

Situación (2): el agente no eligió deliberar, pero si lo hubiese hecho, *habría* cambiado intenciones. En esta situación, la función reconsider(...) no se comporta óptimamente.

Situación (3), el agente eligió deliberar, pero no cambió sus intenciones. En esta situación, la función reconsider(...) no se comporta óptimamente.

Situation (4), el agente eligió deliberar, y cambió sus intenciones. En esta situación, la función reconsider(...) se comporta óptimamente.

Una suposición importante: el coste de reconsider(...) es *mucho* menor que el coste del proceso de deliberación mismo.

# Reconsideración de Intenciones Óptima

Kinny y Georgeff investigaron experimentalmente la efectividad de las estrategias de reconsideración de intenciones

Usaron dos tipos diferentes de estrategias de reconsideración:

- agentes *atrevidos*  
nunca paran a reconsiderar intenciones, y
- agentes *cautos*  
después de cada acción, se paran a reconsiderar

El *dinamismo* en el entorno es representado por el *ratio de cambio en el mundo*,  $\gamma$

# Reconsideración de Intenciones Óptima

Resultados (no sorprendentes):

- Si  $\gamma$  es baja (el entorno no cambia muy rápido), los agentes atrevidos funcionan bien comparados con los cautos.

Esto es porque los cautos pierden tiempo reconsiderando sus compromisos mientras que los agentes atrevidos están ocupados intentando conseguir (y logrando) sus intenciones.

- Si  $\gamma$  es alta (el entorno cambia frecuentemente), los agentes cautos tienden a superar a los agentes atrevidos.

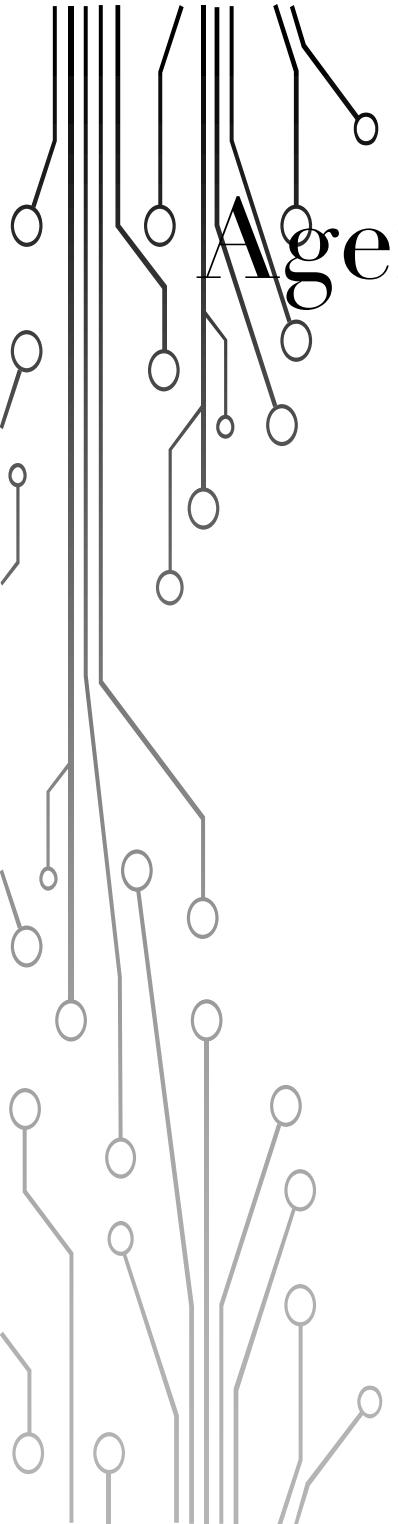
Esto es porque son capaces de reconocer cuando las intenciones van a fracasar, y también aprovechar las situaciones inesperadas y nuevas oportunidades cuando aparecen.

# BDI: Teoría y Práctica

- Consideremos la *semantica* de las arquitecturas BDI: ¿cuánto satisface un agente BDI una *teoría de agentes*?
- Para dotar de semántica a las arquitecturas BDI, Rao & Georgeff han desarrollado una *lógica BDI*: lógica no-clásica con conectivas modales para representar creencias, deseos, e intenciones
- La ‘lógica BDI Básica’ de Rao & Georgeff es una extensión cuantificada de la expresiva lógica temporal ramificada CTL\*
- La estructura semántica subyacente es un framework *temporal ramificado etiquetado*

# Agentes de Razonamiento Práctico

- IRMA
- PRS (Procedural Reasoning System)
- Jadex (java)
- Jason (java – sourceforge.net)
- JAM (java)
- JACK (java)
- ...

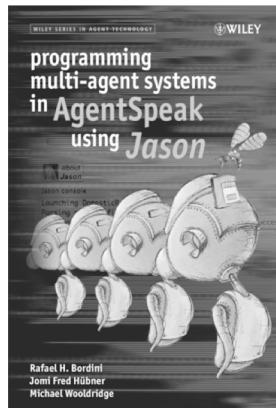


# Implementando Agentes de Razonamiento Práctico

## JASON

<http://JASON.SOURCEFORGE.NET/>

# Bibliografía para Jason



Programming Multi-Agent Systems in AgentSpeak using Jason

Rafael H. Bordini, Jomi Fred Hübner, Michael Wooldridge

ISBN: 978-0-470-02900-8

<http://jason.sourceforge.net/wp/>

# Jason

- El lenguaje que interpreta es una extensión de AgentSpeak (arquitectura BDI).
- Posibilidad de usar distintas infraestructuras de agentes (Jade, SACI, Magentix2, ...)
- Numerosas características configurables por el usuario.
- Disponible en código abierto.
- Vamos a ver su **Lenguaje**

# Elementos del Lenguaje

Los elementos fundamentales del Lenguaje son:

- Creencias (Beliefs)
- Objetivos (Goals)
- Planes (Plans, Intentions)

# Beliefs

Cada agente tiene una base de creencias.

Beliefs: Colección de literales representados como predicados.

tall(jhon)

likes(jhon, music)

Anotaciones: detalles asociados a una creencia.

busy(jhon)[expires(autum)]

Las anotaciones aportan '*elegancia*' al lenguaje y facilitan el manejo de la base de creencias.

# Beliefs

Existen anotaciones que tienen un significado especial para el interprete. En particular la anotación *source*

Hay tres tipos de fuentes de información para los agentes

- Información perceptual: aquella que percibe del entorno - source(percept)
- Comunicación: aquella que proviene de otro agente del sistema - source(id agente)
- Notas mentales: creencias que provienen del propio agente - source(self)

Ejemplos:

- Perceptual (colour(box1,blue)[source(percept)])
- Comunicación (colour(box1,blue)[source(bob)])
- Notas mentales (colour(box1,blue)[source(Self)])

# Beliefs

StrongNegation: se denota con “~”. Expresa que el agente cree explícitamente que algo es falso.

```
~colour(box1, white)[source(john)]
```

Reglas: Permiten inferir nueva información a partir de conocimiento que se tiene:

```
likely_colour(C,B)
:-colour(C,B)[source(S)] &
(S == self | S == percept).
```

```
likely_colour(C,B)
:-colour(C,B)[degOfCert(D1)] &
not (colour(_,B)[degOfCert(D2)] & D2 > D1) &
not ~colour(C,B).
```

# Goals

Existen dos tipos de objetivos en Jason:

Achievement goals(operador !): Expresan un estado del mundo que el agente desea conseguir.

`!own(house)`

Test goals(operador ?): Usados normalmente para recuperar información de la base de creencias.

`?bank_balance(BB)`

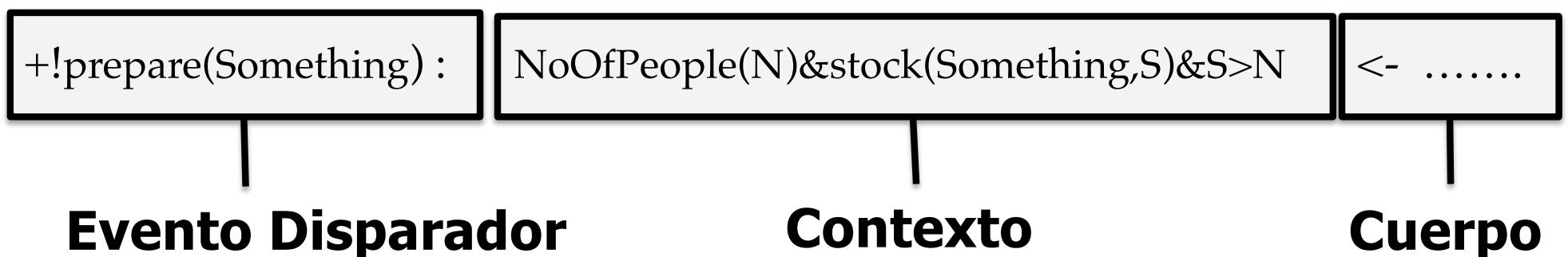
# Plans

Un plan tiene tres partes

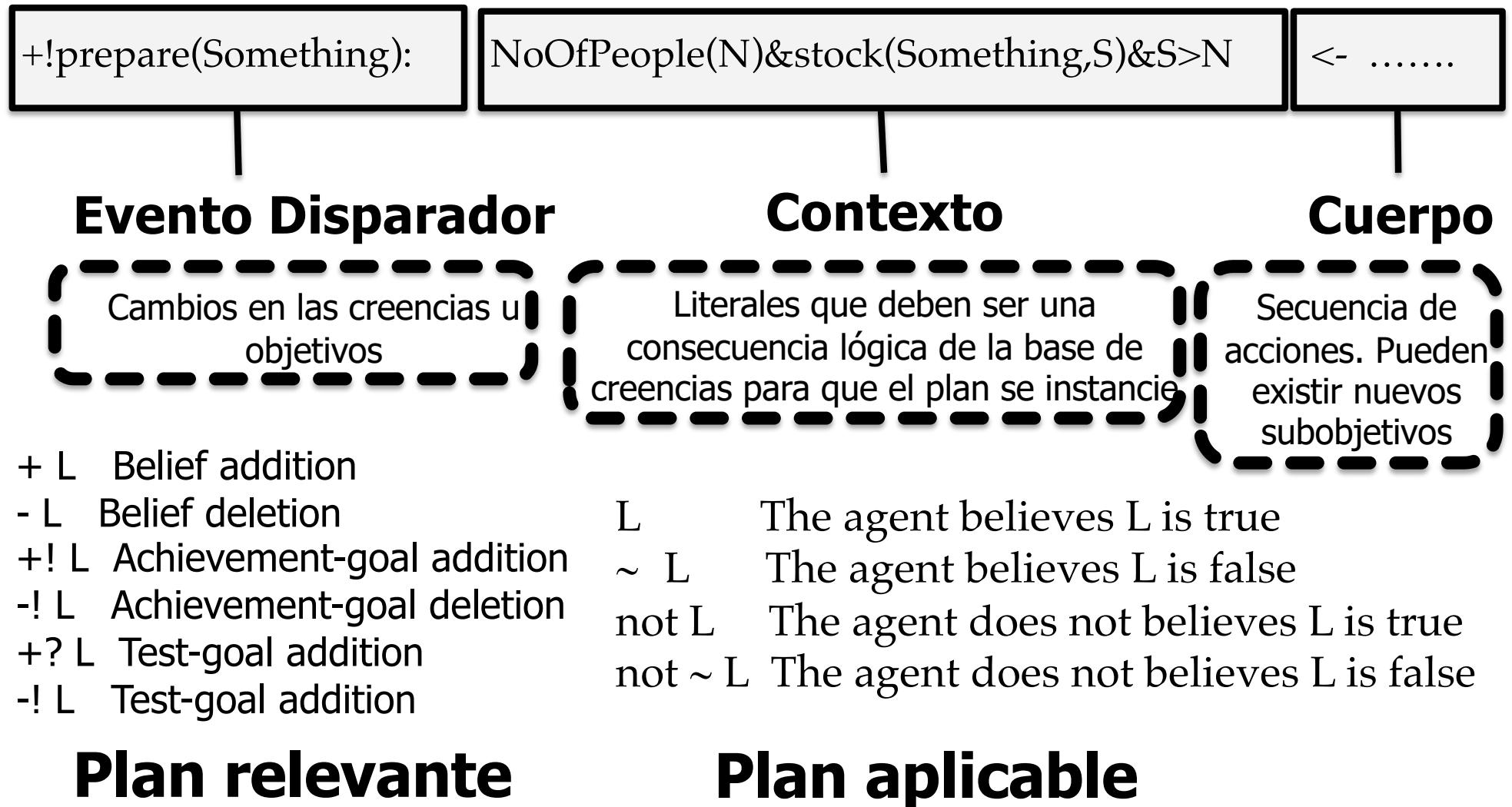
Triggering event: Cambios en las creencias o objetivos del agente

Context: Determinan si un plan es aplicable

Body: Sucesión de acciones que comporta el plan



# Plans



# Plans - Cuerpo

**Cuerpo del plan:** Secuencia de instrucciones separadas por “;”. Estas instrucciones puedes ser:

**Actions:** Acciones externas que se denotan por un predicado. Proporcionan un “feedback”.

rotate(leftarm, 45)

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

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

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

?coords(Tarjet,X,Y).

# Plans - Cuerpo

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

+currenttargets(NumTargets); [Se añade]

-+currenttargets(NumTargets); [Se modifica]

**InternalActions:** Son acciones que no modifican el entorno. Se diferencian de acciones del entorno por el carácter “.”.

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

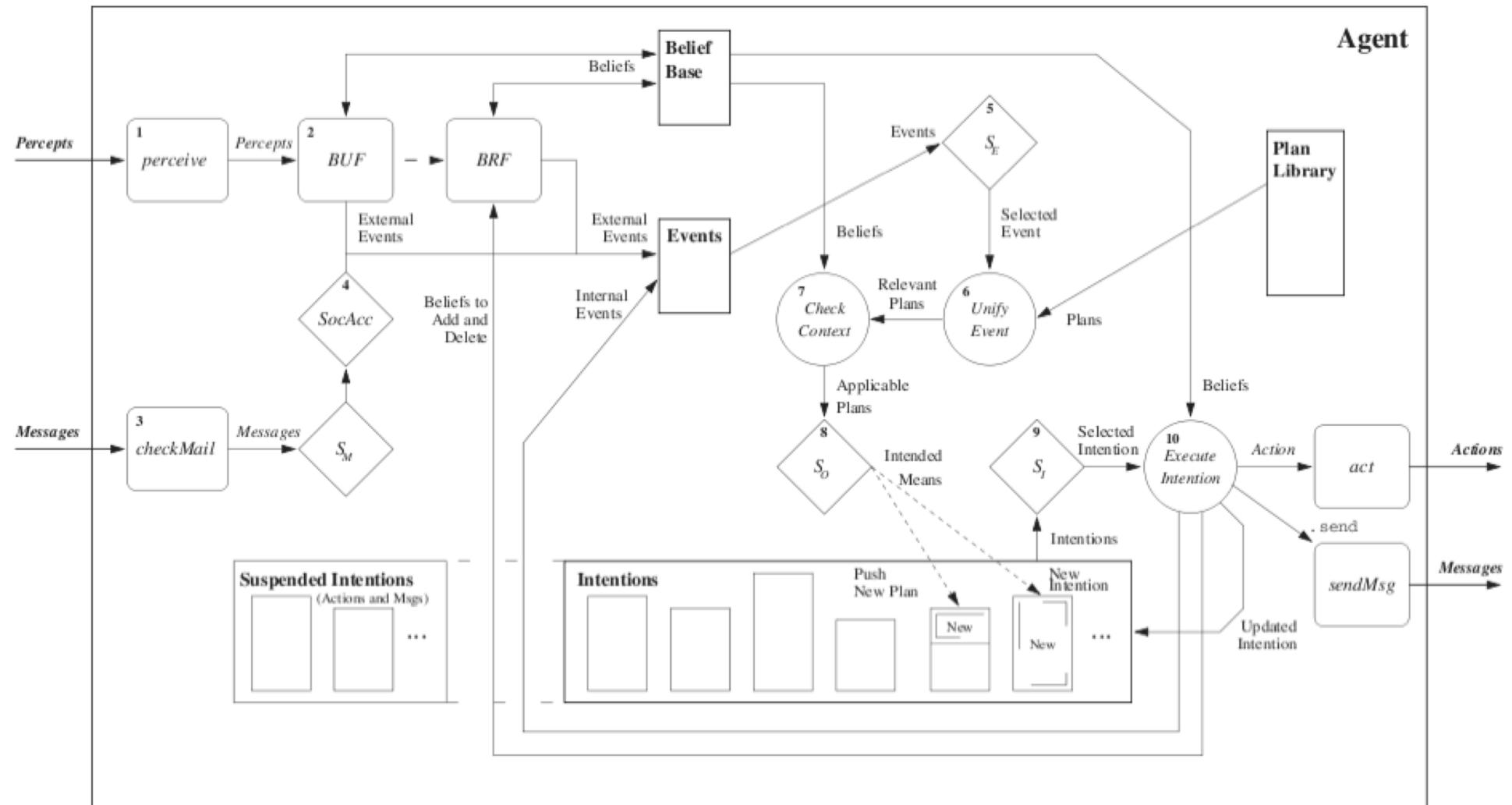
**Expressions:** Su sintaxis es semejante a la de Prolog.

$$X \geq Y^*2$$

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

@labelte: ctxt<-body.

# Arquitectura de Agente



# Fallo de un Plan

Un plan puede fallar por tres causas principales:

- Falta de planes relevantes o aplicables para un ‘achievement goal’
- Fallo de un ‘test goal’
- Fallo de una acción

Cuando un plan falla se genera un evento  $-\text{!g(goal deletion event)}$  si se generó por la adición de un ‘achievement goal’ o ‘test goal’.

El plan que se dispare por el fallo se añade a la pila de intenciones del plan que ha fallado.

# Comunicación

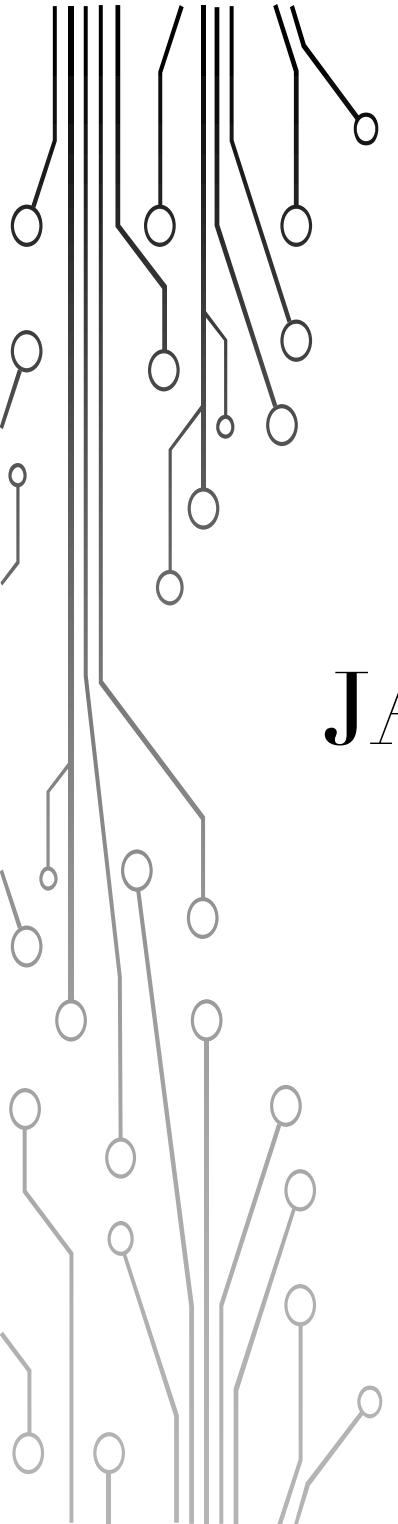
Para cada creencia se anota su origen (id, self, percept)

Cada agente posee una mailbox M y una función de selección de mensajes de M

Performativas:

- tell
- untell
- archive
- unarchive
- tellHow
- untellHow
- askIf
- askAll
- askHow

Ej: send(owner,tell,msg(M))



# JASON: Ejemplos. Ejercicios

# Ej.1Hello World

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

# Ej 2. Factorial

```
/* Creencias iniciales */
```

```
fact(0,1).
```

```
/* Planes */
```

```
+fact(X,Y) : X < 5
```

```
<- +fact(X+1, (X+1) * Y ).
```

```
+fact(X,Y) : X = 5
```

```
<- .print("fact 5 == ", Y ).
```

# Ej 2. Factorial

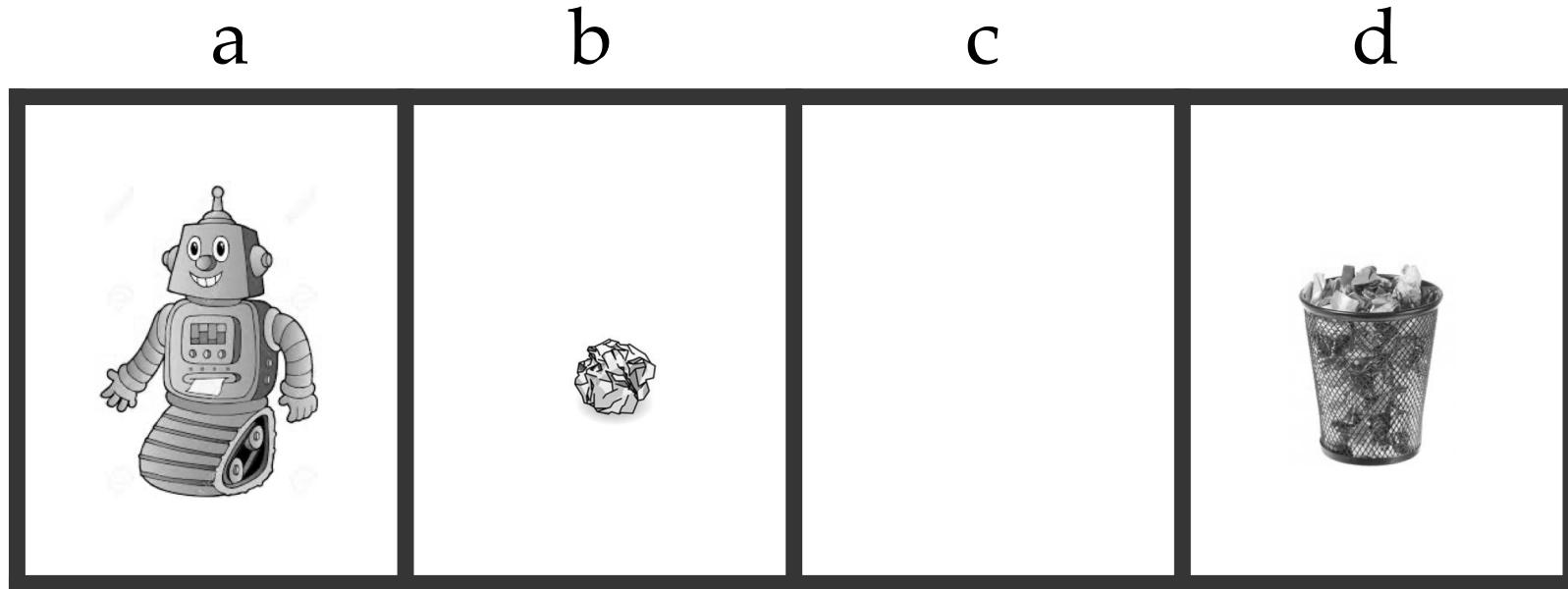
- Modificaciones:
  1. Imprime el valor del factorial en todas las iteraciones.
  2. ¿En qué orden se imprime? ¿Cómo conseguir que sea el inverso?
  3. Prueba otras funciones internas:
    - .stopMAS
    - .wait(<milisegundos>)
    - .random(X)
  4. Modifica el ejemplo para que se pause 3 segundos después de mostrar el resultado y termine automáticamente.

# Ej2: Factorial (I)

- Modificaciones:
  1. Imprime el valor del factorial en todas las iteraciones.
  2. ¿En qué orden se imprime? ¿Cómo conseguir que sea el inverso?
  3. Prueba otras funciones internas:
    - .stopMAS
    - .wait(<milisegundos>)
    - .random(X)
  4. Modifica el ejemplo para que se pause 3 segundos después de mostrar el resultado y termine automáticamente.

# Ej 3. Paro y Arranque

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



# Ejemplo: Robot limpiador

Robot que trata de limpiar basura

Movimientos en una sola dimensión

La basura hay que cogerla y llevarla a la papelera

Las creencias iniciales podrían ser:

adyacente( a, b).

adyacente( b, c).

adyacente( c, d).

localizado( robot, a).

localizado(papel\_usado, b).

localizado(papelera, d).

# Ejemplo: Robot limpiador

**Objetivo inicial:** !localizado(robot, d).

**Planes:**

```
+localizado(robot, X) : localizado(papel_usado, X)
    <- .print("papel cogido en ", X);
    -localizado(papel_usado, X); // simula el coger papel
    +llevando(papel_usado). //simula el llevar papel

+localizado(robot, X) : localizado(papelera, X) & llevando(papel_usado)
    <- -llevando(papel_usado);
    .print("todo el papel tirado a la papelera en ", X). //simula tirar a papelera

+!localizado(robot, X) : localizado(robot, X) <- .print("Ha llegado a su destino").

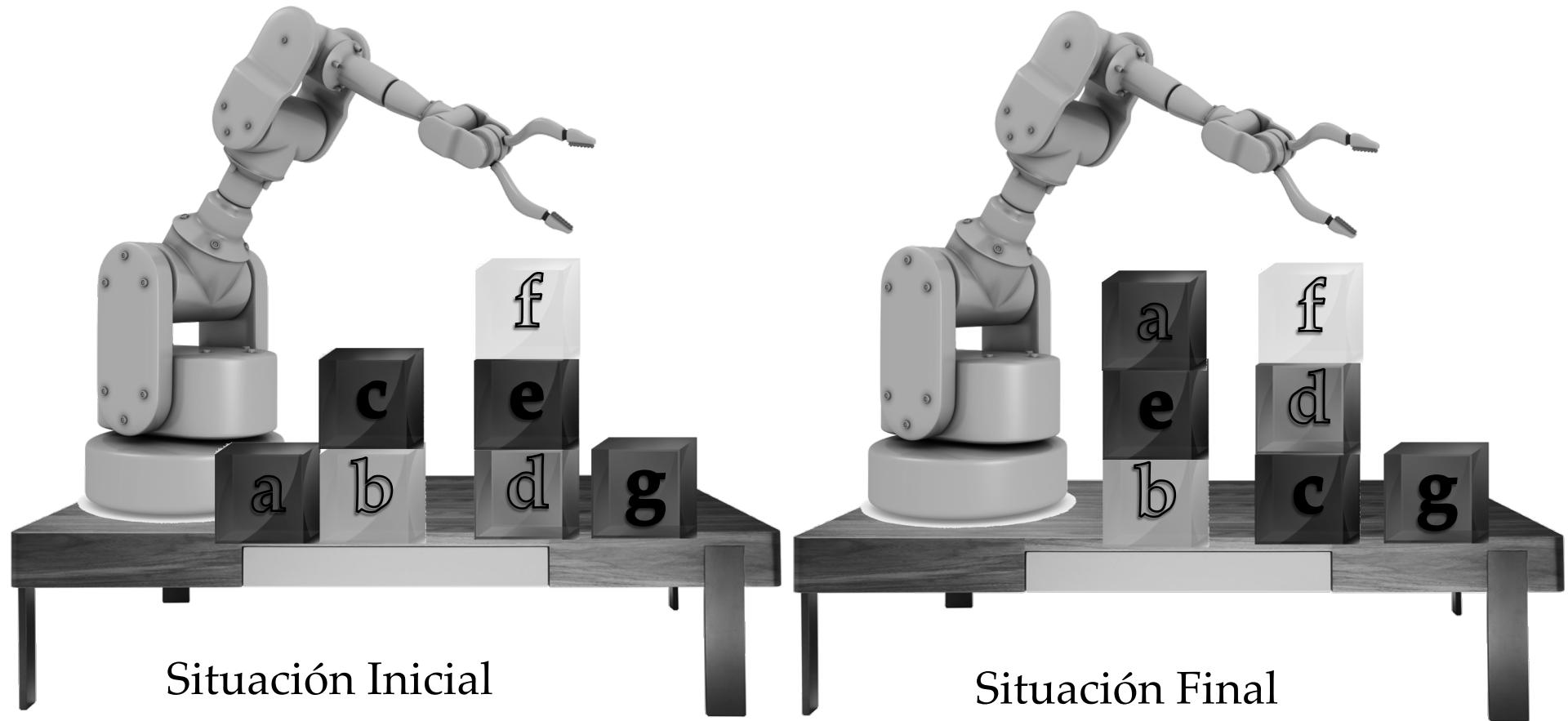
+!localizado(robot, X) : localizado(robot, Y) & (not (X=Y)) & adyacente(Y,Z)
    <- .print("mover de ", Y, " a ", Z);
    -localizado(robot, Y);
    +localizado(robot, Z); // simula el mover de Y a Z
    !localizado(robot, X).
```

# Alguna anotación sobre listas en Jason

list → "[" [ term ( ", " term ) \* [ " | " ( list | <VAR> ) ] ] "] "

- The word term is used to refer to a constant, a variable or a structure.
- A structure is used to represent complex data, for example
  - staff("Jomi Fred Hübner", 145236, lecturer, married, wife(ilze),  
kids([morgana,thales]), salary(133987.56))
- list, is very useful in logic programming. There are special operations for lists; in particular ' | ' can be used to separate the first item in a list from the list of all remaining items in it. So, if we match (more precisely, unify) a list such as [1,2,3] with list [H | T], H (the head) is instantiated with 1 and T (the tail) with the list [2,3].

# Mundo de bloques



# Ej 4. Mundo de Bloques (I)

// Un agente que soluciona el problema del mundo de bloques

/\* Creencias iniciales y reglas \*/

clear(table).

clear(X) :- not(on(\_X)). // la creencia clear(X) es cierta cuando no tiene nada encima

tower([X]) :- on(X,table). // la torre X es cierta cuando X está sobre la mesa

// X puede ser un bloque o varios

tower([X,Y|T]) :- on(X,Y) & tower([Y|T]).

// la torre va creciendo si un bloque X se pone sobre el bloque Y

/\* Objetivos iniciales \*/

// Marca el estado final a alcanzar

!state([[a,e,b],[f,d,c],[g]]).

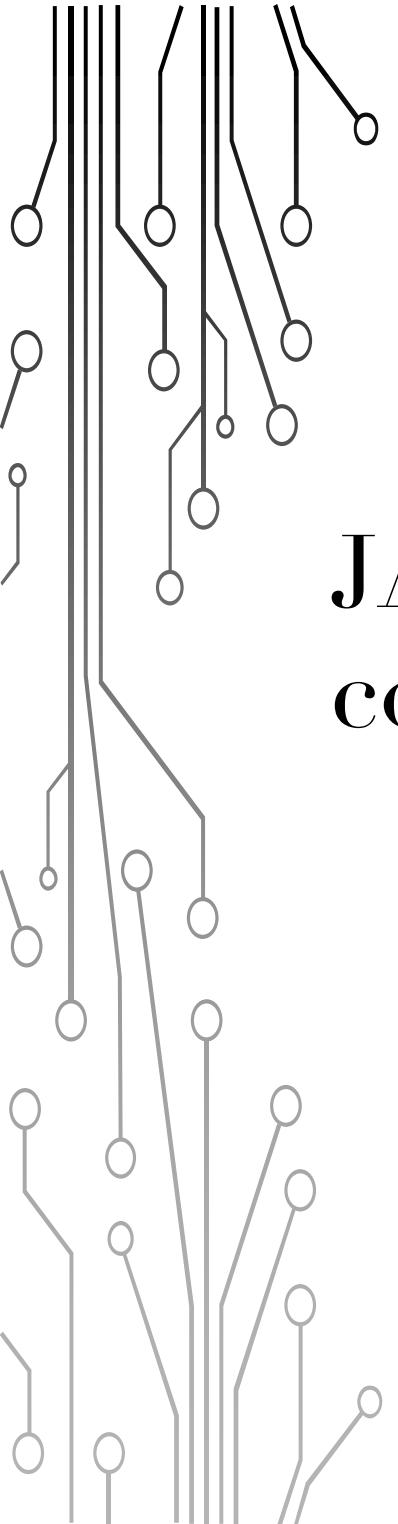
# Ej 4. Mundo de Bloques (II)

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

# Ej 4. Mundo de Bloques (III)

```
// Planes para alcanzar la creencia de que un bloque X está encima de otro Y
+!on(X,Y) : on(X,Y). // ya conseguido
+!on(X,Y) <- !clear(X); !clear(Y); move(X,Y). // consigue que estén libres y mueve X sobre Y

// Planes para alcanzar la creencia de que un bloque X está libre
+!clear(X) : clear(X). // ya conseguido
// Quita el bloque que está encima cuando se necesita dejar libre el que está debajo X
+!clear(X)
: tower([H | T]) & .member(X,T)
<- move(H,table);
!clear(X). // vuelve a lanzar el objetivo hasta que se consiga
```

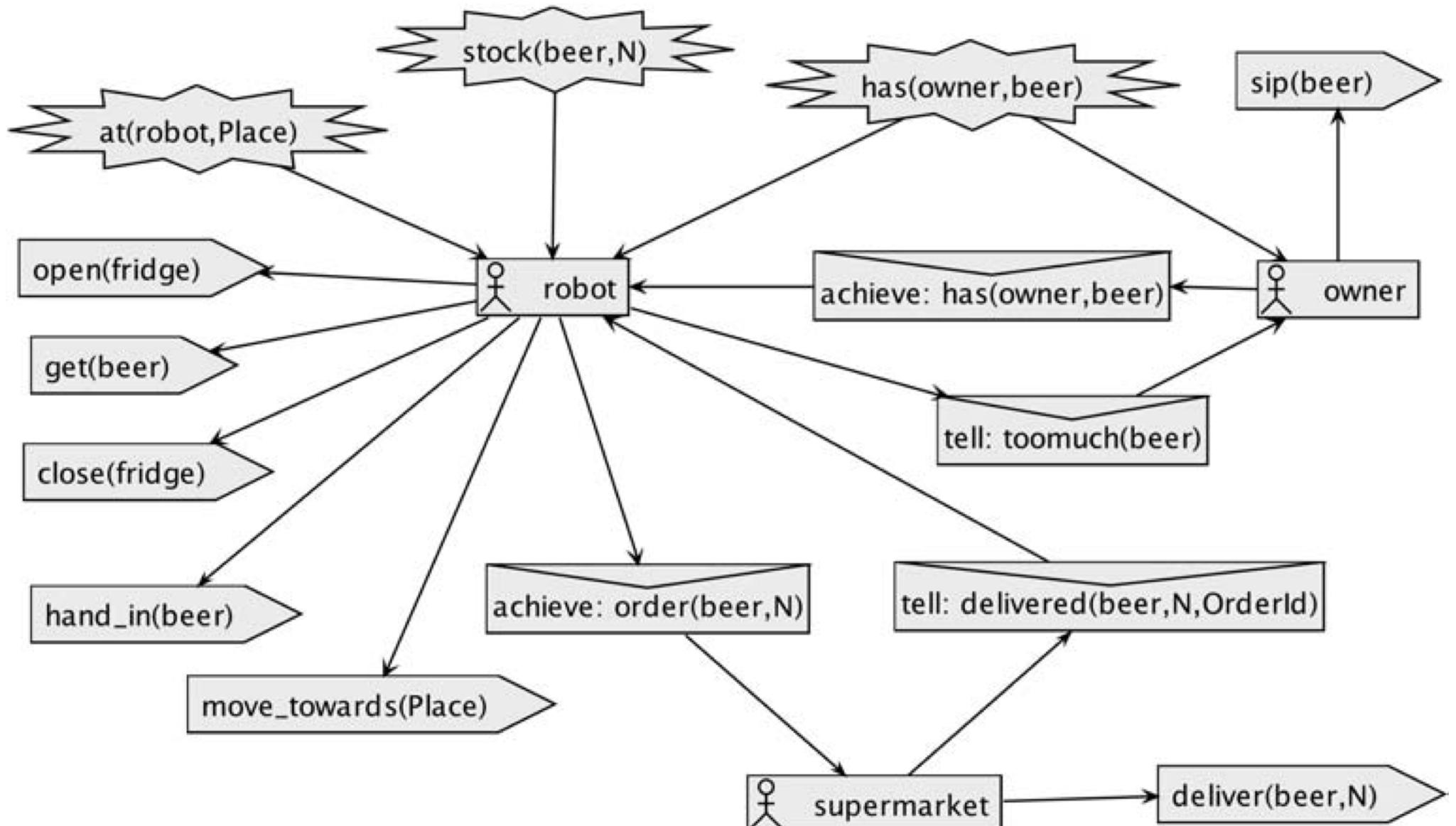


# JASON: Ejemplo de programa completo

# Un programa completo

A domestic robot has the goal of serving beer to its owner. Its mission is quite simple, it just receives some beer requests from the owner, goes to the fridge, takes out a bottle of beer, and brings it back to the owner. However, the robot should also be concerned with the beer stock (and eventually order more beer using the supermarket's home delivery service) and some rules hard-wired into the robot by the Department of Health (in this example this rule defines the limit of daily beer consumption).

# Escenario



# Percepciones

`at(robot,Place)`: to simplify the example, only two places are perceived, `fridge` (when the robot is in front of the fridge) and `owner` (when the robot is next to the owner). Thus, depending on its location in the house, the robot will perceive either `at(robot,fridge)` or `at(robot,owner)`, or of course no at percept at all (in case it is in neither of those places);

# Percepciones

stock(beer,N): when the fridge is open, the robot will perceive how many beers are stored in the fridge (the quantity is represented by the variable N);

has(owner,beer): is perceived by the robot and the owner when the owner has a (non-empty) bottle of beer.

# Agente Propietario

```
!get(beer). // initial goal
/* Plans */
@g
+!get(beer) : true
      <- .send(robot,      achieve,
has(owner,beer)).@h1
+has(owner,beer) : true
      <- !drink(beer).@h2
-has(owner,beer) : true
      <- !get(beer).
```

```
// while I have beer, sip
@d1
+!drink(beer) : has(owner,beer)
      <- sip(beer);
      !drink(beer).
@d2
+!drink(beer) : not has(owner,beer)
      <- true.

+msg(M)[source(Ag)] : true
      <- .print("Message from ",Ag,":",M);
      -msg(M).
```

# Agente Supermercado

```
last_order_id(1). // initial belief
```

```
// plan to achieve the goal "order" from agent Ag
```

```
+!order(Product,Qtd)[source(Ag)] : true
    <- ?last_order_id(N);
        OrderId = N + 1;
        -+last_order_id(OrderId);
        deliver(Product,Qtd);
        .send(Ag, tell, delivered(Product,Qtd,OrderId)).
```

# Agente Robot

```
/* Initial beliefs */  
// initially, I believe that there are some beers in the fridge available(beer,fridge).  
// my owner should not consume more than 10 beers a day :-)  
limit(beer,10).  
  
/* Rules */  
too_much(B) :-  
    .date(YY,MM,DD) &  
    .count(consumed(YY,MM,DD,_,_B),QtdB) &  
    limit(B,Limit) &  
    QtdB > Limit.
```

# Agente Robot (II)

```
/* Plans */  
@h1  
+!has(owner,beer)  
    : available(beer,fridge) & not too_much(beer)  
    <- !at(robot,fridge);  
        open(fridge);  
        get(beer);  
        close(fridge);  
        !at(robot,owner);  
        hand_in(beer);  
        // remember that another beer will be consumed  
        .date(YY,MM,DD); .time(HH,NN,SS);  
        +consumed(YY,MM,DD,HH,NN,SS,beer).
```

# Agente Robot (III)

@h2

```
+!has(owner,beer)
    : not available(beer,fridge)
    <- .send(supermarket, achieve, order(beer,5));
        !at(robot,fridge). // go to fridge and wait there.
```

@h3

```
+!has(owner,beer)
    : too_much(beer) & limit(beer,L)
    <- .concat("The Department of Health does not allow me ",
              "to give you more than ", L,
              " beers a day! I am very sorry about that!",M);
        .send(owner,tell,msg(M)).
```

# Agente Robot (IV)

@m1

```
+!at(robot,P) : at(robot,P) <- true.
```

@m2

```
+!at(robot,P) : not at(robot,P)
    <- move_towards(P);
        !at(robot,P).
```

# Agente Robot (V)

// when the supermarket finishes the order, try the 'has' goal again

@a1

```
+delivered(beer,Qtd,OrderId)[source(supermarket)] : true
    <- +available(beer,fridge);
        !has(owner,beer).
```

# Agente Robot (VI)

// when the fridge is opened, the beer stock is perceived and thus the available belief is updated

@a2

+stock(beer,0)

: available(beer,fridge)  
<- -available(beer,fridge).

@a3

+stock(beer,N)

: N > 0 & not available(beer,fridge)  
<- +available(beer,fridge).

<http://jason.sourceforge.net/jBook/jBook/Home.html>

[http://jason.sourceforge.net/mini-tutorial/getting-started/#\\_exercise](http://jason.sourceforge.net/mini-tutorial/getting-started/#_exercise)