Lección 6: Planeación de Acciones

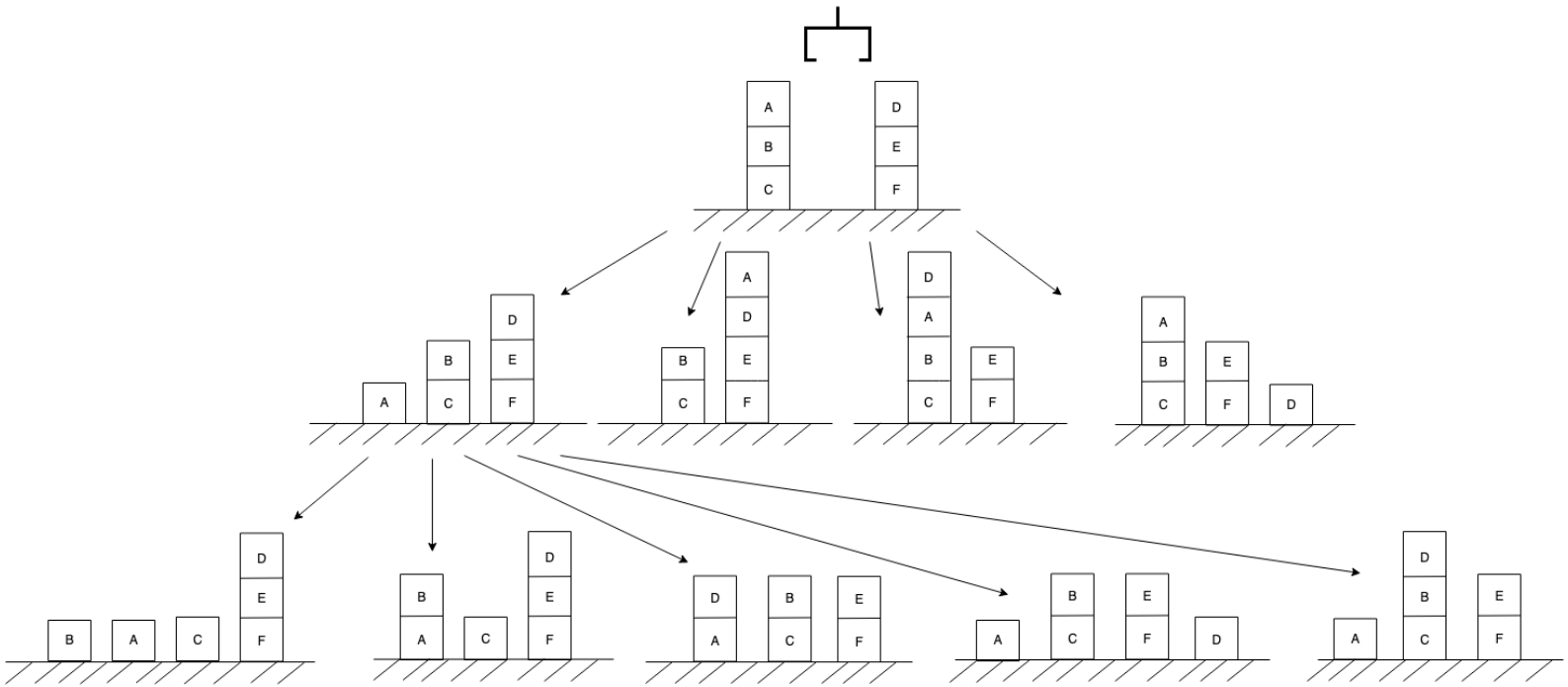
Jesús Savage
Facultad de Ingeniería, UNAM
Trabajo realizado con el apoyo del Programa
UNAM-DGAPA-PAPIME PE100821

19 de junio de 2023

Índice

Introducción

En esta lección se encontrarán cuales son las acciones necesarias que tiene que realizar un robot para lograr un objetivo específico.
Por ejemplo en la siguiente figura se desea colocar el bloque C encima del bloque F.
¿Cual es el conjunto de acciones óptimo que tiene que hacer el manipulador para lograr ésto?



Introducción

La tarea de un planeador es encontrar una secuencia de acciones que permita a un solucionador de problemas, frecuentemente un robot, resolver un problema específico.
Por ejemplo, una secuencia de acciones para: "Ve por el bloque A en el cuarto X", podría ser lo siguiente:

1. Deja lo que tengas en tu manipulador
2. Ve al cuarto X
3. Acercate al bloque A
4. Toma el bloque A
5. Sal del cuarto X
6. Regresa a la posición original

Planeación Espacio/Estado

1. La planeación puede ser vista como una búsqueda de espacio de estado.
2. En cada estado se cuenta con una representación de las condiciones actuales del mundo.
3. Nuevos estados son producidos por reglas generales.
4. Una regla está determinada por precondiciones, las cuales activan ésta, una Lista Aditiva la cual indica los hechos o condiciones nuevas del sistema y una Lista de Borrado que elimina hechos o condiciones viejas.
5. Dentro de las reglas generadas se pueden utilizar OPERADORES que efectúan operaciones físicas en el sistema.
6. Las técnicas de búsquedas de grafos pueden ser utilizadas para encontrar un camino del estado inicial al estado objetivo.

Reglas Generales de Producción

Regla Nombre {

Precondiciones

$\Rightarrow$

Operadores

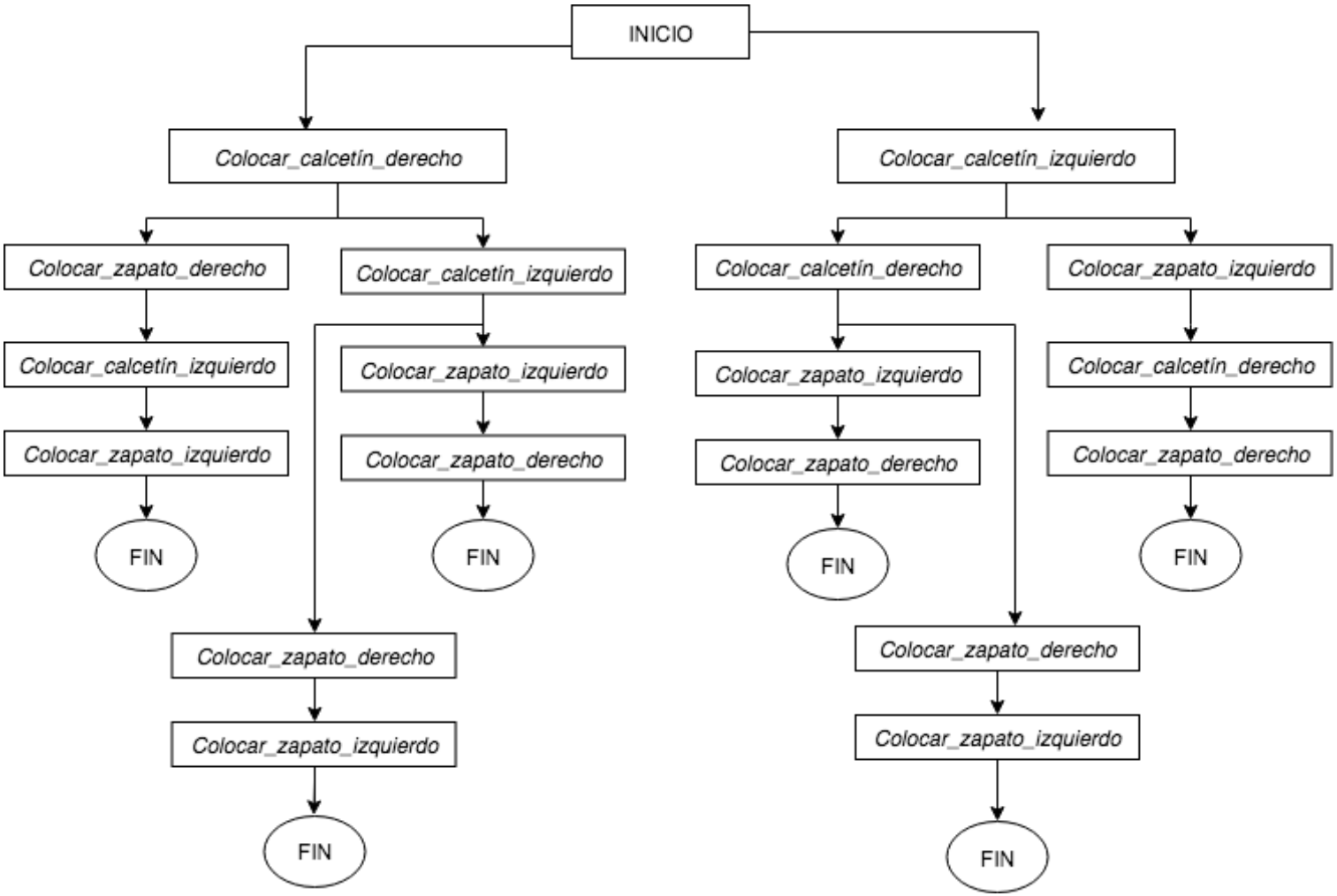
Lista de Borrado

Lista Aditiva

}

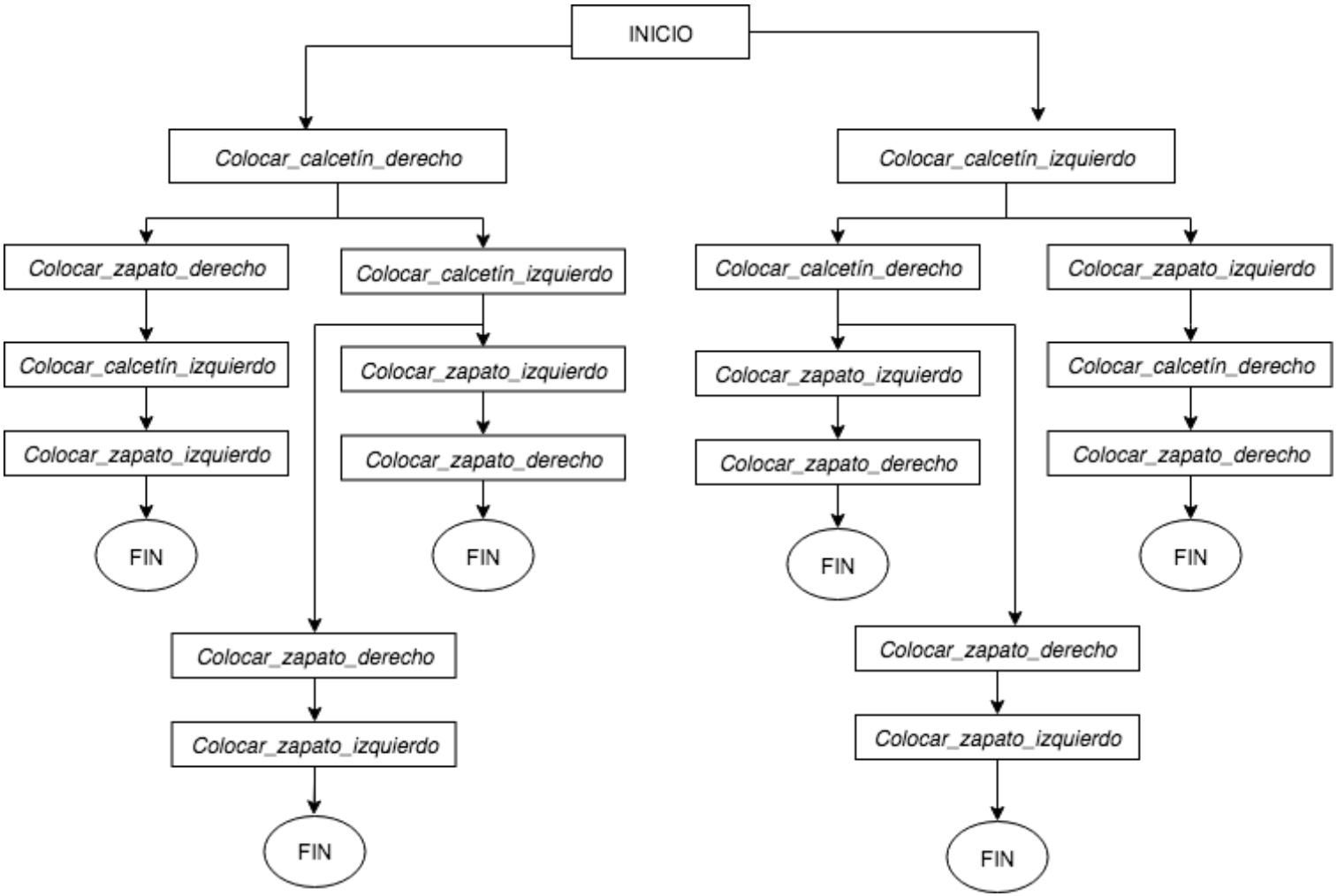
Planeación Espacio Estados

Objetivo: ¿cuales son los pasos para que una persona se ponga un par de calcetines y zapatos?



Planeación Espacio Estados

Para el ejemplo anterior hay seis posibles caminos validos para lograr el objetivo, ¿de éstos cual se consideraría el mejor?



Reglas de los Zapatos

Condiciones iniciales:

- Pie derecho desnudo
- Pie izquierdo desnudo
- Calcetín disponible
- Calcetín disponible
- Zapato derecho disponible
- Zapato izquierdo disponible
- Colocar zapato derecho
- Colocar zapato izquierdo

Reglas de los Zapatos

Regla Colocar_calcetín_derecho {

Precondiciones

Colocar zapato derecho

Pie derecho desnudo

Calcetín disponible

$\Rightarrow$

Operador

PONER calcetín derecho

Lista de Borrado

Pie derecho desnudo

Calcetín disponible

}

Reglas de los Zapatos

Regla Colocar_calcetín_izquierdo {

Precondiciones

Colocar zapato izquierdo

Pie izquierdo desnudo

Calcetín disponible

$\Rightarrow$

Operador

PONER calcetín izquierdo

Lista de Borrado

Pie izquierdo desnudo

Calcetín disponible

}

Reglas de los Zapatos

Regla Colocar_zapato_derecho {

Precondiciones

No pie derecho desnudo

Colocar zapato derecho

Zapato derecho disponible

$\Rightarrow$

Operador

PONER zapato derecho

Lista de Borrado

Zapato derecho disponible

Colocar zapato derecho

}

Reglas de los Zapatos

Regla Colocar_zapato_izquierdo {

Precondiciones

No Pie Izquierdo Desnudo

Colocar Zapato Izquierdo

Zapato Izquierdo Disponible

$\Rightarrow$

Operador

PONER Zapato Izquierdo

Lista de Borrado

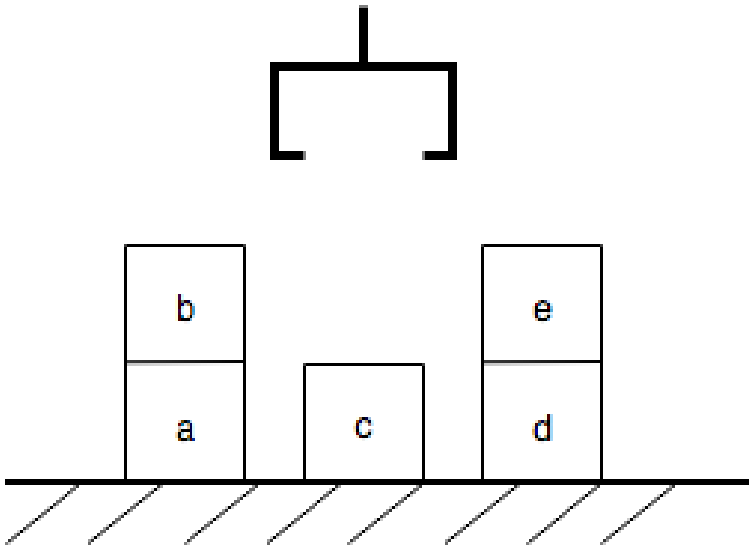
Zapato Izquierdo Disponible

Colocar Zapato Izquierdo

}

Mundo de los Bloques

Para el mundo de los bloques mostrado en la siguiente figura se cuenta con los siguientes operadores:



1. goto(X,Y,Z): El manipulador se coloca en las coordenadas X, Y y Z.
2. pickup(W): Toma el bloque W desde la posición en donde se encuentra y lo sostiene. Se asume que no hay nada encima del bloque W y que el manipulador no esta cargando nada, además que se conoce la posición del bloque W.

Mundo de los Bloques

3. putdown(W): Suelta el manipulador el bloque W en alguna posición en la mesa y se registra ésta como una nueva posición del bloque W. El manipulador debería tener el bloque W en este tiempo.
4. stack(U,V): Coloca el bloque U encima del bloque V. El manipulador debera estar cargando el bloque U y encima de V no debe de haber otros bloques.
5. unstack(U,V): Remueve el bloque U de encima del bloque V.

Cálculo de Predicados y Planeación

Representación del estado del mundo usando predicados (hechos) y sus relaciones entre ellos:

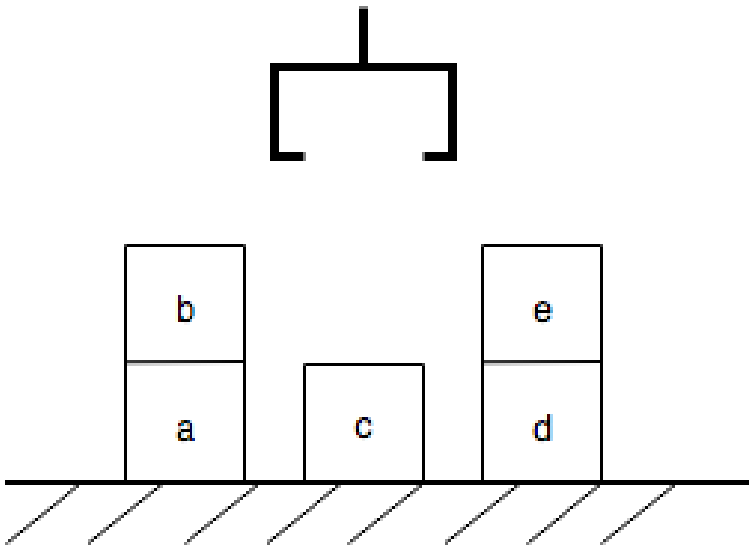
1. location(W,X,Y,Z): El bloque W esta en las coordenadas X, Y, Z.
2. on(X,Y): El bloque X esta encima del bloque Y.
3. clear(X): Encima del bloque X no hay ningún bloque.
4. gripping(X): El manipulador esta cargando al bloque X.
5. gripping( ): El manipulador esta vacio.
6. ontable(W): El bloque W esta sobre la mesa.

Cálculo de Predicados y Planeación

La configuración de los bloques de la siguiente figura puede ser representada por los siguientes predicados o hechos. Esta colección de hechos representan el ESTADO 1 del sistema:

STATE 1

ontable(a)	on(b,a)	clear(b)
ontable(c)	on(e,d)	clear(c)
ontable(d)	gripping( )	clear(e)



Cálculo de Predicados y Planeación

Se cuenta con relaciones de verdad o reglas:

1. $(\forall X)\ (pickup(X) \rightarrow (gripping(X)) \leftarrow (gripping(\ ) \wedge (clear(X)\ )\ ).$
2. $(\forall X)\ (putdown(X) \rightarrow ((gripping() \wedge ontable(X) \wedge clear(X)) \leftarrow gripping(X)\ ).$
3. $(\forall X)\ (\forall Y)\ (unstack(X, Y) \rightarrow ((clear(Y) \wedge gripping(X)) \leftarrow (on(X, Y) \wedge clear(X) \wedge gripping(\ )\ ).$
4. $(\forall X)(\forall Y)(\forall Z)\ (unstack(Y, Z) \rightarrow (ontable(X)) \leftarrow ontable(X)\ )$
5. $(\forall X)(\forall Y)(\forall Z)\ (stack(Y, Z) \rightarrow (ontable(X)) \leftarrow ontable(X)\ )$

Cálculo de Predicados y Planeación

Los operadores, las reglas y los predicados (hechos) definen un espacio de estados
Resumiendo:

1. La planeación puedes ser vista como una busqueda de espacio/estado.
2. Nuevos estados son producidos por operadores generales invocados por las reglas.
3. Técnicas de busqueda pueden ser utilizadas para encontrar un camino de un estado incial a un estado objetivo. Las operaciones que se hacen en este camino se consideran como el plan.

Sistema Basado en Reglas CLIPS

CLIPS es una máquina de inferencias desarrollado por la NASA, el cual permite programar sistemas basado en reglas. Se presenta a continuación la forma como se resuelve el problema de los zapatos.

```
;*****************************************
; Hechos iniciales zapatos
;*****************************************
(deffacts estado-inicial
            (estado levantandose)
            (pantalon puesto)
            (pie derecho desnudo)
            (pie izquierdo desnudo)
            (calcetin derecho)
            (calcetin izquierdo)
            (zapato derecho)
            (zapato izquierdo)
)
```

Sistema Basado en Reglas CLIPS

```
;*****************************************
; REGLAS Zapatos
;*****************************************
; Regla inicial
(defrule inicio
        (estado levantandose)
        (pantalon puesto)
        (pie ?pie desnudo)
        =>
        (assert (colocar zapato ?pie))
)
```

Sistema Basado en Reglas CLIPS

```
; Coloca el calcetin
(defrule colocar-calcetin
            (colocar zapato ?pie )
            ?f1 <- (pie ?pie desnudo)
            ?f2 <- (calcetin ?pie)
            =>
            (retract ?f1 ?f2)
            ; Directiva PONER
            (assert (PONER calcetin ?pie))
)
```

Sistema Basado en Reglas CLIPS

```
; Coloca el zapato
(defrule colocar-zapato
            ?f1 <- (colocar zapato ?pie )
            (not (pie ?pie desnudo))
            ?f2 <- (zapato ?pie)
            =>
            (retract ?f1 ?f2)
            ; Directiva PONER
            (assert (PONER zapato ?pie))
)
```

Uso de la directiva salience en CLIPS

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load zapatos.clp)
Defining deffacts: estado-inicial
Defining defrule: inicio +j+j+j
Defining defrule: colocar-calcetin +j+j+j
Defining defrule: colocar-zapato =j+j+j
TRUE
CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (estado levantandose)
==> f-2     (pantalon puesto)
==> f-3     (pie derecho desnudo)
==> f-4     (pie izquierdo desnudo)
==> f-5     (calcetin derecho)
==> f-6     (calcetin izquierdo)
==> f-7     (zapato derecho)
==> f-8     (zapato izquierdo)
CLIPS> (run)
FIRE    1 inicio: f-1,f-2,f-4
==> f-9     (colocar zapato izquierdo)
FIRE    2 inicio: f-1,f-2,f-3
==> f-10    (colocar zapato derecho)
FIRE    3 colocar-calcetin: f-10,f-3,f-5
<== f-3     (pie derecho desnudo)
<== f-5     (calcetin derecho)
==> f-11    (PONER calcetin derecho)
FIRE    4 colocar-zapato: f-10,,f-7
<== f-10    (colocar zapato derecho)
<== f-7     (zapato derecho)
==> f-12    (PONER zapato derecho)
FIRE    5 colocar-calcetin: f-9,f-4,f-6
<== f-4     (pie izquierdo desnudo)
<== f-6     (calcetin izquierdo)
==> f-13    (PONER calcetin izquierdo)
FIRE    6 colocar-zapato: f-9,,f-8
<== f-9     (colocar zapato izquierdo)
<== f-8     (zapato izquierdo)
==> f-14    (PONER zapato izquierdo)
CLIPS>
```

Uso de la directiva salience en CLIPS

```
;*****************************************
; REGLAS Zapatos
;*****************************************
; Regla inicial
(defrule inicio
            (declare (salience 200))
            (estado levantandose)
            (pantalon puesto)
            (pie ?pie desnudo)
            =>
            (assert (colocar zapato ?pie))
)
```

Uso de la directiva salience en CLIPS

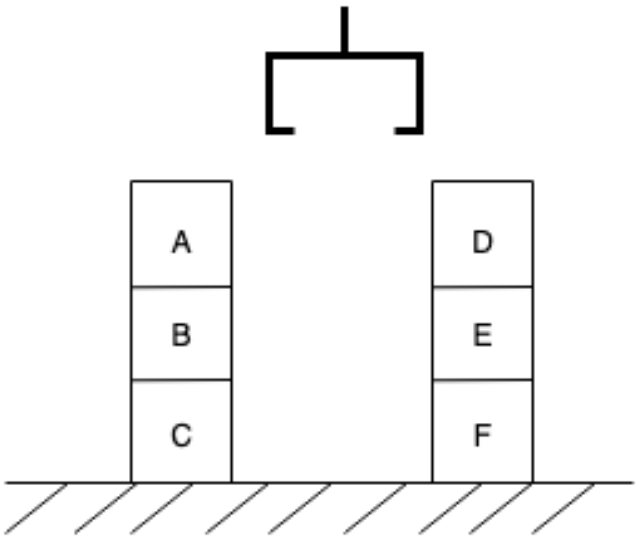
```
; Coloca el calcetin
(defrule colocar-calcetin
            (declare (salience 100))
            (colocar zapato ?pie )
            ?f1 <- (pie ?pie desnudo)
            ?f2 <- (calcetin ?pie)
            =>
            (retract ?f1 ?f2)
            ; Directiva PONER
            (assert (PONER calcetin ?pie))
)
```

Uso de la directiva salience en CLIPS

```
         CLIPS (V6.24 06/15/06)
CLIPS> (load zapatos.clp)
Defining deffacts: estado-inicial
Defining defrule: inicio +j+j+j
Defining defrule: colocar-calcetin +j+j+j
Defining defrule: colocar-zapato =j+j+j
TRUE
CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (estado levantandose)
==> f-2     (pantalon puesto)
==> f-3     (pie derecho desnudo)
==> f-4     (pie izquierdo desnudo)
==> f-5     (calcetin derecho)
==> f-6     (calcetin izquierdo)
==> f-7     (zapato derecho)
==> f-8     (zapato izquierdo)
CLIPS> (run)
FIRE    1 inicio: f-1,f-2,f-4
==> f-9     (colocar zapato izquierdo)
FIRE    2 inicio: f-1,f-2,f-3
==> f-10    (colocar zapato derecho)
FIRE    3 colocar-calcetin: f-10,f-3,f-5
<== f-3     (pie derecho desnudo)
<== f-5     (calcetin derecho)
==> f-11    (PONER calcetin derecho)
FIRE    4 colocar-calcetin: f-9,f-4,f-6
<== f-4     (pie izquierdo desnudo)
<== f-6     (calcetin izquierdo)
==> f-12    (PONER calcetin izquierdo)
FIRE    5 colocar-zapato: f-9,,f-8
<== f-9     (colocar zapato izquierdo)
<== f-8     (zapato izquierdo)
==> f-13    (PONER zapato izquierdo)
FIRE    6 colocar-zapato: f-10,,f-7
<== f-10    (colocar zapato derecho)
<== f-7     (zapato derecho)
==> f-14    (PONER zapato derecho)
CLIPS>
```

Sistema Basado en Reglas CLIPS

Se presenta a continuación la forma como se resuelve el problema de los cubos como el presentado en la siguiente figura, con el objetivo de mover al cubo F encima del cubo C:



Sistema Basado en Reglas CLIPS

```
;*****************************************
; Plantillas cubos
;*****************************************
(deftemplate on-top-of
             (slot upper)
             (slot lower)
)
(deftemplate goal (slot move)(slot on-top-of))
```

Sistema Basado en Reglas CLIPS

```
(deffacts initial-state
        (block A)
        (block B)
        (block C)
        (block D)
        (block E)
        (block F)
        (on-top-of (upper nothing)(lower A))
        (on-top-of (upper A)(lower B))
        (on-top-of (upper B)(lower C))
        (on-top-of (upper C)(lower floor))
        (on-top-of (upper nothing)(lower D))
        (on-top-of (upper D)(lower E))
        (on-top-of (upper E)(lower F))
        (on-top-of (upper F)(lower floor))
        (goal (move F)(on-top-of C)) )
```

Sistema Basado en Reglas CLIPS

```
;*****************************************
; REGLAS Cubos
;*****************************************
(defrule move-directly
        ?goal <- (goal (move ?block1) (on-top-of ?block2))
        (block ?block1)
        (block ?block2)
        (on-top-of (upper nothing) (lower ?block1))
        ?stack-1 <- (on-top-of (upper ?block1)(lower ?block3))
        ?stack-2 <- (on-top-of (upper nothing)(lower ?block2))
        =>
        (retract ?goal ?stack-1 ?stack-2)
        (assert (on-top-of (upper ?block1)(lower ?block2))
        (on-top-of (upper nothing)(lower ?block3)))
        (printout t ?block1 " moved on top of " ?block2 "." crlf)
)
```

Sistema Basado en Reglas CLIPS

```
(defrule move-to-floor
        ?goal <- (goal (move ?block1) (on-top-of floor))
        (block ?block1)
        (on-top-of (upper nothing) (lower ?block1))
        ?stack <- (on-top-of (upper ?block1) (lower ?block2))
        =>
        (retract ?goal ?stack)
        (assert (on-top-of (upper ?block1)(lower floor))
        (on-top-of (upper nothing)(lower ?block2)))
        (printout t ?block1 " moved on top of floor. " crlf)
)
```

Sistema Basado en Reglas CLIPS

```
(defrule clear-upper-block
        (goal (move ?block1))
        (block ?block1)
        (on-top-of (upper ?block2) (lower ?block1))
        (block ?block2)
        =>
        (assert (goal (move ?block2)(on-top-of floor)))
)
```

Sistema Basado en Reglas CLIPS

```
(defrule clear-lower-block
            (goal (on-top-of ?block1))
            (block ?block1)
            (on-top-of (upper ?block2) (lower ?block1))
            (block ?block2)
            =>
            (assert (goal (move ?block2)(on-top-of floor)))
)
```

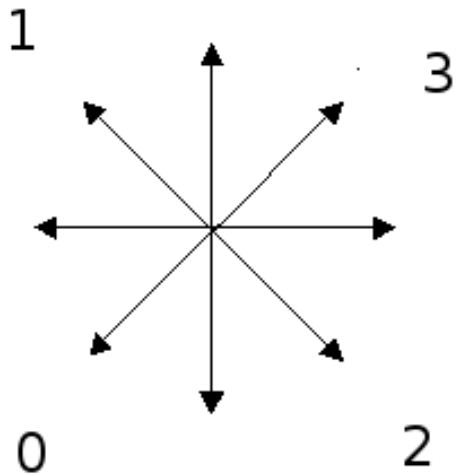
Solución en CLIPS del problema de los cubos

```
CLIPS> (load cubes.clp)
Defining deftemplate: on-top-of
Defining deftemplate: goal
Defining deffacts: initial-state
Defining defrule: move-directly +j+j+j+j+j+j
Defining defrule: move-to-floor +j+j+j+j
Defining defrule: clear-upper-block =j=j+j+j
Defining defrule: clear-lower-block =j+j+j+j
TRUE
CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0     (initial-fact)
==> f-1     (block A)
==> f-2     (block B)
==> f-3     (block C)
==> f-4     (block D)
==> f-5     (block E)
==> f-6     (block F)
==> f-7     (on-top-of (upper nothing) (lower A))
==> f-8     (on-top-of (upper A) (lower B))
==> f-9     (on-top-of (upper B) (lower C))
==> f-10    (on-top-of (upper C) (lower floor))
==> f-11    (on-top-of (upper nothing) (lower D))
==> f-12    (on-top-of (upper D) (lower E))
==> f-13    (on-top-of (upper E) (lower F))
==> f-14    (on-top-of (upper F) (lower floor))
==> f-15    (goal (move F) (on-top-of C))
CLIPS> (run)
FIRE    1 clear-upper-block: f-15,f-6,f-13,f-5
==> f-16    (goal (move E) (on-top-of floor))
FIRE    2 clear-upper-block: f-16,f-5,f-12,f-4
==> f-17    (goal (move D) (on-top-of floor))
FIRE    3 move-to-floor: f-17,f-4,f-11,f-12
<== f-17    (goal (move D) (on-top-of floor))
<== f-12    (on-top-of (upper D) (lower E))
==> f-18    (on-top-of (upper D) (lower floor))
==> f-19    (on-top-of (upper nothing) (lower E))
D moved on top of floor.
FIRE    4 move-to-floor: f-16,f-5,f-19,f-13
<== f-16    (goal (move E) (on-top-of floor))
<== f-13    (on-top-of (upper E) (lower F))
==> f-20    (on-top-of (upper E) (lower floor))
==> f-21    (on-top-of (upper nothing) (lower F))
E moved on top of floor.
FIRE    5 clear-lower-block: f-15,f-3,f-9,f-2
==> f-22    (goal (move B) (on-top-of floor))
FIRE    6 clear-upper-block: f-22,f-2,f-8,f-1
==> f-23    (goal (move A) (on-top-of floor))
FIRE    7 move-to-floor: f-23,f-1,f-7,f-8
<== f-23    (goal (move A) (on-top-of floor))
<== f-8     (on-top-of (upper A) (lower B))
==> f-24    (on-top-of (upper A) (lower floor))
==> f-25    (on-top-of (upper nothing) (lower B))
A moved on top of floor.
FIRE    8 move-to-floor: f-22,f-2,f-25,f-9
<== f-22    (goal (move B) (on-top-of floor))
<== f-9     (on-top-of (upper B) (lower C))
==> f-26    (on-top-of (upper B) (lower floor))
==> f-27    (on-top-of (upper nothing) (lower C))
B moved on top of floor.
FIRE    9 move-directly: f-15,f-6,f-3,f-21,f-14,f-27
<== f-15    (goal (move F) (on-top-of C))
<== f-14    (on-top-of (upper F) (lower floor))
<== f-27    (on-top-of (upper nothing) (lower C))
==> f-28    (on-top-of (upper F) (lower C))
==> f-29    (on-top-of (upper nothing) (lower floor))
F moved on top of C.
CLIPS>
```

Solución en CLIPS del comportamiento reactivo

Para poder utilizar CLIPS en la instrumentación de los comportamientos reactivos se necesita convertir las lecturas de los sensores en símbolos. Para el sensor de detección de luz que se indica con 8 valores la posición de la fuente luminosa estos valores se cuantizan para formar un valor simbólico con 4 símbolos o indices:

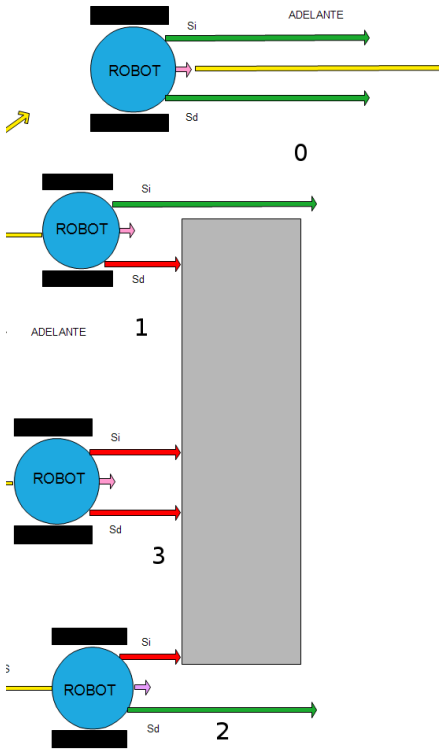
Cuadrante $= \{$ atras derecha, atras izquierda, adelante derecha, adelante izquierda$\}=\{0, 1, 2, 3\}$



Solución en CLIPS del comportamiento reactivo

Para los sensores de detección de obstáculos, utilizando los sensores de proximidad, se indica con 4 valores posición de los obstáculos con 4 símbolos o indices:

Cuadrante $=\{$ no hay obstáculo, obstáculo a la derecha,
obstáculo a la izquierda , obstáculo enfrente$\}=\{0, 1, 2, 3\}$

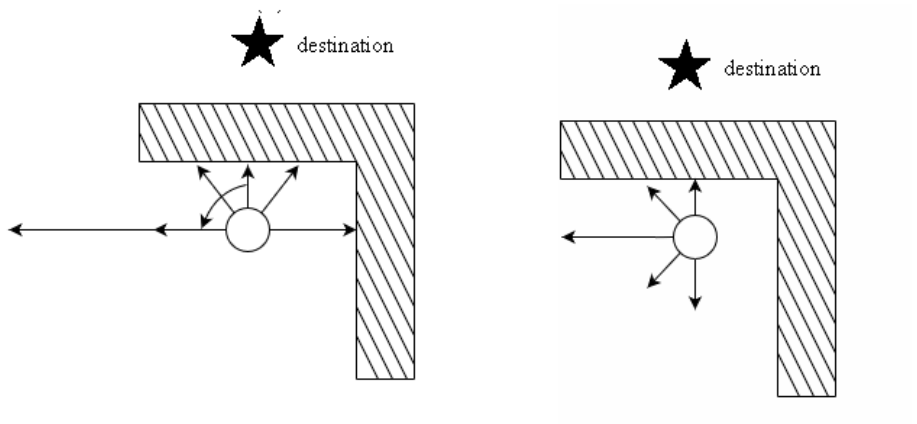


Solución en CLIPS del comportamiento reactivo

Se tienen entonces cientos de reglas del siguiente tipo:

* Si la fuente luminosa esta arriba del robot y hay un obstáculo enfrente y al lado derecho, entonces el robot gira a la izquierda.

* Si la fuente luminosa esta a la derecha del robot y hay un obstáculo a la derecha y atras de éste, entonces el robot avanzara hacia adelante.



Solución en CLIPS del comportamiento reactivo

```
;*****************************************
;*      Jesus Savage                     *
;*                                       *
;*              Bio-Robotics Laboratory  *
;*              UNAM, 2019               *
;*                                       *
;*                                       *
;*****************************************


(defrule clips-alive
       ?f <- (alive clips)
       =>
       (retract ?f)
       (printout t "ROS clips alive ROS")
)


(defrule Max-Values
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (printout t "ROS received max advance rotation ROS")
)


(defrule obs-dest
       ?f <- (step ?num intensity ?int obs ?obs dest ?dest)
       =>
       (retract ?f)
       (assert (received ?num ?obs ?dest))
       (assert (intensity ?num ?int))
       (bind ?num (* ?num 1))
       ;(printout t "Modified number " ?num)
)


(defrule no-obstacle-light-backward-right
       ?f <- (received ?num 0 0)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       ; rotate to the right 135 degrees and advance forward
       (bind ?right_135 (- 0 (* 3 ?max-rotation)))
       (bind ?forward ?max-advance)
       (assert (movement ?num ?right_135 ?forward 0.5))
)


(defrule no-obstacle-light-backward-left
       ?f <- (received ?num 0 1)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       ; rotate to the left 135 degrees and advance forward
       (bind ?left_135 (* 2 ?max-rotation))
       (bind ?forward ?max-advance)
       (assert (movement ?num ?left_135 ?forward 0.5))
)


(defrule no-obstacle-light-front-right
       ?f <- (received ?num 0 2)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       ; rotate to the right 45 degrees and advance forward
       (bind ?right_45 (- 0 ?max-rotation))
       (bind ?forward ?max-advance)
       (assert (movement ?num ?right_45 ?forward 0.5))
)
```

Solución en CLIPS del comportamiento reactivo

```
(defrule no-obstacle-light-front-left
        ?f <- (received ?num 0 3)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        ; rotate to the left 45 degrees and advance forward
        (bind ?left_45 ?max-rotation)
        (bind ?forward ?max-advance)
        (assert (movement ?num ?left_45 ?forward 0.5))
)


(defrule obstacle-right-light-backward-right
        ?f <- (received ?num 1 0)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_90 (* 2 ?max-rotation))
        (bind ?forward ?max-advance)
        ; rotate to the 90 degrees and goes forward
        (assert (movement ?num ?left_90 ?forward 0.5))
)


(defrule obstacle-right-light-back-left
        ?f <- (received ?num 1 1)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_135 (* 3 ?max-rotation))
        (bind ?forward ?max-advance)
        ; rotate to the left 135 degrees and advance forward
        (assert (movement ?num ?left_135 ?forward 0.5))
)
(defrule obstacle-right-light-front-right
        ?f <- (received ?num 1 2)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_45 (- 0 ?max-rotation))
        (bind ?backward (- 0 ?max-advance))
        ; rotate to the left 45 degrees and advance forward
        (assert (movement ?num ?left_45 ?backward 0.5))
)


(defrule obstacle-right-light-front-left
        ?f <- (received ?num 1 3)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_45 ?max-rotation)
        (bind ?forward ?max-advance)
        ; rotate to the left 45 degrees and advance forward
        (assert (movement ?num ?left_45 ?forward 0.5))
)


(defrule obstacle-left-light-backward-right
        ?f <- (received ?num 2 0)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?right_135 (- 0 (* 3 ?max-rotation)))
        (bind ?forward ?max-advance)
        ; rotate to the right 135 degrees and goes forward
        (assert (movement ?num ?right_135 ?forward 0.5))
)
```

Solución en CLIPS del comportamiento reactivo

```
(defrule obstacle-left-light-back-left
        ?f <- (received ?num 2 1)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_135 (* 3 ?max-rotation))
        (bind ?forward ?max-advance)
        ; rotate to the left 135 degrees and advance forward
        (assert (movement ?num ?left_135 ?forward 0.5))
)


(defrule obstacle-left-light-front-right
        ?f <- (received ?num 2 2)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?zero 0.0)
        (bind ?forward (- 0 ?max-advance))
        (assert (movement ?num ?zero ?forward 0.5))
)


(defrule obstacle-left-light-front-left
        ?f <- (received ?num 2 3)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?right_45 ?max-rotation)
        (bind ?forward ?max-advance)
        ; rotate to the right 45 degrees and advance forward
        (assert (movement ?num ?right_45 ?forward 0.5))
)


(defrule obstacle-front-light-backward-left
        ?f <- (received ?num 3 0)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?left_135 (* 3 ?max-rotation))
        (bind ?forward ?max-advance)
        ; rotate to the left 135 degrees and advance forward
        (assert (movement ?num ?left_135 ?forward 0.5))
)


(defrule obstacle-front-light-backward-right
        ?f <- (received ?num 3 1)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?right_135 (- 0 (* 3 ?max-rotation)))
        (bind ?forward ?max-advance)
        ; rotate to the right 135 degrees and goes forward
        (assert (movement ?num ?right_135 ?forward 0.5))
)


(defrule obstacle-front-light-front-right
        ?f <- (received ?num 3 2)
        (max-advance ?max-advance max-rotation ?max-rotation)
        =>
        (retract ?f)
        (bind ?right_90 (- 0 (* 3 ?max-rotation)))
        (bind ?forward ?max-advance)
        ; rotate to the right 90 degrees and advance forward
        (assert (movement ?num ?right_90 ?forward 0.5))
)
```

Solución en CLIPS del comportamiento reactivo

```
defrule obstacle-front-light-backward-right
       ?f <- (received ?num 3 1)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       (bind ?right_135 (- 0 (* 3 ?max-rotation)))
       (bind ?forward ?max-advance)
       ; rotate to the right 135 degrees and goes forward
       (assert (movement ?num ?right_135 ?forward 0.5))


defrule obstacle-front-light-front-right
       ?f <- (received ?num 3 2)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       (bind ?right_90 (- 0 (* 3 ?max-rotation)))
       (bind ?forward ?max-advance)
       ; rotate to the right 90 degrees and advance forward
       (assert (movement ?num ?right_90 ?forward 0.5))


defrule obstacle-front-light-front-left
       ?f <- (received ?num 3 3)
       (max-advance ?max-advance max-rotation ?max-rotation)
       =>
       (retract ?f)
       (bind ?left_90 (* 1.5 ?max-rotation))
       (bind ?forward ?max-advance)
       ; rotate to the left 90 degrees and advance forward
       (assert (movement ?num ?left_90 ?forward 0.5))


defrule arbiter
       ?f <- (movement ?num ?rotation ?advance ?status)
       ?f1 <- (intensity ?num ?intensity)
       =>
       (retract ?f ?f1)
       (if (> ?intensity 30.0) then
              (printout t "ROS movement " ?num " " ?rotation " " ?advance " " 1.0 " ROS")
       else
              (printout t "ROS movement " ?num " " ?rotation " " ?advance " " ?status " ROS")
       )


defrule delete-unused-intensities
       (declare (salience -1000))
       ?f <- (intensity ?num $?)
       =>
       (retract ?f)


defrule delete-unused-received
       (declare (salience -1000))
       ?f <- (received ?num $?)
       =>
       (assert (movement ?num 0.0 0.0 0.5))
       (retract ?f)
```