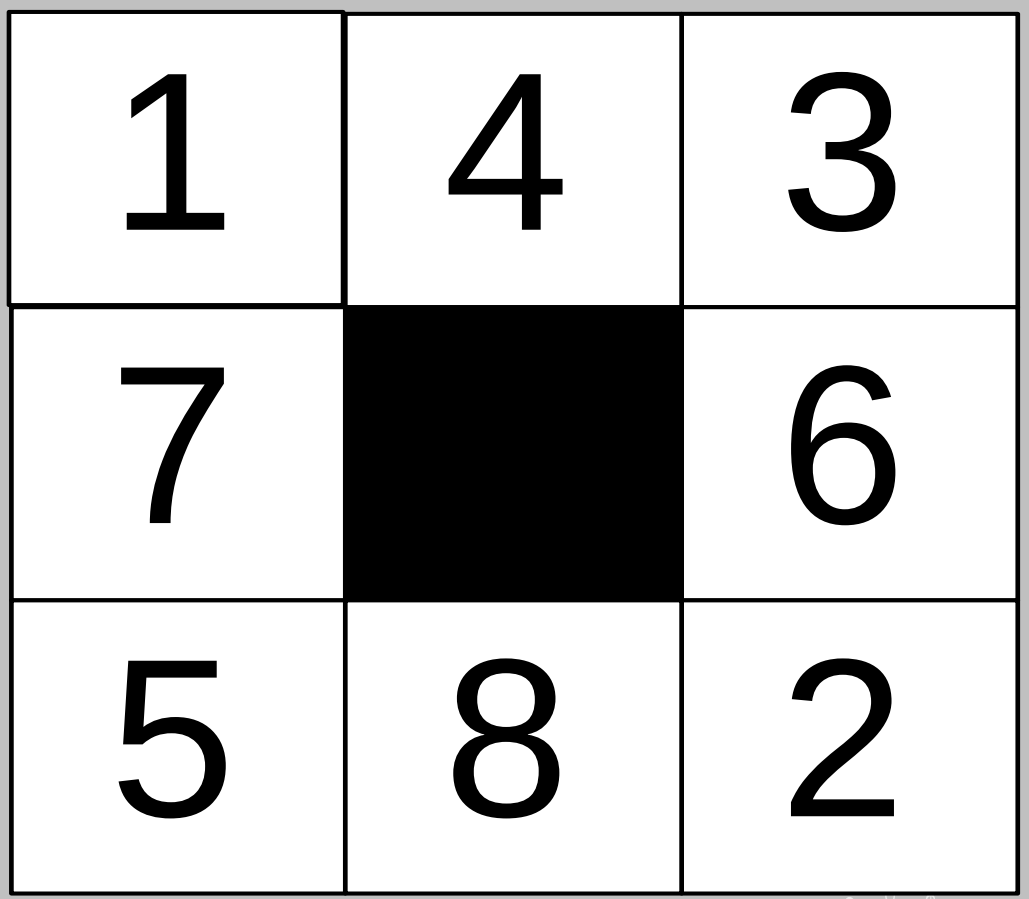
TÉCNICAS DE RESOLUCIÓN DE PROBLEMAS

INTRODUCCIÓN

BÚSQUEDA
+
REPRESENTACIÓN

Juego del 8-puzzle



La inteligencia reside en sistemas de símbolos físicos

-Newell y Simon,

ACM *Turing Award Lecture, 1976*

Una segunda ley de estructura cualitativa para la inteligencia artificial es que los sistemas de símbolos resuelven problemas generando soluciones potenciales y comprobándolas -esto es, con búsqueda.

Las soluciones se buscan usualmente creando expresiones simbólicas y modificandolas secuencialmente hasta que satisfacen las condiciones para una solu

Búsqueda

Origenes históricos de la búsqueda heurística

Newell y Simon,

ACM Turing Award Lecture, 1976

Problema

La tarea con la que se enfrenta un sistema de símbolos, cuando se le presenta un problema y un espacio de problema, es ***utilizar sus limitados recursos de procesamiento para generar posibles soluciones****, una después de otra, hasta que encuentra una que satisface el test de definición del problema.*

…

Búsqueda inteligente

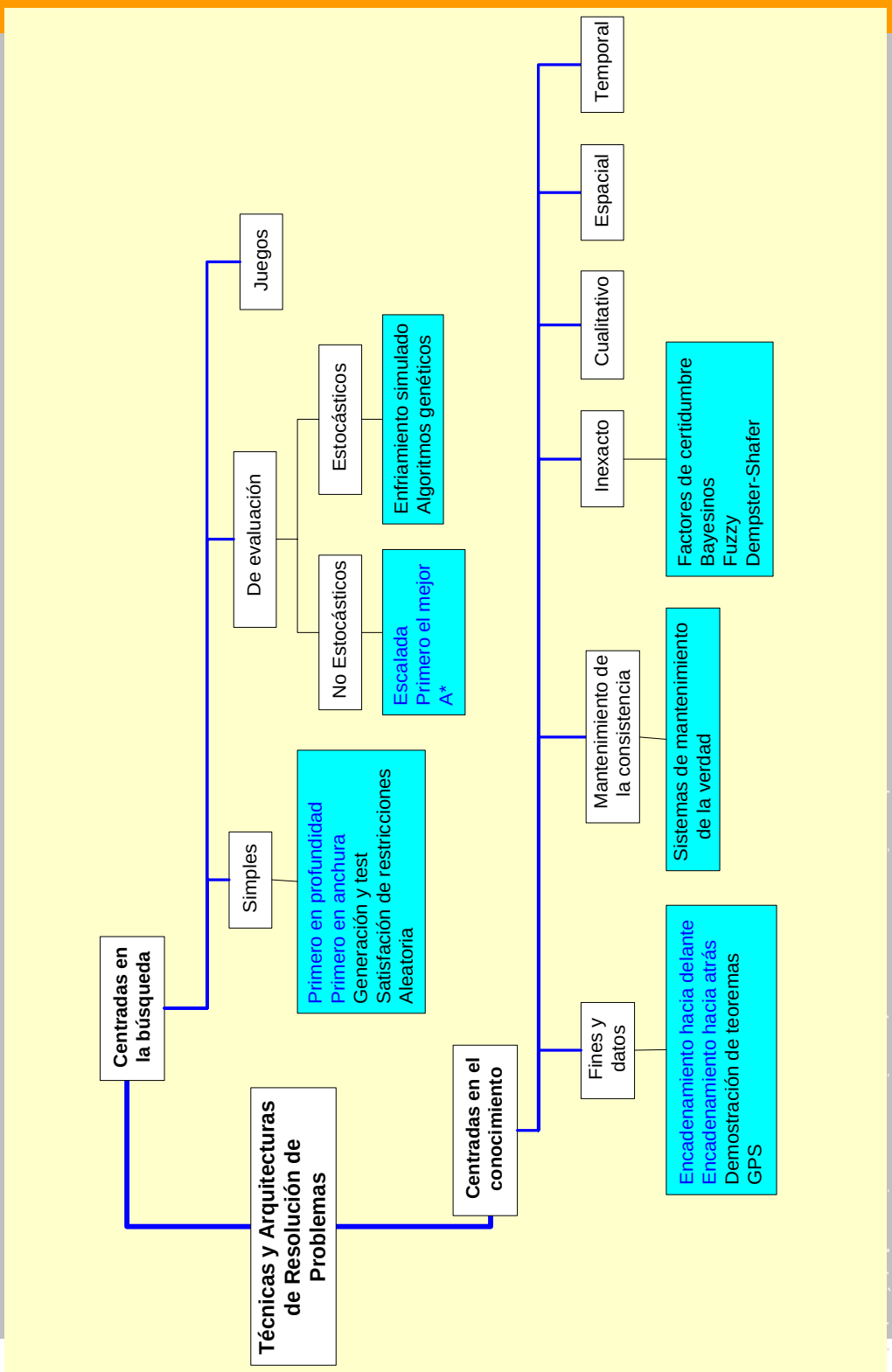
- *Si el sistema de símbolos tuviera algún control sobre el orden en que se generan las posibles soluciones, sería deseable disponer este orden de generación de forma que las soluciones tubieran una alta posibilidad de aparecer tempranamente.*
- *Un* ***sistema de símbolos exhibiría inteligencia*** *en la medida que tuviera éxito haciendo esto.*
- *La inteligencia de un sistema con recursos de procesamiento limitados consiste en* ***realizar elecciones acertadas sobre que hacer a continuación****.*

Acciones para construir un sistema que resuelva un problema

- Definir el problema con precisión
  - de que se parte
  - cual es el objetivo
- Analizar el problema
  - información para elegir las técnicas
- Aislar y representar el conocimiento necesario para resolver el problema
- Elegir la mejor técnica que resuelva el problema y aplicarla al problema particular

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Taxonomía de técnicas de resolución de problemas



BÚSQUEDA EN EL ESPACIO DE ESTADOS

- Objetivos:
  - Definir: espacio del problema, operadores, búsqueda en el espacio de estados, estado objetivo
  - Implementar en Common Lisp
- Indice:
  - Resolución de problemas
    - Ejemplo: problema del 8-puzzle
  - Representación de aspectos del espacio de estados
  - Implementación del problema de las garrafas
    - estados, operadores y su aplicación, ejemplo
  - Implementación del problema del granjero
    - estados, operadores y su aplicación, ejemplo
  - Problema del tren
- Lecturas:
  - E. Rich y K. Knight, 2.1, 2.2, 2.3
  - Luger/Stubblefield, 3.0, 3.1
  - N. Nilsson, 1.1

Resolución de problemas

- Un poblema consiste en:
  - Una descripción del estado inicial del mundo
  - Una descripción de acciones que puedan transformar un estado del mundo en otro

    operadores
  - Una descripción (parcial) del estado del mundo deseado
- Asunciones:
  - Nosotros somos el único agente del mundo
  - Nosotros conocemos el efecto de nuestras acciones y cuando pueden ser aplicadas
- Ejemplos ¨manejables¨:
  - El 8-puzzle
  - El problema de las garrafas de vino
  - El problema del granjero, el lobo, el carnero y la lechuga
  - El problema del viajante
  - Ajedrez (este bastante menos manejable)

El problema del 8-puzzle

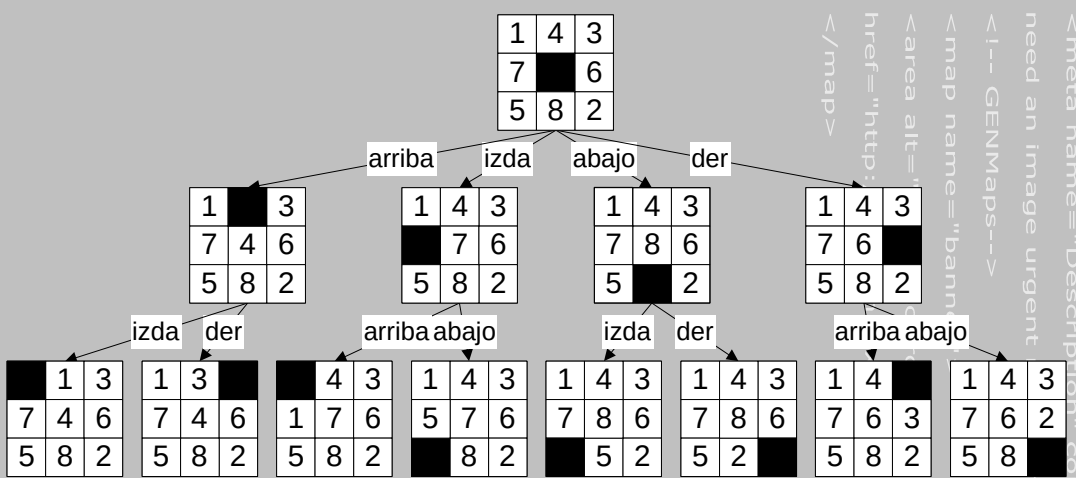
- El problema consiste en:
  - Situación inicial, p.e.:

1	4	3
7		6
5	8	2

  - Estado objetivo

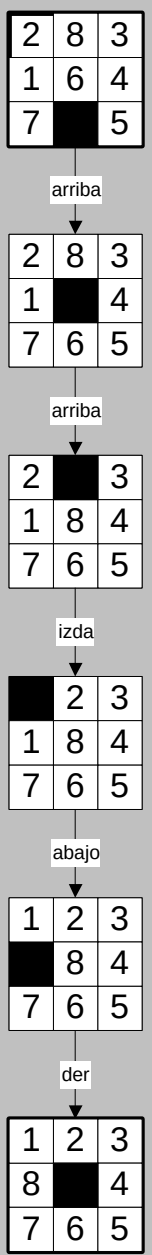
1	2	3
8		4
7	6	5

  - Reglas de juego
    - R1: Mover hueco arriba
    - R2: Mover hueco abajo
    - R3: Mover hueco derecha
    - R4: Mover hueco izquierda



Solución deseada en un problema del 8-puzzle

- La solución consiste en:
  - Una secuencia de operadores que transforman el estado inicial en el estado final

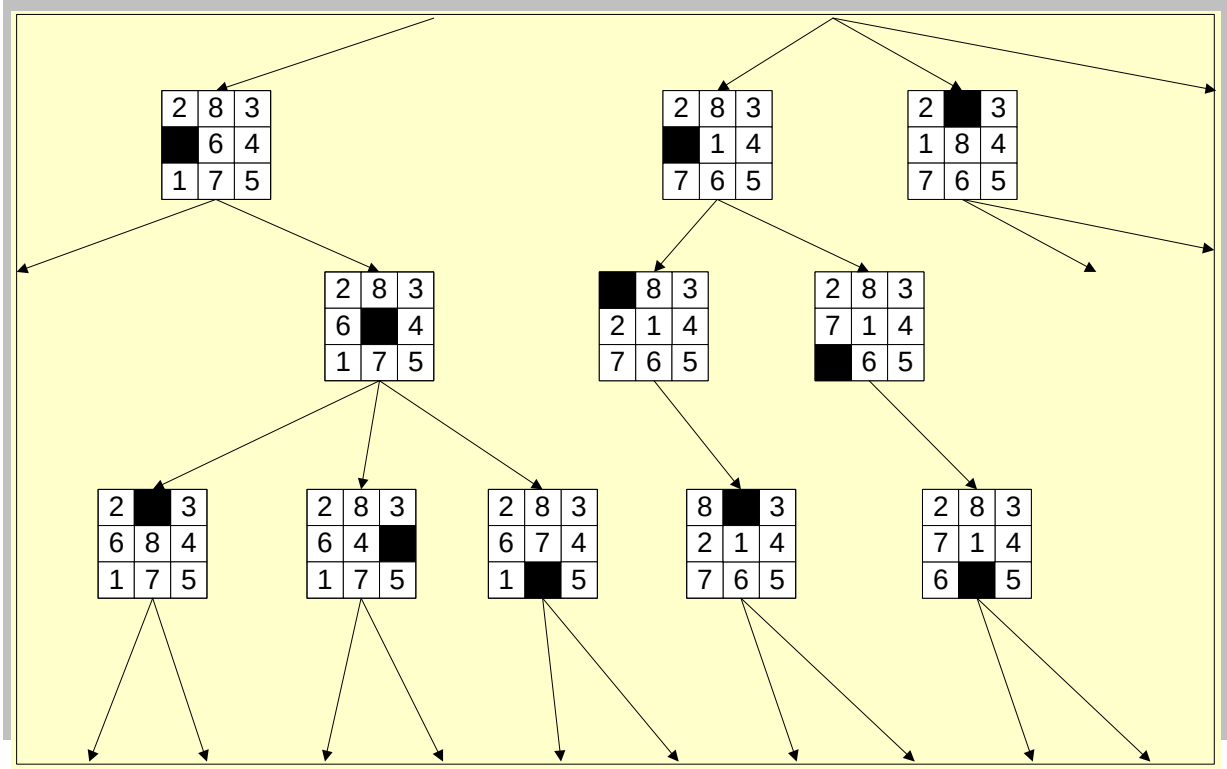


Representación del estado

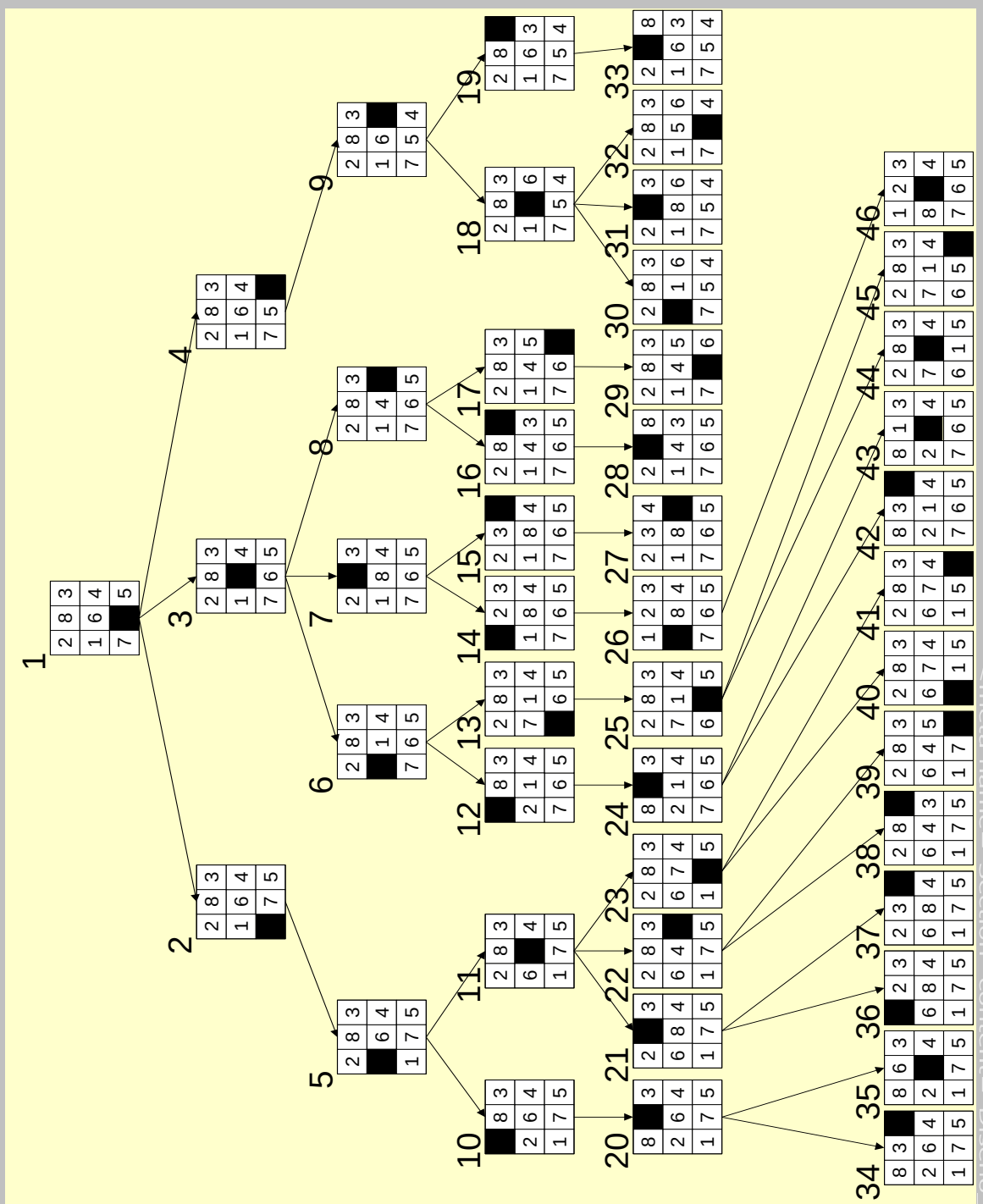
- Un espacio de estados debería describir
  - Todo lo que es necesario para resolver el problema
  - Nada que no sea necesario para resolver el problema
- Para el problema del 8-puzzle
  - Localización de cada casilla y la blanca
  - Nada mas
- Muchas representaciones posibles
  - Matriz de 3X3
  - Vector de longitud nueve: 2,1,3,4,8,7,6,5,blanco
  - Un conjunto de hechos
    - {(superior-izda = 2), (superior-centro = 1), ...}
- Elige la representaciòn que resulte más fácil de implementar

Representación del espacio de estados

- La búsqueda no supone que el árbol (o grafo) esté previamente generado
- No supone que se tengan que generar todos los nodos (sólo los necesarios para resolver el problema)
  - el arbol del 8-puzzle tiene 362.880 nodos¡
- Se guardan en listas de símbolos (objetos) relacionados

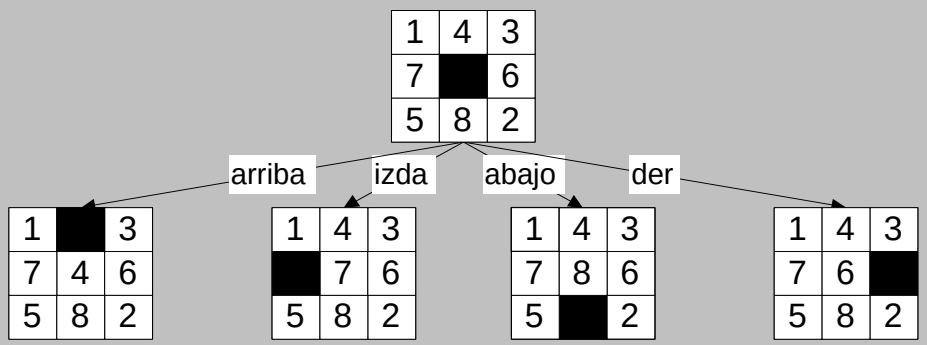


Un espacio de búsqueda para el 8-puzzle



Operadores

- Acciones simples que pueden transformar un estado en otro
- Los operadores consisten en:
  - Precondición: Descripción parcial del estado del mundo que debe ser verdad para realizar una acción
  - Instrucciones para crear el nuevo estado



- Los operadores deben ser tan generales como sea posible para así reducir el número de operadores
  - 4X9! operadores para pasar de un estado cualquiera a sus, como máximo 4, estados sucesores
    - if (2,1,3,4,8,7,6,5, _) then (2,1,3,4,8,_,6,5, 7)
  - 4X8 operadores que mueven cualquier ficha arriba, abajo, derecha o izquierda
  - 4 operadores que mueven el hueco arriba, abajo, derecha o izquierda

El estado objetivo y la solución del problema

- Una descripción de un estado deseado del mundo
- Implementaciones típicas
  - Una función que recibe un estado y devuelve verdad si el estado es el estado objetivo y falso en otro caso
  - Una descripción parcial del estado del mundo
- Resolución del problema: encontrar una secuencia de operadores que transformen el estado inicial del mundo en el estado objetivo
  - Encontrando la secuencia más corta
  - Encontrando la secuencia menos cara
  - Encontrando cualquier secuencia lo más rápido posible
- Evaluación de una estratégia
  - tiempo de cálculo <=> calidad de la solución

Estrategias de control

- ¿Como se decide que operador a aplicar?
  - ¿se puede aplicar?
  - ¿produce algún cambio en el estado?
  - ¿que estado elegir para aplicar los operadores?
  - ¿que pasa si hay varios operadores posibles a aplicar?
  - se tienen que poder aplicar de forma sistemática
- Hacia delante / hacia atrás / bidireccional
- Irrevocable / tentativas
- Informada / no informada

Los sistemas de producción

- La búsqueda es un proceso muy general en procesos inteligentes
- Este proceso se ha generalizado y estructurado en lo que se llaman:

SISTEMAS DE PRODUCCIÓN

Consiste en:

- Una base de datos/conocimiento con información sobre el problema
- Conjunto de reglas (operadores), generalmente con:
  - parte izquierda (patrón): determina la aplicabilidad de la regla
  - parte derecha: describe la operación a llevar a cabo al aplicar la regla
- Una estratégia de control que especifique
  - el orden en que se comprueba la aplicabilidad de las reglas
  - la forma de resolver los conflictos cuando varias reglas se pueden aplicar a un mismo estado
- Un aplicador de reglas
  - ciclo de reconocimiento-actuación

Ventajas de los sistemas de producción

- Separación de conocimiento y control
  - reglas
  - ciclo de reconocimiento-actuación
- Modularidad de las reglas/operadores
  - no hay interacciones sintácticas entre reglas
  - sólo modificando la memoria de trabajo
- Control dirigido por patrones
  - programas en IA necesitan flexibilidad
  - reglas pueden dispararse en cualquier secuencia
  - estado > reglas aplicables > camino de la solución
- Traza y explicación
  - secuencia de aplicación de reglas
  - cada reglas es una pieza de conocimiento con su justificación para un cambio de estado
- Independencia del lenguaje
  - el modelo es independiente de la representación elegida para reglas y memoria de trabajo
  - siempre que la representación soporte el reconocimiento de patrones
- Modelo plausible del mecanismo humano de resolución de problemas

Intérpretes generales de sistemas de producción

- Lenguajes basados en reglas
  - OPS5 (Brownston ... 1985) CMU
    - (en CLisp, RETE)
  - CLIPS (NASA)
    - (parece Lisp pero es C)
- Lenguajes basados en lógica
  - PROLOG
    (problemas para razonamiento probabilístico, evidencia incierta, asunciones por defecto, ...)
- Armazones de sistemas expertos
  - EMYCIN
  - en entornos de IA
    - KEE, KnowledgeCraft, LOOPS, ...
- Arquitecturas generales de resolución de problemas
  - GPS (Newell, Shaw & Simon) 1963
  - SOAR (Newell, Laird, ...) ... 1987

El problema de las Torres de Hanoi

- Hay tres palos, etiquetados como A, B, y C
- Hay tres discos en A. El disco de arriba tiene un diámetro de 1, el del medio de 2 y el de abajo de 3.
- No hay discos en C ni en B
- Sólo se puede mover un disco a la vez y no se puede situar un disco más grande sobre otro más pequeño

- El objetivo es mover todos los discos de A a C

El problema de las garrafas de vino

- Esblecimiento del problema
  - Se tienen dos garrafas, una de cuatro litros y otra de tres y un depósito con suficiente vino.
  - ¿Como se puede lograr tener exactamente dos litros de vino en la garrafa de cuatro litros de capacidad?.

El problema del granjero, el lobo, el carnero y la lechuga

- Establecimiento del problema
  - Un granjero tiene un lobo, un carnero, y una lechuga en la orilla derecha de un río.
  - Quiere llevarlos a la orilla izquierda.
  - Tiene una barca con la que puede cruzar el río y puede llevar una, y solo una, de las cosas.
  - El lobo se comerá el carnero si los deja juntos inatendidos.
  - El carnero se comerá la lechuga si los deja juntos inatendidos.

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

El problema del tren

- Establecimiento del problema
  - Un viajante se encuentra en una capital
  - Quiere viajar a otra capital
  - Hay un tren entre capitales de provincia colindantes

El problema de las garrafas de vino

- Establecimiento del problema
  - Se tienen dos garrafas, una de cuatro litros y otra de tres y un depósito con suficiente vino.
  - ¿Como se puede lograr tener exactamente dos litros de vino en la garrafa de cuatro litros de capacidad?.

- Espacio de estados del problema
  - Representación del estado: (x, y)
    - x: contenido en litros de la garrafa de 4 litros
    - y: contenido en litros de la garrafa de 3 litros
  - Estado inicial: (0, 0)
  - Estado objetivo: (2, n)
  - Operadores:
    - Llenar la garrafa de 3 litros desde el depósito
    - Llenar la garrafa de 4 litros desde el depósito
    - Vaciar la garrafa de 3 litros en el depósito
    - Vaciar la garrafa de 4 litros en el depósito
    - Echar lo que se pueda de la garrafa de 4 en la de 3
    - Echar lo que se pueda de la garrafa de 3 en la de 4

Estados para el problema de las garrafas de vino

- Representación del estado

```
;;; Representacion del estado como una lista
;;; con 2 componentes:
;;;
(defun make-garrafas (&key tres cuatro)
   (list tres cuatro))

;;; Primitivas de acceso a la estructura de
;;; datos del estado
;;;
(defun garrafas-tres (s) (first s))
(defun garrafas-cuatro (s) (second s))
```

- Estado inicial

```
(defparameter *estado-inicial*
  (make-garrafas :cuatro 0
                 :tres 0))
```

- Estado objetivo

```
(defun estado-objetivop (estado)
  ¨Una descripción del estado es la solucion
   si la garrafa de 4 litros tiene exactamente 2¨
  (= 2 (garrafas-cuatro estado)))
```

Operadores para el problema de las garrafas

- Conjunto de operadores posibles

```
(setq *operadores*
   '(llena-cuatro
     llena-tres
     vacia-cuatro
     vacia-tres
     echa-la-cuatro-a-la-tres
     echa-la-tres-a-la-cuatro))
```

- Implementación de los operadores
  - precondiciones: ¿puede ser aplicado el operador?
  - accion: generación de un nuevo estado

```
(defun llena-cuatro (s)
   (if (< (garrafas-cuatro s) 4)
       (make-garrafas :tres (garrafas-tres s)
                      :cuatro 4)))
```

precondición

acción

```
(defun vacia-cuatro (s)
   (if (> (garrafas-cuatro s) 0)
       (make-garrafas :tres (garrafas-tres s)
                      :cuatro 0)))
```

Operadores más complejos del problema de las garrafas

```
(defun echa-la-cuatro-a-la-tres (s)
   (let* ((antes-3 (garrafas-tres s))
          (antes-4 (garrafas-cuatro s))
          (se-echa (if (> (+ antes-3 antes-4) 3)
                       (- 3 antes-3)
                       antes-4)))
      (and (> antes-4 0)      ;algo en 4
           (< antes-3 3)      ;3 no esta llena
           (make-garrafas
              :tres (+ antes-3 se-echa)
              :cuatro (- antes-4 se-echa)))))
```

precondición

acción

```
(defun echa-la-tres-a-la-cuatro (s)
   (let* ((antes-3 (garrafas-tres s))
          (antes-4 (garrafas-cuatro s))
          (se-echa (if (> (+ antes-3 antes-4) 4)
                       (- 4 antes-4)
                       antes-3)))
      (and (> antes-3 0)      ;algo en 3
           (< antes-4 4)      ;4 no esta llena
           (make-garrafas
              :cuatro (+ antes-4 se-echa)
              :tres (- antes-3 se-echa)))))
```

Aplicando los operadores en el problema de las garrafas

```
> (make-garrafas :cuatro 0
                 :tres 0)
(0 0)
> (make-garrafas :cuatro 2
                 :tres 2)
(2 2)
> (llena-cuatro
    (make-garrafas :cuatro 0
                   :tres 0))
(0 4)
> (vacia-tres
   (echa-la-cuatro-a-la-tres
    (llena-cuatro
     (make-garrafas :cuatro 0
                    :tres 0))))
(0 1)
> (estado-objetivop
            (echa-la-cuatro-a-la-tres
             (llena-cuatro
              (echa-la-cuatro-a-la-tres
               (vacia-tres
                (echa-la-cuatro-a-la-tres
                 (llena-cuatro
                  (make-garrafas :cuatro 0
                                 :tres 0)))))))
T
>
```

El problema del granjero, el lobo, el carnero y la lechuga

- Esblecimiento del problema
  - Un granjero tiene un lobo, un carnero, y una lechuga en la orilla derecha de un rio.
  - Quiere llevarlos a la orilla izquierda.
  - Tiene una barca con la que puede cruzar el rio y puede llevar una, y solo una, de las cosas.
  - El lobo se comerá el carnero si los deja juntos inatendidos.
  - El carnero se comerá la lechuga si los deja juntos inatendidos.

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Introduction">
<meta name="Diseño_Gráfico">
<meta name="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Estados para el problema del granjero

- Representación del estado:
  (gra lob car lec)
  - gra: orilla donde está el granjero (der. o izda.)
  - lob: orilla donde está el lobo
  - car: orilla donde está el carnero
  - lec: orilla donde están las lechugas
- Estructura de datos del estado

```
(defun make-situacion
      (&key granjero lobo carnero lechugas)
  (list granjero lobo carnero lechugas))

;;;Primitivas de acceso
(defun situacion-granjero (s) (first s))
(defun situacion-lobo (s) (second s))
(defun situacion-carnero (s) (third s))
(defun situacion-lechugas (s) (fourth s))
```

- Estado inicial

```
(defparameter *estado-inicial*
              (make-situacion
                  :granjero 'izda
                  :lobo 'izda
                  :carnero 'izda
                  :lechugas 'izda))
```

- Estado objetivo

```
(defun estado-objetivop (estado)
  "Una descripción de estado es la solución si
    todos estan en la margen derecha"
  (equal estado '(dere dere dere dere)))
```

Funciones auxiliares de los operadores para el problema del granjero

```
(defun margen-opuesta (margen)
   ;;devuelve el lado opuesto del rio
   (cond ((eql margen 'izda) 'dere)
         ((eql margen 'dere) 'izda)))



(defun situacion-segurap (s)
   ;;;Devuelve nil si no es una situacion segura
   ;;; y devuelve el estado si es sugura
   (cond ((and (eql (situacion-lobo s)
                    (situacion-carnero s))
               (not (eql (situacion-granjero s)
                         (situacion-lobo s))))
          nil)
         ((and (eql (situacion-lechugas s)
                    (situacion-carnero s))
               (not (eql (situacion-granjero s)
                         (situacion-carnero s))))
          nil)
         (t s)))
```

Operadores para el problema del granjero

- Conjunto de operadores posibles

```
(setq *operadores*
   '(el-granjero-va-solo
     el-granjero-lleva-al-lobo
     el-granjero-lleva-al-carnero
     el-granjero-lleva-a-las-lechugas))
```

- Implementación de los operadores

```
(defun el-granjero-va-solo (s)
   (situacion-segurap
      (make-situacion
         :granjero (margen-opuesta
                     (situacion-granjero s))
         :lobo     (situacion-lobo s)
         :carnero  (situacion-carnero s)
         :lechugas (situacion-lechugas s)
         )))

(defun el-granjero-lleva-al-lobo (s)
   (situacion-segurap
      (if (eql (situacion-granjero s)
               (situacion-lobo s))
         (make-situacion
            :granjero (margen-opuesta
                        (situacion-granjero s))
            :lobo     (margen-opuesta
                        (situacion-lobo s))
            :carnero  (situacion-carnero s)
            :lechugas (situacion-lechugas s)
            ))))
```

Aplicando los operadores en el problema del granjero

```
> (setq s (make-situacion
           :granjero 'izda
           :lobo 'izda
           :carnero 'izda
           :lechugas 'izda))
(IZDA IZDA IZDA IZDA)
>
>
> (el-granjero-va-solo s)
NIL
>
>
> (el-granjero-lleva-al-carnero s)
(DERE IZDA DERE IZDA)
>
>
> (el-granjero-lleva-al-carnero
   (el-granjero-va-solo
      (el-granjero-lleva-a-las-lechugas
       (el-granjero-lleva-al-carnero
        (el-granjero-lleva-al-lobo
         (el-granjero-va-solo
            (el-granjero-lleva-al-carnero s)))))))
(DERE DERE DERE DERE)
>
```

El problema del tren

- Establecimiento del problema
  - Un viajante se encuentra en una capital
  - Quiere viajar a otra capital
  - Hay un tren entre capitales de provincia colindantes

Estados y operadores para el problema del tren

- Representación del estado: un símbolo de lisp: la ciudad de partida

- Estado inicial

```
(defparameter *estado-inicial* ‘Zaragoza)
```

- Estado objetivo

```
(defparameter *estado-objetivo* ‘Murcia)

(defun estado-objetivop (estado)
  ¨Una descripción del estado es la solucion
   si es la ciudad de destino¨
  (eql estado *estado-objetivo*))
```

- Operadores (50)
  - Ir de una capital a otra capital

```
(defparameter *operadores*
  ‘(GOTO-ALBACETE GOTO-ALMERIA ...
    GOTO-ZAMORA GOTO-ZARAGOZA)))

(defun goto-albacete (state)
  (when (member state
          '(Valencia Alicante Murcia Granada
            Jaen Ciudad-Real Cuenca)
        'Albacete))
```

Problema del tren 2

- Establecimiento del problema
  - Estas en Teruel
  - Quieres ir a cualquier sitio cálido
  - Hay un tren entre capitales de estados colindantes
- Estado inicial

```
(defparameter *estado-inicial* 'Teruel)
```

- Estado objetivo

```
(defparameter *capitales-calidas*
  '(Almeria Malaga ...)

(defun estado-objetivop (estado)
  (member estado *capitales-calidas*))
```

El problema del 8-puzzle

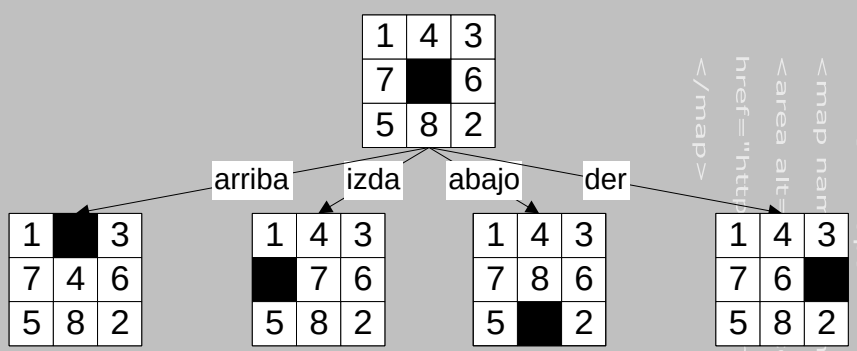
- El problema consiste en:
  - Situación inicial, p.e.:

1	4	3
7		6
5	8	2

  - Estado objetivo

1	2	3
8		4
7	6	5

  - Reglas de juego
    - R1: Mover hueco arriba
    - R2: Mover hueco abajo
    - R3: Mover hueco derecha
    - R4: Mover hueco izquierda



Estados para el problema del 8-puzzle

- Representación del estado

```
(setq estado
  (make-array '(3 3)
              :initial-contents '((1   2   3)
                                  (8 space 4)
                                  (7   6   5))))
(setf (aref estado 1 2) 'space)
(aref estado 1 2)
> space
```

  - se puede elegir aleatoriamente (más adelante)

- Estado inicial

2	8	3
	6	4
1	7	5

```
(defparameter *estado-inicial*
  (make-array '(3 3)
              :initial-contents '((  2   8 3)
                                  (space 6 4)
                                  (  1   7 5))))
```

- Estado objetivo

1	2	3
8		4
7	6	5

```
(defparameter *estado-objetivo*
  (make-array '(3 3)
              :initial-contents '((1   2   3)
                                  (8 space 4)
                                  (7   6   5))))


(defun estado-objetivop (estado)
  (equal estado *estado-objetivo*))
```

Funciones para manipular el tablero en el problema del 8-puzzle

- Para generar un nuevo estado se copia el tablero del anterior y se hace el cambio

```
(defun copia-tablero(tablero)
  (let ((new-tablero(make-array '(3 3))))
    (loop for i from 0 to 2
        do (loop for j from 0 to 2
                 do (setf (aref new-tablero i j)
                          (aref tablero i j) )))
    new-tablero))
```

- Para encontrar el emplazamiento de una pieza

```
(defun encuentra-pieza(x tablero)
"devuelve una lista con las coordenadas x y de
 la pieza x en el tablero"
  (loop for i from 0 to 2
      thereis (loop for j from 0 to 2
            thereis
              (when (eq (aref tablero i j) x)
                      (list i j)))))
```

- Para imprimir un estado

```
(defun imprime-tablero(tablero)
  (format t "~%-------")
  (loop for i from 0 to 2
     do (format t "~%|")
     (loop for j from 0 to 2
        do (format t "~A|"
             (if (eq (aref tablero i j) 'space)
                 " "
                 (aref tablero i j))))
        (format t "~%-------")))
```

Operadores para el problema del 8-puzzle

- Conjunto de operadores posibles

```
(setq *operadores*
   '(move-up
     move-down
     move-left
     move-right))
```

- Implementación de los operadores

```
(defun move-up(state)
  (let* ((at-space (encuentra-pieza 'espacio state))
         (i (first at-space))
         (j (second at-space))
         (new-state (copia-tablero state)))
    (when (> i 0)
      (setf (aref new-state i j)
            (aref new-state (- i 1) j))
      (setf (aref new-state (- i 1) j)
            'espacio)
      new-state)))
```

Elegir una situación inicial aleatoriamente

- Para hacer un movimiento aleatorio

```
(defun movimiento-aleatorio(state)
"Coge aleatoriamente uno de los 4 operadores.
 Si el operador no es aplicable, elige otra vez"
  (let ((r (random 4)))
    (or (cond ((= r 0)(move-left state))
        ((= r 1) (move-right state))
        ((= r 2) (move-up state))
        ((= r 3) (move-down state)))
      (movimiento-aleatorio state))))



(defun movimientos-aleatorios (n state)
  "hace n movimientos aleatorios"
  (loop for i from 1 to n
        do (setq state
                 (movimiento-aleatorio state)))
  state)



(defparameter *estado-inicial*
  (movimientos-aleatorios 20 *estado-objetivo*))
```

Código

```
(defun movimiento-aleatorio(state)
"Coge aleatoriamente uno de los 4 operadores.
 Si el operador no es aplicable, elige otra
vez"
  (let ((r (random 4)))
    (or (cond ((= r 0)(move-left state))
        ((= r 1) (move-right state))
        ((= r 2) (move-up state))
        ((= r 3) (move-down state)))
      (movimiento-aleatorio state))))


(defun movimientos-aleatorios (n state)
  "hace n movimientos aleatorios"
  (loop for i from 1 to n
        do (setq state
                 (movimiento-aleatorio state)))
  state)


(defparameter *estado-inicial*
  (movimientos-aleatorios 20 *estado-
objetivo*))
```

Contexto de desarrollo

- A través de los años los filósofos y matemáticos han propuesto muchas formas diferentes de lógica. Objetivo:
  - Caracterizar los principios del razonamiento correcto (formalidad, completitud, ...)
    - Su énfasis está en conseguir operaciones en las que se preserve la verdad sobre expresiones bien formadas
      - esto es para probar
        - y esto también
- Las líneas de trabajo de psicólogos y lingüistas tratan de caracterizar la naturaleza del entendimiento humano. Objetivo:
  - Describir la forma en que los humanos adquieren y usan su conocimiento del mundo
- (Los informáticos no han acordado todavía que forma de
lógica es, o sería, apropiada para representar conocimiento del mundo real.)

ESTRATEGIAS DE CONTROL CIEGAS

- Objetivos:
  - Ver como
    - aplicar los operadores
    - guardar los nodos y los caminos
    - integrar estrategias de búsqueda
    - escribir el comino de la solucion
  - Aprender estratégias básicas de búsqueda
  - Aprender a implementar la búsqueda primero en anchura en Lisp
- Indice:
  - Algoritmo de búsqueda primero en anchura
    - Implementación
  - Algoritmo de búsqueda primero en profundidad
    - BP con límite de profundidad
    - BP profundización iterativa
  - Búsqueda bidireccional
  - Comparaciones
- Lecturas:
  - E. Rich y K. Knight, Cap 2 (excepto 2.4)
  - N. Nilsson, 2.0, 2.3

Algoritmo genérico de búsqueda

```
PRINCIPIO
1   ABIERTOS := (nodo_inicial)
    RESUELTO := falso
2   mientras que ABIERTOS <> ()
              AND not RESUELTO hacer
3     N := quitar primer elemento de ABIERTOS
      E := estado asociado a N
4     si E es el estado solución
5       entonces RESUELTO := verdad
        sino
          para cada operador O hacer
6           si O se puede aplicar a E
             entonces crear nodo correspondiente
              al estado obtenido por aplicacion de
7             O a E y añadir ese nodo a ABIERTOS
    fmq
    si RESUELTO
8     entonces devuelve el estado objetivo (y si
                se requiere una explicacion, el
                camino para llegar a el)
9     sino informa de que el objetivo no puede ser
            alcanzado
  fsi
FIN
```

Búsquedas no informadas

- El tipo de búsqueda depende del paso número 7 del algoritmo anterior

BUSQUEDAS NO INFORMADAS

- El algoritmo no conoce nada del problema concreto que debe resolver => el paso 7 se realiza con criterios independientes del dominio del problema

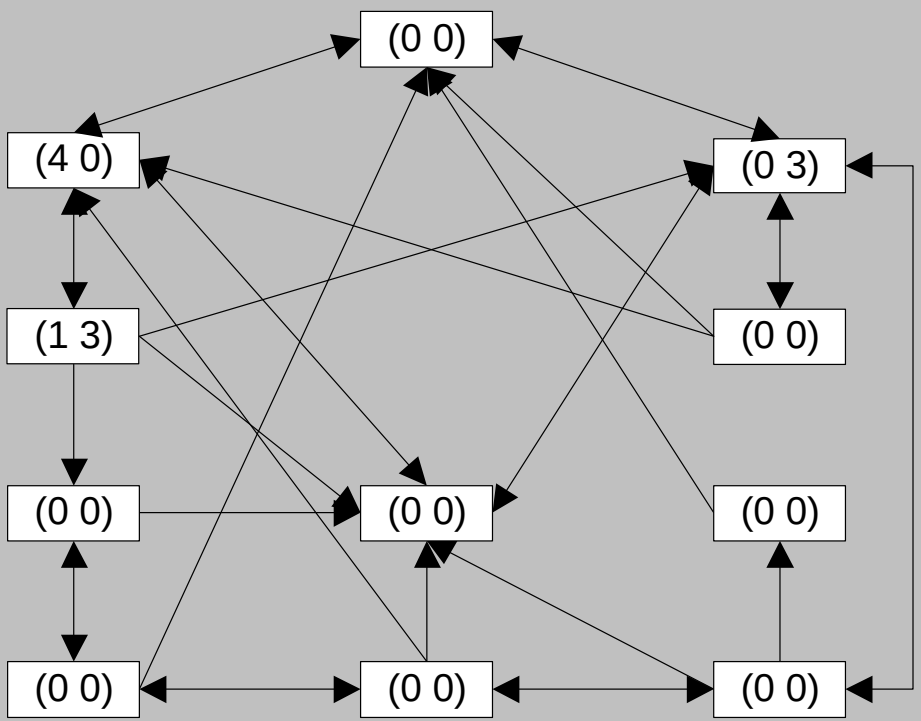
BUSQUEDA PRIMERO EN ANCHURA

- Paso nº 7:
  - Los nodos ABIERTOS se colocan en orden a su profundidad en el árbol. Los menos profundos al principio, los más profundos al final.
  - Los nodos de igual profundidad se ordenan arbitrariamente.

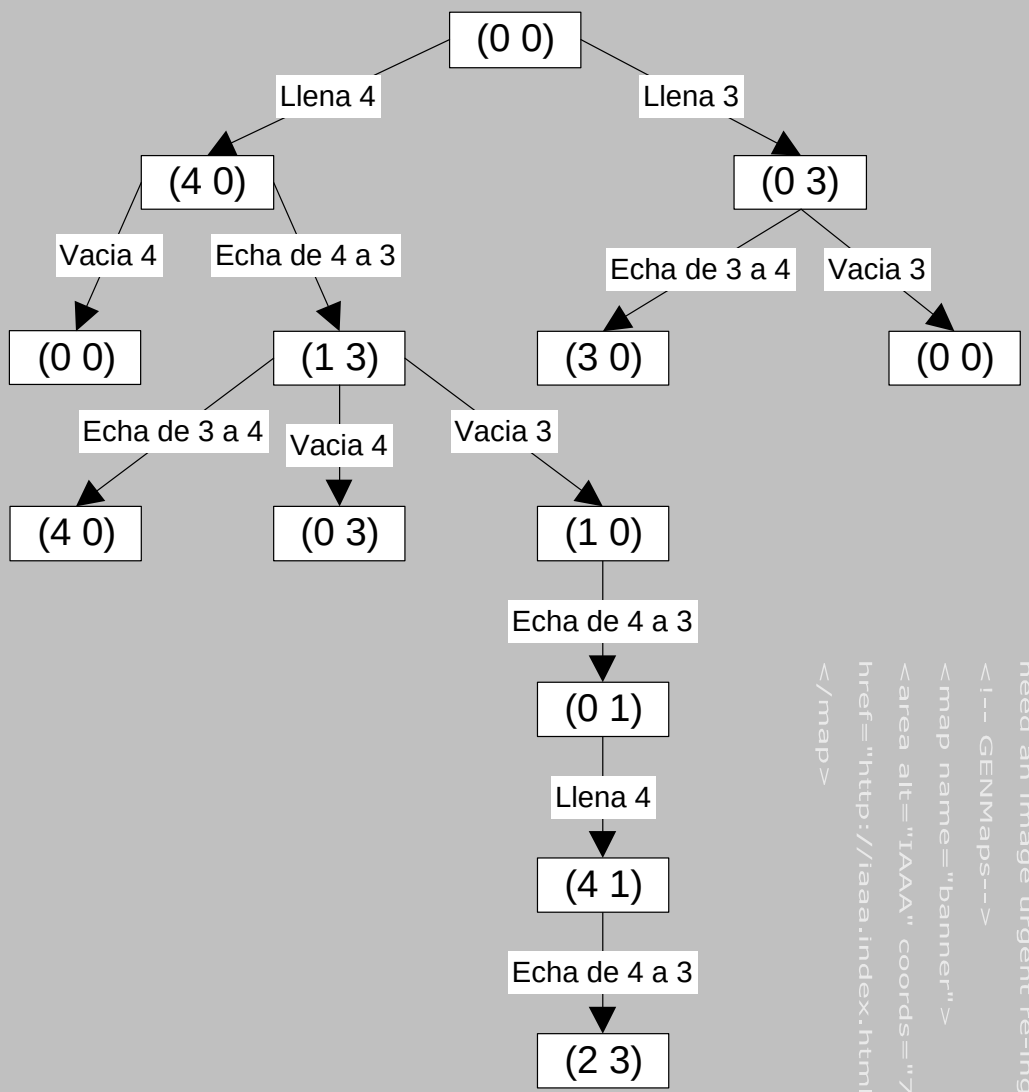
- Profundidad de un nodo:
  - número de arcos del camino que lo une al nodo raiz

- Mediante esta estrategia, el árbol se va generando por niveles de profundidad. Hasta que todos los nodos de un nivel no han sido revisados, no se revisa ninguno del siguiente nivel

Espacio de estado vs árbol de búsqueda



Arbol de búsqueda problema de las garrafas (4 3)



Resultado de la búsqueda en anchura en el problema de las garrafas

- Proceso de búsqueda

```
Añadiendo (0 0) al nivel 1
NODO-INICIAL
expandiendo el estado --->(0 0)
Añadiendo (0 4) al nivel 2
Añadiendo (3 0) al nivel 2
expandiendo el estado --->(0 4)
Añadiendo (3 4) al nivel 3
Añadiendo (3 1) al nivel 3
expandiendo el estado --->(3 0)
Añadiendo (0 3) al nivel 3
expandiendo el estado --->(3 4)
expandiendo el estado --->(3 1)
Añadiendo (0 1) al nivel 4
expandiendo el estado --->(0 3)
Añadiendo (3 3) al nivel 4
expandiendo el estado --->(0 1)
Añadiendo (1 0) al nivel 5
expandiendo el estado --->(3 3)
Añadiendo (2 4) al nivel 5
expandiendo el estado --->(1 0)
Añadiendo (1 4) al nivel 6
expandiendo el estado --->(2 4)
Añadiendo (2 0) al nivel 6
expandiendo el estado --->(1 4)
Añadiendo (3 2) al nivel 7
expandiendo el estado --->(2 0)
Añadiendo (0 2) al nivel 7
```

- Solución al problema

```
Resuelto ! El camino de la solucion es:
(ESTADO-INICIAL            (0 0))
(LLENA-CUATRO              (0 4))
(ECHA-LA-CUATRO-A-LA-TRES  (3 1))
(VACIA-TRES                (0 1))
(ECHA-LA-CUATRO-A-LA-TRES  (1 0))
(LLENA-CUATRO              (1 4))
(ECHA-LA-CUATRO-A-LA-TRES  (3 2))
```

Resultado en otro problema de las garrafas

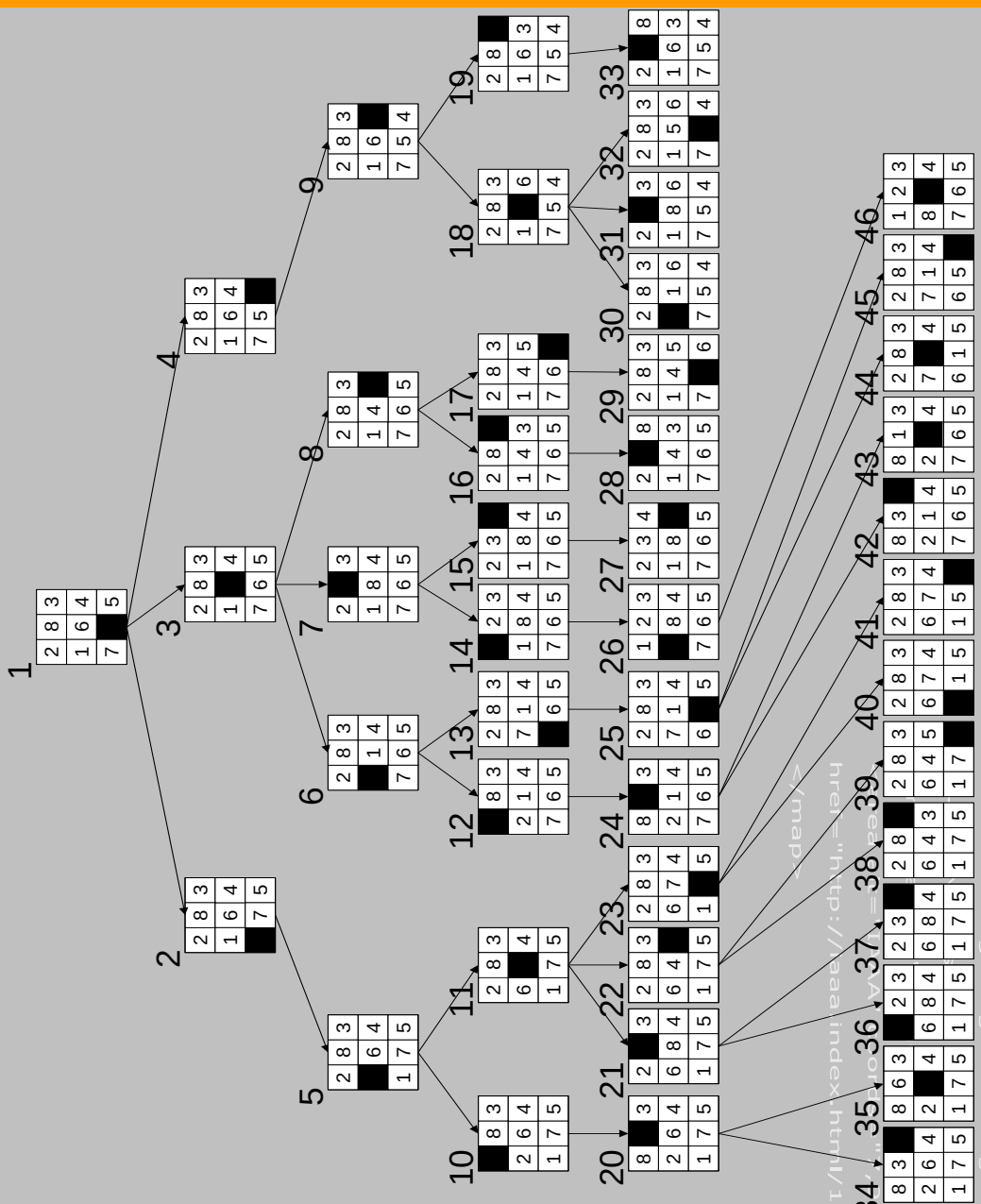
- Problema:
  - Se tiene:
    - Una garrafa de 9 litros
    - Una garrafa de 7 litros
  - Se quiere:
    - Se quieren 3 litros en la de 9

- Cambios?
  - poniendo los límites como constantes
  - poniendo nuevos operadores

- Solución al problema

```
Resuelto ! El camino de la solucion es:
(ESTADO-INICIAL              (0 0))
(LLENA-TRES                  (7 0))
(ECHA-LA-TRES-A-LA-CUATRO    (0 7))
(LLENA-TRES                  (7 7))
(ECHA-LA-TRES-A-LA-CUATRO    (5 9))
(VACIA-CUATRO                (5 0))
(ECHA-LA-TRES-A-LA-CUATRO    (0 5))
(LLENA-TRES                  (7 5))
(ECHA-LA-TRES-A-LA-CUATRO    (3 9))
(VACIA-CUATRO                (3 0))
(ECHA-LA-TRES-A-LA-CUATRO    (0 3))
```

Búsqueda primero en anchura en el problema del 8-puzzle



- **Los números muestran el orden en que los nodos son examinados**

Resultado en el problema del granjero

Proceso de búsqueda

```
Añadiendo (IZDA IZDA IZDA IZDA) al nivel 1
NODO-INICIAL
expandiendo el estado --->(IZDA IZDA IZDA IZDA)
Añadiendo (DERE IZDA DERE IZDA) al nivel 2
expandiendo el estado --->(DERE IZDA DERE IZDA)
Añadiendo (IZDA IZDA DERE IZDA) al nivel 3
expandiendo el estado --->(IZDA IZDA DERE IZDA)
Añadiendo (DERE DERE DERE IZDA) al nivel 4
Añadiendo (DERE IZDA DERE DERE) al nivel 4
expandiendo el estado --->(DERE DERE DERE IZDA)
Añadiendo (IZDA DERE IZDA IZDA) al nivel 5
expandiendo el estado --->(DERE IZDA DERE DERE)
Añadiendo (IZDA IZDA IZDA DERE) al nivel 5
expandiendo el estado --->(IZDA DERE IZDA IZDA)
Añadiendo (DERE DERE IZDA DERE) al nivel 6
expandiendo el estado --->(IZDA IZDA IZDA DERE)
expandiendo el estado --->(DERE DERE IZDA DERE)
Añadiendo (IZDA DERE IZDA DERE) al nivel 7
expandiendo el estado --->(IZDA DERE IZDA DERE)
Añadiendo (DERE DERE DERE DERE) al nivel 8
```

Solución al problema

```
Resuelto ! El camino de la solucion es:
(ESTADO-INICIAL                      (IZDA IZDA IZDA IZDA))
(EL-GRANJERO-LLEVA-AL-CARNERO        (DERE IZDA DERE IZDA))
(EL-GRANJERO-VA-SOLO                 (IZDA IZDA DERE IZDA))
(EL-GRANJERO-LLEVA-AL-LOBO           (DERE DERE DERE IZDA))
(EL-GRANJERO-LLEVA-AL-CARNERO        (IZDA DERE IZDA IZDA))
(EL-GRANJERO-LLEVA-A-LAS-LECHUGAS    (DERE DERE IZDA DERE))
(EL-GRANJERO-VA-SOLO                 (IZDA DERE IZDA DERE))
(EL-GRANJERO-LLEVA-AL-CARNERO        (DERE DERE DERE DERE))
```

granjero carnero lechuga lobo

Estadísticas de la búsqueda en anchura en problemas anteriores

- Estadísticas del proceso de búsqueda
  - Problema de las garrafas

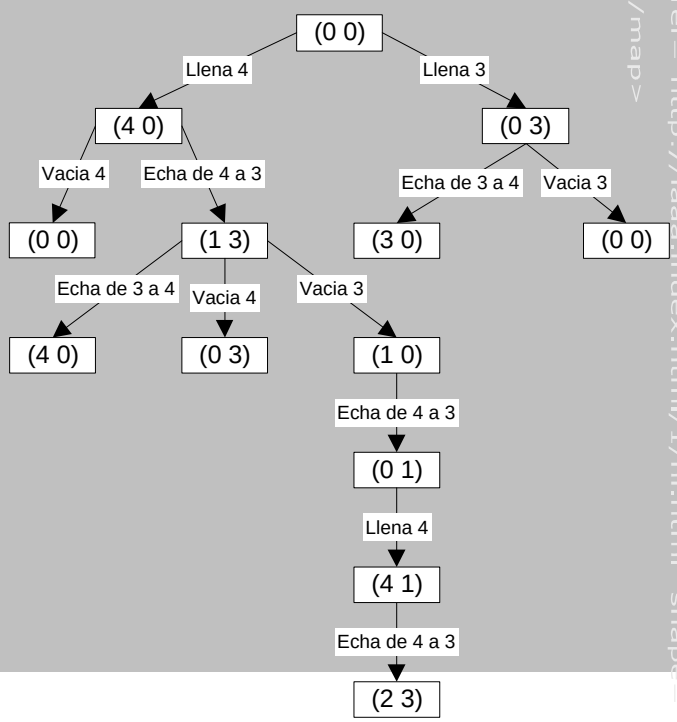
```
(NODOS-CHEQUEADOS 42)
(NODOS-CREADOS 13)
(NODOS-EXPANDIDOS 12)
(MAXIMA-LONGITUD-DE-LA-LISTA-DE-NODOS 3)
(LONGITUD-DE-LA-SOLUCION 7)
```

  - Problema del granjero

```
(NODOS-CHEQUEADOS 19)
(NODOS-CREADOS 9)
(NODOS-EXPANDIDOS 9)
(MAXIMA-LONGITUD-DE-LA-LISTA-DE-NODOS 2)
(LONGITUD-DE-LA-SOLUCION 8)
(MAXIMA-PROFUNDIDAD 8)
```

Complejidad de algoritmos de búsqueda

- Complejidad en tiempo
  - Número de nodos generados
- Complejidad de espacio
  - Máximal longitud de la lista de nodos
- Afectado por:
  - b: factor de ramificación (branching factor). Media del número de nodos generados desde un nodo
  - d: Profundidad. Número de aplicación de operadores necesarios para transformar el estado inicial en el estado final

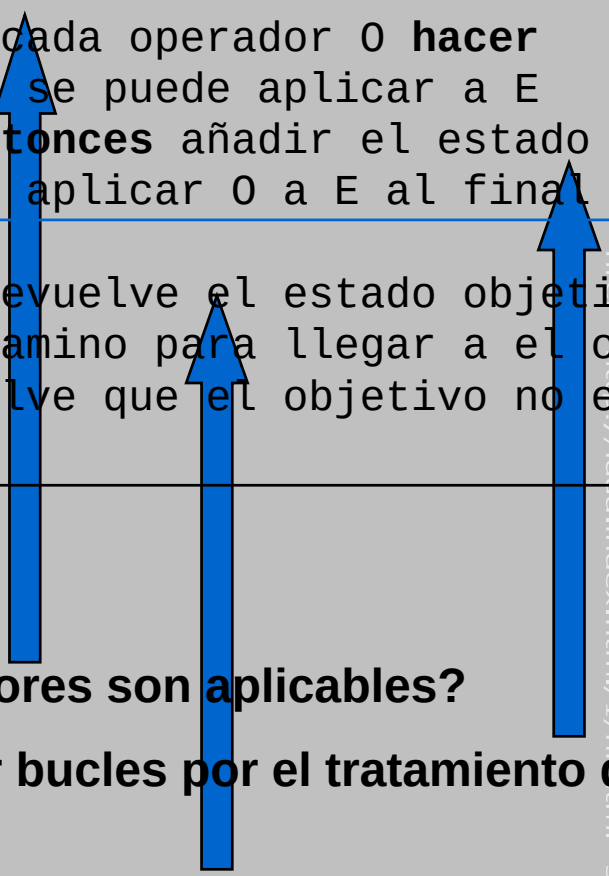


Complejidad del algoritmo de búsqueda en anchura

- Tiempo
  - $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Espacio
  - $O(b^d)$
- Búsqueda primero en anchura
  - Garantiza que encuentra la secuencia más corta de operadores
  - Su complejidad espacial lo hace impracticable para problemas grandes
  - Para el problema del 8-puzzle
    - $b = 3$, $d = 20$, $3^{20} = 3.486.784.401$
  - Para el problema de las garrafas de vino
    - $b = 3$, $d = 6$, $3^6 = 729$

Problemas para implementar el algoritmo

```
PRINCIPIO
  ABIERTOS = (nodo_inicial)
  RESUELTO = falso
  mientras que ABIERTOS <> ()
            AND not RESUELTO hacer
    E = quitar primer elemento de ABIERTOS
    si E es el estado solución
      entonces RESUELTO = verdad
       sino
         para cada operador O hacer
          si O se puede aplicar a E
            entonces añadir el estado obtenido al
                 aplicar O a E al final de ABIERTOS
  si RESUELTO
    entonces devuelve el estado objetivo y el
              camino para llegar a el objetivo
    sino devuelve que el objetivo no es posible
FIN
```



- ¿Que operadores son aplicables?
- ¿Como evitar bucles por el tratamiento de nodos repetidos?
- ¿Como reconstruir el camino de la solución?

Implementación de la búsqueda primero en anchura

```
PRINCIPIO
1  ABIERTOS := (nodo_inicial}    RESUELTO := falso
2  mientras que ABIERTOS <> () AND not RESUELTO hacer
3    N := quitar primer elemento de ABIERTOS
     E := estado asociado a N
4    si E es el estado solución
5      entonces RESUELTO := verdad
       sino para cada operador O hacer
6          si O se puede aplicar a E
7           entonces crear nodo correspondiente al estado obtenido por
                     aplicacion de O a E y añadir ese nodo a ABIERTOS
   si RESUELTO
8    entonces devuelve el estado objetivo (y si se requiere una
            explicacion, el camino para llegar a el)
9    sino informa de que el objetivo no puede ser alcanzado
FIN
```

```
(defun b ()
   (do* (                                          ;(2)
         (ABIERTOS                                 ;(1)
            (list (crea-nodo *estado-inicial* nil
                             'estado-inicial
                             'nodo-inicial))
            (reorganizar-nodos-a-expandir          ;(7)
              (expandir-nodo el-nodo)              ;(6)
              (rest ABIERTOS)))
          )
         (el-nodo                                  ;(3)
            (first ABIERTOS)
            (if ABIERTOS
               (first ABIERTOS))                   ;(4)
          )
         )
        (                                          ;(2)y(5)
         (or (if (endp ABIERTOS)
                 (mensaje-de-error))               ;(9)
             (if (estado-objetivop
                    (get el-nodo 'estado))
                 (escribe-solucion el-nodo)) ;(8)
         )
        ))
```

Estructura de datos y creación de un nodo

```
(defun crea-nodo (estado
                  padre
                  operacion
        &optional (simbolo (gensym "NODO-")))

  (setf (get simbolo 'estado) estado)
  (setf (get simbolo 'padre) padre)
  (setf (get simbolo 'operacion) operacion)

  (format t "~%Añadiendo ~S " estado)
  simbolo)
```

para guardar el estado

para recordar el camino

para recordar una explicación

para la traza de ejecución

Escribir el camino (8)

- Para escribir la solución (8)

Cuando es alcanzado un estado para el que esta resuelto el problema, ESCRIBE-SOLUCION devuelve el camino de la solucion.

Una solucion es una secuencia de mundos generados por movimientos legales que comienzan en el mundo inicial.

```
(defun escribe-solucion (solucion)
  (format t "~%~%Resuelto !
             El camino de la solucion es: ")
  (escribe-un-camino solucion)
  (print 'hecho)
  )


(defun escribe-un-camino (nodo)
  (if (get nodo 'padre)
      (escribe-un-camino (get nodo 'padre)))
  (print (list (get nodo 'operacion)
               (get nodo 'estado)))
 )
```

Objetivo imposible (9)

- En caso de no haber ningún nodo a examinar (9)

```
(defun mensaje-de-error ()
  (format t "~%~%ERROR!!!, no es posible
     seguir con el proceso de busqueda."))
```

¿Se ha logrado el objetivo? (5)

- Es una función dependiente del dominio de aplicación
  - ha de ser proporcionada por la aplicaciòn

```
;;;Test para encontrar el objetivo
;;;
(defun estado-objetivop (estado)
  )
```

- Ejemplo
  - Para el caso de las garrafas

```
(defun estado-objetivop (estado)
   "Una descripción de estado es la
    solución si la garrafa de cuatro
    litros tiene 2 litros"
  (= c2 (garrafas-cuatro estado)))
```

Expandir un nodo (6)

- Para cada estado del problema genera todas las siguientes estados posibles y los añade a una lista
- Si el nuevo estado es no-NIL lo añade a la lista

```
(defun expandir-nodo (nodo)
   (let* ((estado (get nodo 'estado)))
      (format t "~%expandiendo el estado --->~s"
                estado)
      (mapcan
         #'(lambda (x)
              (let ((nuevo-estado
                        (funcall x estado)))
                 (and nuevo-estado
                      (list (crea-nodo
                                nuevo-estado
                                nodo
                                x)))))
         *operadores*)))
```

se aplica el operador

si se hay un nuevo estado se mete en un nodo

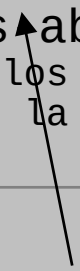
para dar una explicación interactiva

- ¡¡¡ PROBLEMAS !!!
  - Explosión de nodos

Aplicación de la estratégia (7)

- Reorganización de los nodos a expandir

```
(defun reorganizar-nodos-a-expandir
                (nodos-nuevos nodos-abiertos)

 (if (endp nodos-nuevos)
     nodos-abiertos
     ; si no hay nodos nuevos la lista ABIERTOS
     ;    es la de antes

     (append nodos-abiertos nodos-nuevos)))
     ; en otro caso, los nuevos nodos se añaden
     ;    al final de la lista
```



- En la búsqueda en anchura el nuevo nodo se pone al final para ser visitado después de todos los ya añadidos

BUSQUEDA PRIMERO EN PROFUNDIDAD

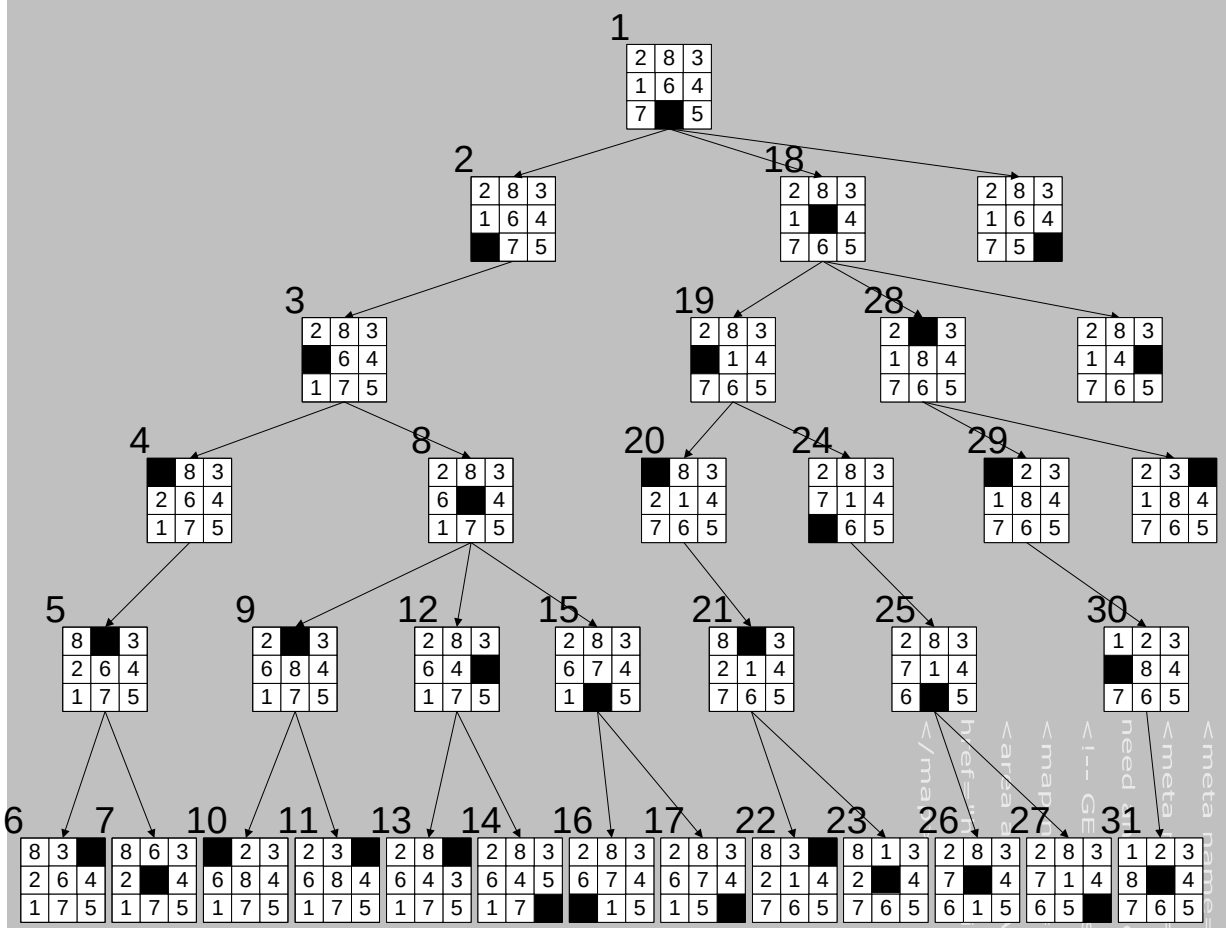
- Paso nº 7:
  - Los nodos ABIERTOS se colocan en orden inverso a su profundidad en el árbol. Los más profundos al principio, los menos profundos al final.
  - Los nodos de igual profundidad se ordenan arbitrariamente.

- Mediante esta estrategia se genera un camino hasta encontrar el objetivo o el límite de una rama, en este caso se retrocede y se prueba con caminos alternativos inmediatos.

- PROBLEMAS ...

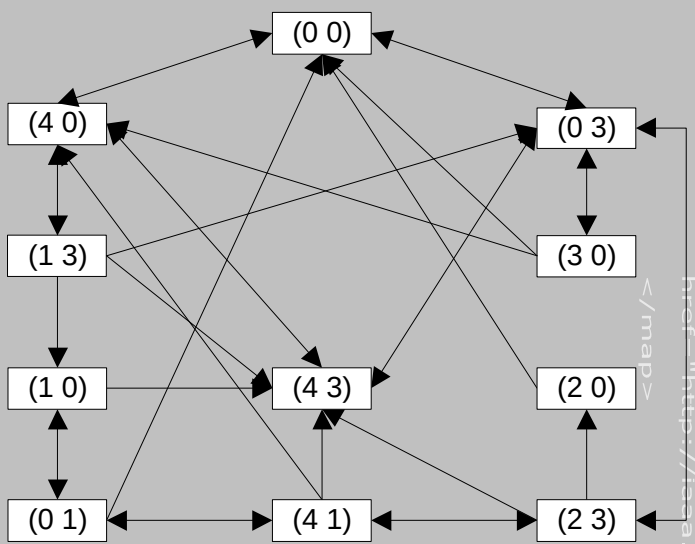
<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Busqueda primero en profundidad en el problema del 8-puzzle



Búsqueda en grafos, tratando con bucles infinitos

- Todavía NO hemos contemplado una de las complicaciones màs importantes del proceso de búsqueda:
  - ¨gastar tiempo expandiendo nodos que ya han sido encontrados y expandidos en otra rama¨
  - surgen sobretodo en problemas donde los operadores son reversibles (garrafas, granjero, misioneros, 8-puzzle, ...)



- Problemas con árboles de búsqueda infinitos
  - Hay que podar las ramas con estados repetidos
- Trata el espacio de estados como un grafo: NO explores un nodo si ya ha sido explorado
- En el caso de búsqueda en profundidad se cae en bucles con mucha más facilidad (facilmente se alcanza el límite de memoria)

Sol 1 - Búsqueda en profundidad con límite de profundidad

- ¿Como evitarlo?:
  - Poniendo un máximo de profundidad en la búsqueda
  - No se exploran nodos si el camino es mayor que esta longitud

- Poner un límite de profundidad global

```
(defparameter *limite-profundidad-por-defecto* 8)
```

```
(defun busqueda-en-profundidad-con-limite
   (estado-inicial operadores
    &optional (limite-profundidad
               *limite-profundidad-por-defecto*)
```

- La información de profundidad puede sacarse de la estructura del arbol que se ha creado
  - Contando los nodos que hay hasta el inicial
  - Poniendo los operadores usados para llegar al nodo y contarlos
  - Poniendo una propiedad en el nodo con la profundidad

Resultado en el problema del tren

- Problema
  - Problema del tren con ciudades de USA
- Solución

```
> (depth-first-search-with-duplicate-node-detection
   'sacramento.california *state-operators*)
#S(NODE STATE AUSTIN.TEXAS PATH
    (GOTO-PHOENIX.ARIZONA GOTO-DENVER.COLORADO
     GOTO-LINCOLN.NEBRASKA GOTO-DES-MOINES.IOWA
     GOTO-SPRINGFIELD.ILLINOIS GOTO-INDIANAPOLIS.INDIANA
     GOTO-FRANKFORT.KENTUCKY GOTO-COLUMBUS.OHIO
     GOTO-LANSING.MICHIGAN GOTO-MADISON.WISCONSIN
     GOTO-SAINT-PAUL.MINNESOTA GOTO-BISMARK.NORTH-DAKOTA
     GOTO-HELENA.MONTANA GOTO-BOISE.IDAHO
     GOTO-CARSON-CITY.NEVADA GOTO-SALT-LAKE-CITY.UTAH
     GOTO-SANTA-FE.NEW-MEXICO GOTO-OKLAHOMA-CITY.OKLAHOMA
     GOTO-LITTLE-ROCK.ARKANSAS GOTO-BATON-ROUGE.LOUISIANA
     GOTO-AUSTIN.TEXAS))
> *stats*
#S(SEARCH-STATISTICS
    NODES-VISITED 58
    MAXIMUM-LENGTH-OF-NODE-LIST 59
    LENGTH-OF-SOLUTION 21
    MAXIMUM-DEPTH 35
    COST-OF-SOLUTION 0)
```

- Estadísticas

```
> *stats*
#S(SEARCH-STATISTICS
     NODES-VISITED 58
     MAXIMUM-LENGTH-OF-NODE-LIST 59
     LENGTH-OF-SOLUTION 21
     MAXIMUM-DEPTH 35
     COST-OF-SOLUTION 0)
```

Resultado con límite de profundidad en el problema del tren

- Solución

```
> (depth-first-search-with-depth-limit
   'sacramento.california *state-operators*)
#S(NODE STATE AUSTIN.TEXAS PATH
    (GOTO-PHOENIX.ARIZONA GOTO-SACRAMENTO.CALIFORNIA
       GOTO-PHOENIX.ARIZONA GOTO-SANTA-FE.NEW-MEXICO
       GOTO-AUSTIN.TEXAS))
```

- Estadísticas

con límite de profundidad

```
> *stats*
#S(SEARCH-STATISTICS
     NODES-VISITED 21
     MAXIMUM-LENGTH-OF-NODE-LIST 17
     LENGTH-OF-SOLUTION 5
     MAXIMUM-DEPTH 5
     COST-OF-SOLUTION 0)
```

sin límite de profundidad

```
> *stats*
#S(SEARCH-STATISTICS
     NODES-VISITED 58
     MAXIMUM-LENGTH-OF-NODE-LIST 59
     LENGTH-OF-SOLUTION 21
     MAXIMUM-DEPTH 35
     COST-OF-SOLUTION 0)
```

Sol 1' - Búsqueda de profundización iterativa

- PROBLEMA de la búsqueda en profundidad con límite:
  - ¿Cual es el límite?
  - p.e. problema del tren para España (unas 12 ciudades)

- Profundización iterativa [Korf 87]

- 1.- Poner una longitud máxima de 1
- 2.- Intentar el algoritmo de búsqueda en profundidad con ese límite
  - Si se encuentra una solución
    - Entonces devuelve la solución
    - Sino añade 1 a la máxima profundidad
      Ir a 2

Algunos aspectos de la profundización iterativa

- Dependiendo del factor de ramificación, expande del orden del 11% (para *b=10*), 50% (para *b=2*) más nodos que bfs o dfs con límite

- ¿Es una buena idea?
  - Tiene las ventajas de consumo de espacio de la búqueda en profundidad (O(bd)) y las ventajas de la búsqueda en anchura
  - En general, resulta preferible cuando hay un gran espacio de búsqueda y no se tiene una estimación de la profundidad de la solución

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Sol 2 - Otras maneras de tratar con el problema de estados repetidos

- Tres maneras por órden incremental de eficiencia y coste computacional
  - No volver al estado del que se vino. El operador debe rechazar la generación de cualquier sucesor que sea el mismo estado que el padre del nodo.
  - No crear caminos con ciclos. El operador debe rechazar la generación de cualquier sucesor que sea el mismo estado que algún antecesor del nodo.
  - No generar ningún estado que se ha generado antes. Esto requiere guardar en memoria todos los estados que ya han sido generados antes (resulta en una complejidad de espacio $O(b^d)$)
    - se puede mantener una lista
    - los buenos algoritmos utilizan una tabla hash
- Hay que contrapesar
  - coste de almacenar y chequear
  - coste de la búsqueda extra

Algoritmo genérico de búsqueda con control de nodos duplicados

```
PRINCIPIO
1   ABIERTOS := (nodo_inicial)
1'  CERRADOS := ()
    RESUELTO := falso
2   mientras que ABIERTOS <> ()
              AND not RESUELTO hacer
3     N := quitar primer elemento de ABIERTOS
      E := estado asociado a N
4     si E es el estado solución
5       entonces RESUELTO := verdad
5'       sino añadir E a CERRADOS
          para cada operador O hacer
6           si O se puede aplicar a E
            entonces crear nodo correspondiente
              al estado E' obtenido por aplicacion
              de O a E, siempre que:
6'            - E' no aparezca en ABIERTOS
6''           - E' no aparezca en CERRADOS
7              y añadir ese nodo a ABIERTOS
    fmq
    si RESUELTO
8     entonces devuelve el estado objetivo (y si
              se requiere una explicacion, el
              camino para llegar a el)
9     sino informa de que el objetivo no puede ser
            alcanzado
  fsi
FIN
```

Control de nodos duplicados
Expansión mejorada de un nodo (6)

- ¿Y si ese nodo ya se habia visitado?
  - guardamos una lista de nodos expandidos

```
(member nuevo-estado NODOS-CERRADOS
        :test #'equal
        :key #'(lambda (y) (get y 'estado)))
```

- ¿Y si el nodo estaba en la lista de nodos a expandir

```
(member nuevo-estado NODOS-ABIERTOS
        :test #'equal
        :key #'(lambda (y) (get y 'estado)))
```

```
(and nuevo-estado
     (member nuevo-estado NODOS-CERRADOS
             :test #'equal
             :key #'(lambda (y) (get y 'estado)))
     (member nuevo-estado NODOS-ABIERTOS
             :test #'equal
             :key #'(lambda (y) (get y 'estado)))
     (list (crea-nodo nuevo-estado nodo x)))
```

Expandir un nodo sin duplicar

```
(defun expandir-nodo
       (nodo NODOS-ABIERTOS NODOS-CERRADOS)
 (let* ((estado (get nodo 'estado)))
  (format t "~%expandiendo el estado --->~s"
                 estado)
  (mapcan
   #'(lambda (x)
      (let ((nuevo-estado
                       (funcall x estado)))
       (and nuevo-estado
            (member nuevo-estado NODOS-CERRADOS
               :test #'equal
               :key #'(lambda(y)(get y 'estado)))
            (member nuevo-estado NODOS-ABIERTOS
               :test #'equal
               :key #'(lambda(y)(get y 'estado)))
            (list (crea-nodo nuevo-estado nodo x)))
       ))
   *operadores*)))
```

Haciendo programas más fáciles de corregir

- Sacar información estadística del proceso de búsqueda

```
(defstruct estadisticas-busqueda
   (nodos-chequeados 0)
   (maxima-longitud-de-la-lista-de-nodos 0)
   (longitud-de-la-solucion 0)
   (maxima-profundidad 0))
```

```
(defvar *estadisticas*
   (make-estadisticas-busqueda)
   "Informacion de las estadisticas del arbol" )
```

```
(defun actualiza-estadisticas ()
   (if (> (length *nodos-a-expandir*)
          (estad-maxima-longitud-de-la-lista-de-nodos
            *estadisticas*))
      (setf (estad-maxima-longitud-de-la-lista-de-nodos
              *estadisticas*)
            (length *nodos-a-expandir*)))
   )
```

Complejidad del algoritmo de búsqueda en profundidad

- Búsqueda primero en profundidad
  - Espacio: O(bm)
    - b = factor de ramificación
    - m = máxima profundidad del árbol
  - Tiempo: $O(b^m)$
  - SIN garantia de encontrar la solución más corta
    - Ni siquiera de encontrar una solución
- Búsqueda primero en profundidad con límite de profundidad
  - Espacio: O(bl)
    - l = profundidad del límite
  - Tiempo: $O(b^l)$
  - SIN garantia de encontrar la solución más corta
    - Ni siquiera de encontrar una solución (sólo si l >= d)
- Profundización iterativa
  - Espacio: O(bd)
    - d = profundidad de la solución
    - mucho menor que la búsqueda en anchura
  - Tiempo: $1 + (1 + b) + (1 + b + b^2) + \ldots + (1 + b + \ldots + b^{d)} = O(b^d)$
  - CON garantia de encontrar la solución más corta
    - (aunque puede tardar mucho)

Ventajas de las búsqueda en anchura y en profundidad

- Ventajas de la búsqueda primero en profundidad
  - Necesita menos memoria
    - profundidad: sólo se almacenan los nodos del camino que se sigue
    - anchura: almacena todo el árbol que se ha generado
  - ¨Con suerte¨ puede encontrar una soluciòn sin tener que examinar gran parte del espacio de estados. (Util sobretodo cuando existen varias soluciones posibles)

- Ventajas de la búsqueda primero en anchura
  - No queda atrapada explorando callejones sin salida
  - Si existe una solución garantiza que la encuentra. Si existen varias encuentra la solución mínima

Búsqueda bidireccional

- Idea: buscar simultaneamente
  - hacia delante, comenzando desde el estado inicial
    - (búsqueda dirigida por datos: data driven)
  - hacia atrás, comenzando desde el estado objetivo
    - (búsqueda dirigida por objetivos: goal driven)

- Problemas donde el factor de ramificación es similar en las dos direcciones

- Hay una solución de profundidad *d*, la solución se encontrará en $O(2b^{d/2}) = O(b^{d/2})$ (en cada sentido
sólo se avanza la mitad)
  - p.e. para b=10 y d=6
    - bfs: 1.111.111 nodos
    - bidirec.: 2.222 nodos
  - Tiempo: $O(b^{d/2})$
  - Espacio: $O(b^{d/2})$

Aspectos de la búsqueda bidireccional

- Aspectos a tener en cuenta:
  - Hay que buscar predecesores
  - A veces operadores idénticos, otras difíciles
  - Si hay varios estados objetivo posibles y sólo tenemos una descripción ...
  - Necesita modo eficiente de comprobar si un nodo ya ha aparecido en la otra mitad

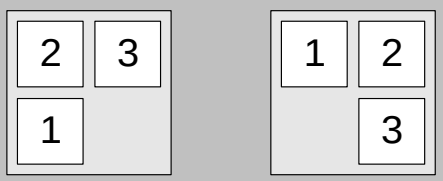
Comparación de estrategias de búsqueda ciegas

Criterio	Anchura-Primero	Profund-Primero	Profund-Limite	Profundiz-Iterativa	Bidirecc (si aplic)
Tiempo	b^d	b^m	b^l	b^d	$b^{d/2}$
Espacio	b^d	bm	bl	bd	$b^{d/2}$
¿Optimo?	Si	No	No	Si	Si
¿Completo?	Si	No	Si, si $l>=d$	Si	Si

b **: factor de ramificación**
d **: profundidad de la solución**
m **: máxima profundidad del árbol de búsqueda**
l **: límite de profundidad**

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Problema - Puzzle



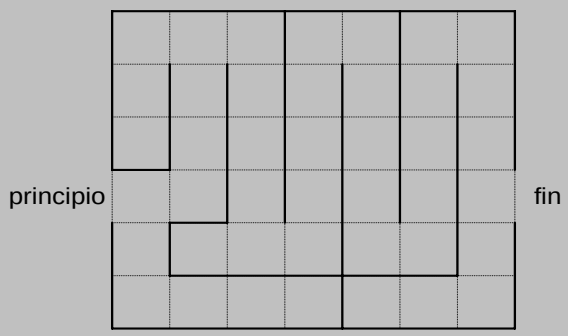
inicial objetivo

Considerar el 3-puzzle mostrado arriba. Hay 4 celdas cuadradas: tres numeradas como 1, 2, 3 y una celda en blanco. Los posibles operadores (**arriba**, **abajo**, **izquierda**, **derecha**) actuan sobre el cuadrado blanco, conmutando la posición del cuadrado blanco con el cuadrado numerado que se encuentra en esa dirección. Así, desde el estado inicial mostrado, sólo se pueden aplicar los operadores **izquierda** y **arriba**. (Notar que para cualquier estado, sólo hay dos movimientos posibles.)

Considerar los estados inicial y final mostrados, y aplicar operadores siempre siguiendo este orden fijo: **arriba**, **abajo**, **izquierda**, **derecha**. NO asumir que los estados repetidos no son visitados.

1. Dibujar el árbol de búsqueda utilizando búsqueda primero en anchura.
2. ¿La búsqueda primero en profundidad encontraría el objetivo? ¿Porqué o porque no?
3. ¿Cuantos nodos se generarían si se utilizara la estrategia de profundización iterativa con incremento en profundidad de 1?. Muestra tu respuesta como la suma de los nodos generados en cada iteración del algoritmo. (Recuerda que los operadores son aplicados en un orden fijo, por lo que esta pregunta tiene una única respuesta correcta.)

Problema - Laberinto



Una búsqueda informada puede resolver este laberinto, encontrando un camino desde la localización inicial hasta la final. Los posibles operadores son: **arriba**, **abajo**, **izquierda**, **derecha,** que son sólo válidos si hay una línea de puntos entre las celdas del laberinto. Considerar que la heurística utilizada es la distancia en línea recta entre la posición actual y la final del laberinto.

<meta name="HTTP-EQUIV" content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32" href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

BUSQUEDA HEURISTICA 1

- Objetivos:
  - Aprender sobre funciones de evaluación heurística
  - Aprender sobre técnicas de escalada
    - conseguir un objetivo

- Indice:
  - Búsqueda heurística
  - Métodos de escalada
    - Implementación del método de escalada
    - Propiedades, ventajas y desventajas

- Lecturas:
  - E. Rich y K. Knight, Cap 3.2 (excepto 3.2.3)

Origenes históricos de la búsqueda heurística

- ¨Heuristic search¨ [Newell and Ernst, 1965]

- *La tarea con la que se enfrenta un sistema de símbolos, cuando se le presenta un problema y un espacio de problema, es utilizar sus limitados recursos de procesamiento para generar posibles soluciones, una después de otra, hasta que encuentra una que satisface el test de definición del problema. ...*

  Si el sistema de símbolos tuviera algún control sobre el orden en que se generan las posibles soluciones, sería deseable disponer este orden de generación de forma que las soluciones tubieran una alta posibilidad de aparecer tempranamente. Un ***sistema de símbolos exhibiría inteligencia*** *en la medida que tuviera éxito haciendo esto. La inteligencia de un sistema con recursos de procesamiento limitados consiste en realizar elecciones acertadas sobre que hacer a continuación. ...*
  -Newell y Simon, ACM *Turing Award Lecture, 1976*

- **¨Un proceso que puede resolver un problema dado, pero no ofrece ninguna garantía de hacerlo, se llama una heurística para ese problema¨**
  -Newell, Shaw, Simon 1963

- ***El primer proyectos de sistema experto se llamó: ¨Heuristic Programming Project¨***
  [Feigenbaum, Buchanan, Lederberg]

Búsqueda heurística

- A los métodos de búsqueda primero en anchura y búsqueda primero en profundidad se les llama métodos ciegos (blind methods) o métodos débiles (weak methods) de búsqueda

- Los métodos de búsqueda heurística disponen de alguna idea sobre la proximidad de cualquier estado al estado objetivo, y puede explorar primero los caminos más prometedores

- Los métodos heurísticos
  - pueden NO encontrar siempre la mejor solución
  - pero ¨garantizan¨ encontrar una buena solución en un tiempo razonable
  - sacrificando la completitud incrementan la eficiencia
  - útiles en la resolución de problemas difíciles que
    - pueden no ser resueltos de otra forma
    - las soluciones toman un tiempo muy grande

Funciones de evaluación heurística

- Una de las estrategias de búsqueda el mejor primero consiste en minimizar el coste estimado para alcanzar el objetivo
  - normalmente este coste puede ser estimado pero no determinado exactamente
  - se expande el nodo que se cree más próximo al objetivo

- Típicamente, disponen de una función de evaluación heurística que es 0 para el estado objetivo, es más grande cuanto más lejos se encuentra del estado objetivo
  - h(n) = coste estimado del camino más barato para llegar del nodo n al objetivo

- Encontrar la función heurística adecuada es lo realmente difícil

Ejemplo de funciones de evaluación heurística

- Problema del tren:
  - Distancia del estado actual al estado objetivo

```
(defparameter *capitales*
  '(
    (Montgomery.Alabama (87 . 33))
    (Juneau.Alaska (150 . 62))
    (Phoenix.Arizona (112 . 33))
    (Little-Rock.Arkansas (92 . 35))
    (Sacramento.California (122 . 38))
    (Denver.Colorado (105 . 39))
    (Hartford.Connecticut (73 . 42)) ...))
```

  - Código para calcular la distancia entre dos ciudades dadas su latitud y longitud

```
(defun distancia-estimada-desde-objetivo
       (estado)
  (distancia-aerea estado *estado-objetivo*))
```

Algoritmo de búsqueda heurística

```
PRINCIPIO
1   ABIERTOS = (nodo_inicial)
    RESUELTO = falso
2   mientras que ABIERTOS <> ()
            AND not RESUELTO hacer
3     E = quitar primer elemento de ABIERTOS
4     si E es el estado solución
5       entonces RESUELTO = verdad
         sino
           para cada operador O hacer
6           si O se puede aplicar a E
7            entonces añadir el estado, obtenido
                  al aplicar O a E, a ABIERTOS
    si RESUELTO
8     entonces devuelve el estado objetivo y el
              camino para llegar al objetivo
9     sino devuelve que el objetivo no es posible
FIN
```

- El tipo de búsqueda depende de los pasos 7 (a veces también el 6) del algoritmo anterior

BUSQUEDAS INFORMADAS

- El algoritmo tiene cierto conocimiento del problema concreto que debe resolver => el paso 7 se realiza con criterios dependientes del dominio del problema

Métodos de escalada o ascenso de la colina (hill climbing search)

- BÚSQUEDA EN PROFUNDIDAD
  - Paso nº 6:
    - Se aplican todos los operadores posibles
  - Paso nº 7:
    - Los nodos ABIERTOS se colocan en orden inverso a su profundidad en el árbol. Los más profundos al principio, los menos profundos al final.
    - Los nodos de igual profundidad se ordenan arbitrariamente.

- Métodos de ESCALADA o ASCENSO DE LA COLINA (hill climbing)
  - El método de búsqueda en profundidad aumentado con una función heurística que mide la cercanía al objetivo
  - Evaluar un nuevo estado
    - SI es mejor que el estado actual, convertirlo en el estado actual
    - SI NO es mejor que el estado actual, continuar con el bucle

Método de escalada simple

- Método de ESCALADA SIMPLE
  - Seleccionar un operador que no haya sido aplicado con anterioridad al estado actual y aplicarlo para generar un nuevo estado
  - Evaluar el nuevo estado
    - SI es mejor que el estado actual, convertirlo en el estado actual
    - SI NO es mejor que el estado actual, continuar con el bucle

Método de escalada por la máxima pendiente

- Método de ESCALADA POR LA MÁXIMA PENDIENTE
  - (steepest-ascent hill climbing)
  - Búsqueda del gradiente (gradient search)

- Algoritmo (pasos 6 y 7)
  - Aplicar todos los operadores al estado actual
  - Evaluar los nuevos estados
    - SI algún nuevo estado es mejor que el estado actual, convertirlo en el estado actual
    - SI NO es mejor que el estado actual, falla la búsqueda

.Implementación del método de escalada por la máxima pendiente

- El usuario proporciona
  - *estado-inicial*
  - *operadores*
  - estado-solucionp
  - distancia-estimada-al-objetivo (estado)

- Estructura de datos para el nodo y su generación

```
(defun crea-nodo
      (estado padre operacion
       &optional (simbolo (gensym "NODO-")))

 (setf (get simbolo 'estado) estado)
 (setf (get simbolo 'padre) padre)
 (setf (get simbolo 'operacion) operacion)

 (setf (get simbolo
         'distancia-estimada-al-objetivo)
       (distancia-estimada-al-objetivo
         estado))
)
```

Encontrar el siguiente mejor estado

```
(defun expandir-nodo (nodo)
   ;;;para la busqueda en escalada por la maxima
   ;;;pendiente
   (let* ((estado (get nodo 'estado))
          nuevo-estado
          nuevo-nodo
          (minima-distancia (get nodo 'distancia-estimada-al-objetivo))
          mejor-nodo)
     (format t "~%expandiendo el estado --->~s" estado)

     (dolist (x *operadores* nil)
        (if (setq nuevo-estado (funcall x estado))
            (and (setq nuevo-nodo (crea-nodo nuevo-estado nodo x))
                 (setf (estad-nodos-chequeados *estadisticas*)
                       (+ 1 (estad-nodos-chequeados *estadisticas*)))
                 (< (get nuevo-nodo 'distancia-estimada-al-objetivo)
                    (get nodo 'distancia-estimada-al-objetivo))
                 (setq mejor-nodo nuevo-nodo)
                 (setq minima-distancia
                       (get nuevo-nodo 'distancia-estimada-al-objetivo)
                  ))))

     (if (or (null mejor-nodo)
             (= minima-distancia
                (get nodo 'distancia-estimada-al-objetivo)))
        nil
        (list mejor-nodo))))
```

.Usando el método de escalada por la máxima pendiente

```
(setq  *estado-inicial* 'sacramento.california)
(setq  *estado-objetivo* 'Little-Rock.Arkansas)
```

```
Resuelto ! El camino de la solucion es:
(SACRAMENTO.CALIFORNIA        0     2695)
(CARSON-CITY.NEVADA           206   2520)
(SALT-LAKE-CITY.UTAH          844   1954)
(CHEYENNE.WYOMING             1516  1320)
(PIERRE.SOUTH-DAKOTA          2045  1214)
(LINCOLN.NEBRASKA             2460  799)
(JEFFERSON-CITY.MISSOURI      3005  334)
(LITTLE-ROCK.ARKANSAS         3339  0)
HECHO

ESTADISTICAS
(NODOS-CHEQUEADOS 39)
(NODOS-CREADOS 7)
(NODOS-EXPANDIDOS 7)
(MAXIMA-LONGITUD-DE-LA-LISTA-DE-NODOS 1)
(LONGITUD-DE-LA-SOLUCION -7)
(MAXIMA-PROFUNDIDAD 7)
```

.Usando el método primero en profundidad

```
(setq  *estado-inicial* 'sacramento.california)
(setq  *estado-objetivo* 'Little-Rock.Arkansas)
```

```
Resuelto ! El camino de la solucion es:
(SACRAMENTO.CALIFORNIA   0     2695)
(PHOENIX.ARIZONA         1063  1857)
(DENVER.COLORADO         1981  1237)
(LINCOLN.NEBRASKA        2699  799)
(DES-MOINES.IOWA         2973  798)
(SPRINGFIELD.ILLINOIS          3376  584)
(INDIANAPOLIS.INDIANA          3738  695)
(COLUMBUS.OHIO                 4019  970)
(HARRISBURG.PENNSYLVANIA       4530  1435)
(ANNAPOLIS.MARYLAND            4642  1404)
(RICHMOND.VIRGINIA             4753  1382)
(RALEIGH.NORTH-CAROLINA  4993  1273)
(COLUMBIA.SOUTH-CAROLINA       5421  924)
(ATLANTA.GEORGIA         5637  771)
(MONTGOMERY.ALABAMA            5918  512)
(JACKSON.MISSISSIPPI           6198  289)
(LITTLE-ROCK.ARKANSAS          6487  0.0)
HECHO
ESTADISTICAS
(NODOS-CHEQUEADOS 77)
(NODOS-CREADOS 37)
(NODOS-EXPANDIDOS 18)
(MAXIMA-LONGITUD-DE-LA-LISTA-DE-NODOS 20)
(LONGITUD-DE-LA-SOLUCION -16)
(MAXIMA-PROFUNDIDAD 16)
(MAXIMA-PROFUNDIDAD 16)
```

Búsquedas ciegas vs heurísticas

- PROBLEMA del tren
  - Sacramento a Little Rock

- Búsqueda primero en profundidad con eliminación de nodos duplicados
  - nodos visitados: 37
  - longitud de la solución: 17
  - distancia entre ciudades para la solución: 6.487 Km
  - memoria: 20

- Escalada por la máxima pendiente
  - nodos visitados: 7
  - longitud de la solución: 7
  - distancia entre ciudades para la solución: 3.339 Km
  - memoria: 1

Propiedades del algoritmo de escalada

- Espacio: O(1)
- Tiempo: $O(b^d)$ (en el peor caso)
- NO óptimo: No garantiza que encuentra el camino
  más corto
- NO completo: puede no encontrar la solución aunque exista.
  - Máximo local: no es el objetivo pero ningún siguiente estado es mejor

```
(setq  *estado-objetivo* 'Augusta.Maine)
```

Problemas del método

- Pueden llegar a un estado desde el que todos los operadores conducen a estados peores
  - MAXIMO LOCAL
    Lo pero es que suele aparecer en las cercanías de la solución
  - MESETA
    Area plana en la que no es posible encontrar la mejor dirección para moverse
  - CRESTA

- SOLUCIONES
  - Volver hacia atrás e intentar un camino diferente (bueno para máximos locales)
  - Realizar un gran salto en alguna dirección para intentar otro camino (bueno para mesetas)
  - Aplicar dos o más operadores antes de realizar la evaluación (bueno para crestas)

Ventajas y desventajas

- MOTIVOS
  - La escalada es un método local (visión corta)
  - Ventaja: limita sensiblemente el espacio de búsqueda
  - Desventaja: sin garantias de resultar eficaz

- OTRA ALTERNATIVA
  - Enfriamiento simulado (se permiten pasos hacia atrás)

Resolviendo algunos problemas del método de escalada

- No es completo
  No garantiza que encuentra una solución aunque ésta exista
  - Mantener una lista de estados ordenados por la distancia estimada al objetivo *(búqueda del mejor primero)*
  - Si no se encuentra uno mejor que el estado actua, tratar con una búqueda primero en profundidad profundizando iterativamente desde el estado corriente hasta que se encuentre un estado mejor que el actual

- No es óptimo
  No encuentra la solución más corta
  - Mantener una lista de estados ordenados por la distancia estimada al objetivo, más una estimación del coste de llegar al objetivo *(A*)*. Optimo sólo si no se sobreestima la distancia al objetivo

BUSQUEDA HEURISTICA 2

- Objetivos:
  - Aprender sobre la búsqueda mejor primero y A*
  - Comparar varios algoritmos de búsqueda
    - complejidad
    - completitud (habilidad para encontrar una solución, si existe)
    - optimalidad (habilidad para encontrar la solución óptima)
  - Funciones heurísticas
    - f'(estado) = g(estado) + h'(estado)
  - Discutir la implementación en Lisp de la búqueda el mejor primero
- Indice:
  - Método primero el mejor
    - propiedades, implementación del 8-puzzle, sus heurísticas
    - búsqueda en rayo
  - Búsqueda heurística óptima
    - Algoritmos A y A*, comparaciones
- Lecturas:
  - Rich&Knight, Cap 3.3 (excepto 3.3.3) y 12.5
  - Otras: N.J.Nilsson (Principios de IA. Diaz de Santos), Cap. 2

Algunas definiciones

- Coste de un camino = suma de los costes de sus arcos
- Longitud del camino = Nº de arcos = **d(n)**
- **Si** coste de cada arco = 1
  Entonces coste de un camino = su longitud

- Los métodos de búsqueda heurística tienen una función que determina la calidad de cualquier estado del espacio de búsqueda
  - Funciones para evaluar la calidad de un estado
    - **h(n)** : coste del camino óptimo de n al objetivo
    - **g(n)** : coste del camino óptimo del estado inicial a n.Es conocido pues se conoce ese camino.
    - **f(n) = h(n) + g(n)** : coste de un camino óptimo que pase por el nodo n
    - **f'(n)** es una estimación de **f(n)** : una estimación del coste del camino óptimo que pasa por n
    - **h'(n)** es una estimación de **h(n)** : una estimación de la distancia al objetivo. Es la <u>información heurística del problema</u>
  - p.e. si **h(n)=0** y **g(n)=d(n)** entonces es la búsqueda en anchura

Perspectiva de los métodos de búsqueda

- Las búsquedas primero en anchura y primero en profundidad son métodos de búsqueda ciegos (o ininformados)
- Búsqueda por escalada
  - h'(estado) = una estimación de la distancia al objetivo
  - h(estado) la verdadera distancia al objetivo no es conocida
- Búsqueda primero el mejor
  - Como la búsqueda por escalada, pero mantiene una lista de estados a ser expandidos, ordenados por h'(estado)
- Búsqueda primero el más corto
  - Como la búsqueda por escalada, pero mantiene una lista de estados a ser expandidos, ordenados por g(estado)
- Búsqueda A*
  - Como la búsqueda por escalada, pero mantiene una lista de estados a ser expandidos, ordenados por f'
  - f'(estado) = g(estado) + h'(estado)
  - Una estimación de la calidad de un estado es el coste total de llegar a ese estado desde el estado inicial g(estado) mas una estimación de la distancia al objetivo h'(estado)

Problemas con el método de escalada

- Máximo local: No se ha llegado al objetivo pero
  - todos los sucesores son peores que el estado actual
  - muchos sucesores tienen el mismo valor, de forma que no se puede elegir razonablemente entre ellos

- El método de escalada usa poca memoria (1), pero no puede explorar alternativas si el estado actual es un camino muerto

- Búsqueda primero el mejor: mantiene una lista de alternativas para el caso en que en un momento determinado se piensa que un camino es el mejor y luego no funciona

- Es una combinación de las ventajas de
  - búsqueda en profundidad (puede encontrar una solución sin expandir todos los nodos)
  - búsqueda en anchura (no queda atrapada en caminos sin salida)

  - avanza en profundidad y puede volver hacia atrás (cosa que no puede hacer el de escalada)

Método primero el mejor

- ALGORITMO
  - ABIERTOS: cola de prioridad de nodos que han sido evaluados por la función heurística pero no han sido expandidos
  - CERRADOS: estados (no nodos) ya expandidos pero que se guardan para hacer una búsqueda en árbol, no en grafo

- Paso nº 7
  - ABIERTOS se ordena por **h'** (la distancia estimada al objetivo)

Propiedades del método primero el mejor

- Espacio: $O(b^d)$ (en el peor caso)
- Tiempo: $O(b^d)$ (en el peor caso)
- NO óptimo: No garantiza que encuentra el camino
  más corto
- Completo: encuentra una solución (si existe)
- Es un algoritmo general para cualquier espacio de estados
- Es igualmente aplicable a búsquedas
  - dirigidas por datos
  - dirigidas por objetivos

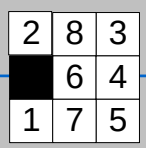
. Estados para el problema del 8-puzzle

- Representación del estado

```
(setq estado
  (make-array '(3 3)
              :initial-contents '((1   2   3)
                                  (8 space 4)
                                  (7   6   5))))
(setf (aref estado 1 2) 'space)
(aref estado 1 2)
> space
```

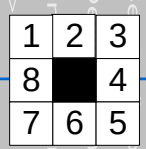
  - se puede elegir aleatoriamente (más adelante)

- Estado inicial




```
(defparameter *estado-inicial*
  (make-array '(3 3)
              :initial-contents '((  2   8 3)
                                  (space 6 4)
                                  (  1   7 5))))
```

- Estado objetivo




```
(defparameter *estado-objetivo*
  (make-array '(3 3)
              :initial-contents '((1   2   3)
                                  (8 space 4)
                                  (7   6   5))))

(defun estado-objetivop (estado)
  (equal estado *estado-objetivo*))
```

. Funciones para manipular el tablero en el problema del 8-puzzle

- Para generar un nuevo estado se copia el tablero del anterior y se hace el cambio

```
(defun copia-tablero(tablero)
  (let ((new-tablero(make-array '(3 3))))
    (loop for i from 0 to 2
        do (loop for j from 0 to 2
                 do (setf (aref new-tablero i j)
                          (aref tablero i j) )))
    new-tablero))
```

- Para encontrar el emplazamiento de una pieza

```
(defun encuentra-pieza(x tablero)
"devuelve una lista con las coordenadas x y de
 la pieza x en el tablero"
  (loop for i from 0 to 2
      thereis (loop for j from 0 to 2
            thereis
              (when (eq (aref tablero i j) x)
                    (list i j)))))
```

- Para imprimir un estado

```
(defun imprime-tablero(tablero)
  (format t "~%-------")
  (loop for i from 0 to 2
     do (format t "~%|")
     (loop for j from 0 to 2
        do (format t "~A|"
             (if (eq (aref tablero i j) 'space)
                 " "
                 (aref tablero i j))))
        (format t "~%-------")))
```

. Operadores para el problema del 8-puzzle

- Conjunto de operadores posibles

```
(setq *operadores*
   '(move-up
     move-down
     move-left
     move-right))
```

- Implementación de los operadores

```
(defun move-up(state)
  (let* ((at-space (find-square 'espacio state))
         (i (first at-space))
         (j (second at-space))
         (new-state (copia-tablero state)))
    (when (> i 0)
      (setf (aref new-state i j)
            (aref new-state (- i 1) j))
      (setf (aref new-state (- i 1) j)
            'espacio)
      new-state)))
```

. Elegir una situación inicial aleatoriamente

- Para hacer un movimiento aleatorio

```
(defun movimiento-aleatorio(state)
"Coge aleatoriamente uno de los 4 operadores.
 Si el operador no es aplicable, elige otra vez"
  (let ((r (random 4)))
    (or (cond ((= r 0)(move-left state))
        ((= r 1) (move-right state))
        ((= r 2) (move-up state))
        ((= r 3) (move-down state)))
      (movimiento-aleatorio state))))



(defun movimientos-aleatorios (n state)
  "hace n movimientos aleatorios"
  (loop for i from 1 to n
        do (setq state
                 (movimiento-aleatorio state)))
  state)



(defparameter *estado-inicial*
  (movimientos-aleatorios 20 *estado-objetivo*))
```

Funciónes de evaluación heurística para el 8-puzzle

- ¿Como podriamos estimar el coste hasta conseguir el objetivo?

a

- Una buena heurística es la distancia Manhattan:

¨suma de las diferencias en las coordenadas x e y de cada pieza entre el tablero actual y el tablero objetivo¨

- b ras heurísticas:
  - Piezas fuera de sitio
    - Muy simple
    - NO usa toda la información disponible (esfuerzo para llevar la pieza a su lugar)

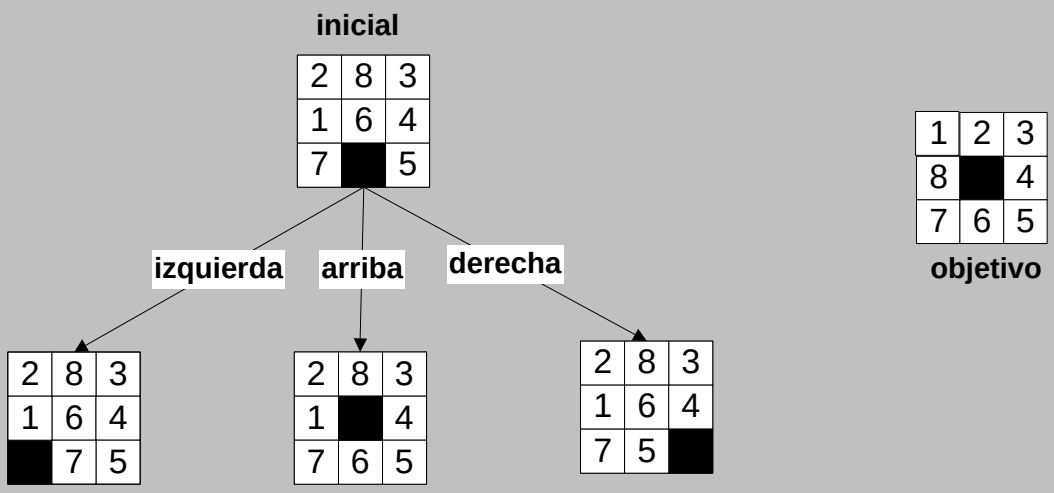
c

  - La distancia Manhattan no tiene en cuenta el esfuerzo adicional si dos piezas estan en posiciones contrapuestas
    - puede ser necesario más de 2 movimientos para ponerlos en su lugar

d

- Distancia Manhattan + 2*(piezas contrapuestas)

Ejemplos de las heurísticas



2 8 3 1 6 4 ■ 7 5	6	5	0	6
2 8 3 1 ■ 4 7 6 5	4	3	0	4
2 8 3 1 6 4 7 5 ■	6	5	0	6
	h_a	h_b	h_c	h_d

. Implementación de heurísticas para el 8-puzzle

- ¿Como podriamos estimar el coste hasta conseguir el objetivo?

- Una buena heurística es la distancia Manhattan:

¨suma de las diferencias en las coordenadas x e y de cada pieza entre el tablero actual y el tablero objetivo¨

```
(defun distancia-estimada-al-objetivo (tablero)
"Computa la distancia Manhattan para cada pieza
 (excepto el espacio)"
  (loop for i from 1 to 8
        summing
        (distancia-manhattan
           (find-square i tablero)
           (find-square i *estado-objetivo*))))


(defun distancia-manhattan (p1 p2)
"dadas dos listas de coordinadas x y, suma la
 diferencia entre xs e ys"
  (+ (abs (- (first p1) (first p2)))
     (abs (- (second p1) (second p2)))))
```

. Aplicando funciones en el problema del 8-puzzle

```
> (setq *estado-inicial*
    (movimientos-aleatorios 30
       *estado-objetivo*))
#2A((1 5 2) (7 SPACE 3) (8 4 6))
> (escribe-tablero *estado-inicial*)
-------
|1|5|2|
-------
|7| |3|
-------
|8|4|6|
-------
NIL
> (escribe-tablero (move-up *start-state*))
-------
|1| |2|
-------
|7|5|3|
-------
|8|4|6|
-------
NIL
> (escribe-tablero *goal-state*)
-------
|1|2|3|
-------
|4|5|6|
-------
|7|8| |
-------
NIL
> (distancia-estimada-al-objetivo
    *estado-inicial*)
8
```

. Reorganización de los nodos (7)

- Reorganización de los nodos a expandir

```
(defun reorganizar-nodos-a-expandir (nodos)
  (and nodos
    (setq ABIERTOS
          (sort (append nodos ABIERTOS)
                #'<
                :key #'(lambda (nodo)
(get nodo 'distancia-estimada-al-objetivo))
))))
```

Búsqueda en rayo (beam search)

- Una variación de la búsqueda primero el mejor
- Solo se consideran los n estados más prometedores para futuras consideraciones
  - Paso nº 7
    - ABIERTOS se ordena por la distancia estimada al objetivo
    - Se quedan en ABIERTOS los n primeros nodos
- Ejemplo: sistema ISIS/OPIS n=9
- Ventajas: reduce drásticamente el espacio de búsqueda
- Desventajas: peligro de no encontrar la mejor solución (no óptimo) e incluso de ser incompleto

Búsqueda heurística óptima

BÚSQUEDA PRIMERO EL MÁS CORTO

- Como la búsqueda primero el mejor, pero ordena los nodos por el coste del camino desde el estado inicial al estado actual: g(estado)
- Garantiza encontrar el camino más corto pero tan caro en memoria y espacio como la búsqueda primero en anchura.
  - Si el coste es el mismo para todos los operadores: constituye una nueva versión del algoritmo primero en anchura (conseguia el camino óptimo en cantidad de operadores)

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Introduction">
<meta name="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

.Más información en el nodo

```
(defun crea-nodo
      (estado padre operacion
       &optional (simbolo (gensym "NODO-")))

 (setf (get simbolo 'estado) estado)
 (setf (get simbolo 'padre) padre)
 (setf (get simbolo 'operacion) operacion)
 (setf (get simbolo
         'distancia-estimada-al-objetivo)
       (distancia-estimada-al-objetivo
         estado))

 (setf (get simbolo
         'coste-del-camino-hasta-ahora)
       (if padre
          (+ (coste-de-aplicar-operador
              padre operacion)
             (get padre
              'coste-del-camino-hasta-ahora))
          0))
)
```

para el 8-puzzle: 1

para el tren: distancia real entre las dos ciudades

. Reorganización de los nodos (7)

- Reorganización de los nodos a expandir

```
(defun reorganizar-nodos-a-expandir (nodos)
  (and nodos
    (setq ABIERTOS
          (sort (append nodos ABIERTOS)
                #'<
                :key #'(lambda (nodo)
(get nodo 'coste-del-camino-hasta-ahora))
))))
```

- Problema del tren

```
(defun coste-de-aplicar-operador
        (estado operacion)
   (distancia-aerea estado
                    (operacion estado)))
```

- 8-puzzle

```
(defun coste-de-aplicar-operador
        (estado operacion)
   1)
```

Funciones heurísticas más completas

- La búsqueda mejor primero ordena estimando la distancia al objetivo h'.
- La búsqueda primero el más corto ordena por la distancia real desde el estado inicial: g.
- A ordena por $f'=g+h'$
  - $(g>=0 \;,,\; h'>=0)$
  - estimación del coste total
- [Hart, Nilsson and Raphael 1968], [Hart et al 1972]:
  Algoritmo A es el algoritmo primero el mejor cuando utiliza f' como función heurística

Algoritmo A

- ALGORITMO
  - Se necesitan (ahora obligatoriamente) dos listas de nodos
    - ABIERTOS: cola de prioridad de nodos que han sido evaluados por la función heurística pero no han sido expandidos
    - CERRADOS: estados (no nodos) ya expandidos pero que se guardan para hacer una búsqueda en árbol, no en grafo

  - Paso nº 6 (generar un estado sucesor y evaluar el camino):
    - 1.- Si el estado NO se ha generado con anterioridad, añadirlo a ABIERTOS
    - 2.- Si el estado SI se ha generado con anterioridad y este nuevo camino es mejor que el anterior, cambiar el padre del anterior y actualizar el coste empleado para alcanzar el anterior y a los sucesores que pudiera tener

  - Paso nº 7
    - ABIERTOS se ordena por **f'=g+h'** (una estimación del coste total)

Algoritmo A*

- Si se hace una restricción a h' se puede conseguir se puede probar que el es completo y óptimo
  - h' nunca sobreestima el coste de alcanzar el objetivo

    heurística admisible
    - optimistas (se creen que el coste de alcanzar el objetivo es menor del real)

- El algoritmo A se llama algoritmo A* cuando h' es minorante de h (no sobrestima a h).
  - **h'(n)<= h(n)** para todo n

MUY IMPORTANTE

- Ejemplo de heurística admisible
  - la distancia aérea entre dos ciudades es menor que la distancia en tren

Características del A*

- Complejidad del A* (peor caso- p.e., h=0)
  - espacio: $O(b^d)$
  - tiempo: $O(b^d)$
    - Para la mayoría de los problemas es exponencial
      - a no ser que
        $|h(n) - h^*(n)| <= O(\log h^*(n))$
      - Normalmente el error es proporcional al coste del camino lo que provoca crecimiento exponencial
    - Normalmente el programa cae por espacio (A* obliga a mantener todos los nodos en memoria) antes que por tiempo

Admisibilidad del A* (optimalidad)

- Propiedades de TODOS los algoritmos A*

  - son admisibles
    - Un algoritmo de búsqueda es admisible si, para cualquier grafo, encuentra el camino óptimo siempre que este exista
  - son óptimos
  - exhiben monotonicidad
    - para todos los caminos que salen del inicial, la función de coste nunca decrece
  - son completos
    - para grafos localmente finitos (con factor de ramificación finito)

Propagación hacia atrás de costes durante la búsqueda

- A* es un verdadero engorro para implementar. Cada vez que se encuentra un camino más corto entre el nodo de comienzo y un nodo, A* debe actualizar el coste del camino que va a través de ese nodo

- Se necesita para eso:
  - Información de los sucesores de un nodo
  - Información sobre el coste total del nodo

.Propagación hacia atrás

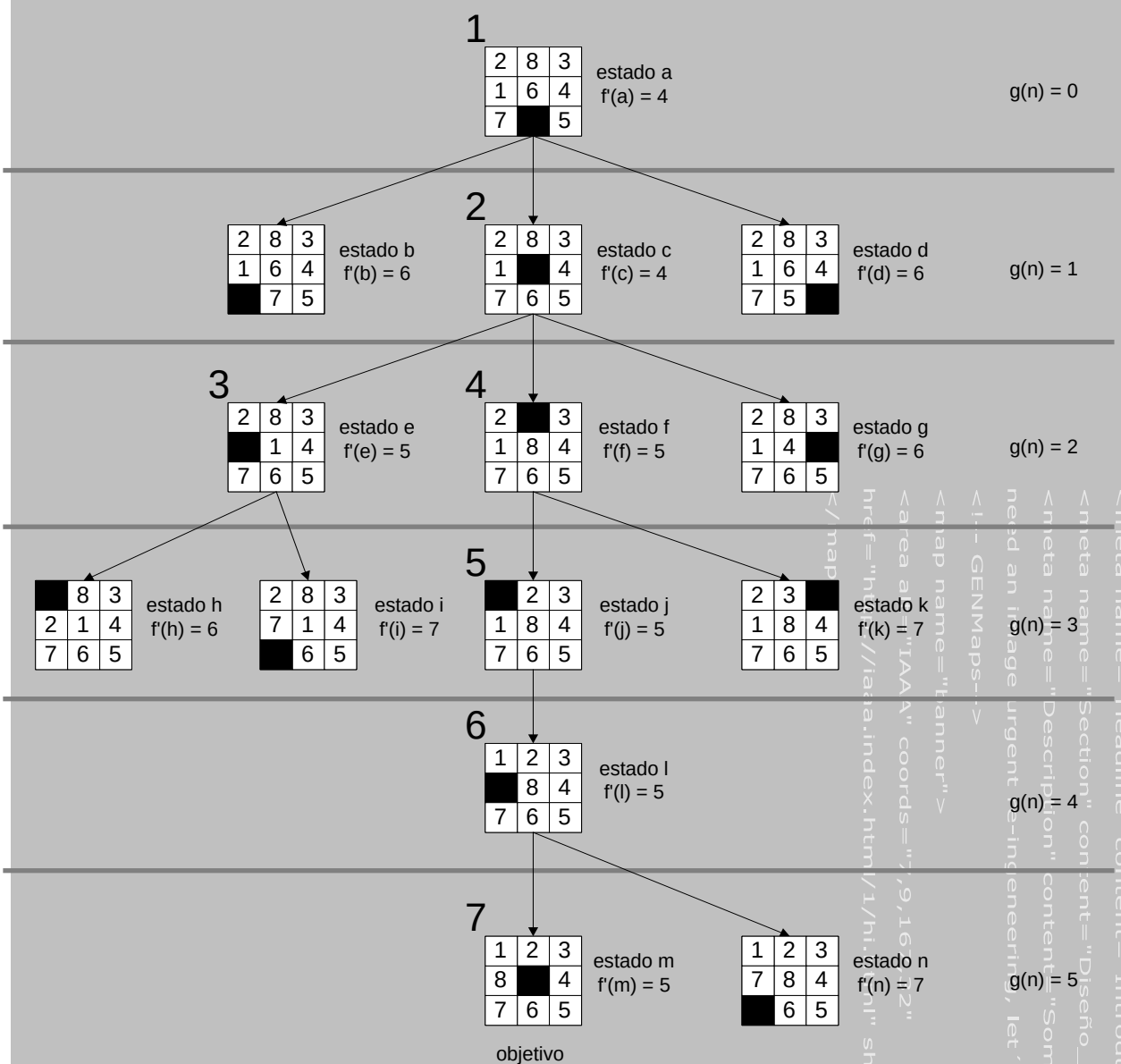
```
(defun crea-nodo
      (estado padre operacion
       &optional (simbolo (gensym "NODO-")))

  (setf (get simbolo 'estado) estado)
  (setf (get simbolo 'padre) padre)
  (setf (get simbolo 'sucesores) nil)
  (setf (get simbolo 'operacion) operacion)
  (setf (get simbolo
          'distancia-estimada-al-objetivo)
        ...)
  (setf (get simbolo
          'coste-del-camino-hasta-ahora)
        ...)
  (setf (get simbolo
          'coste-total-estimado)  +   )
)
```

- Posibilidades
  - hay otro nodo con el mismo estado
  - está en ABIERTOS o en CERRADOS
  - tiene el <, =, > coste

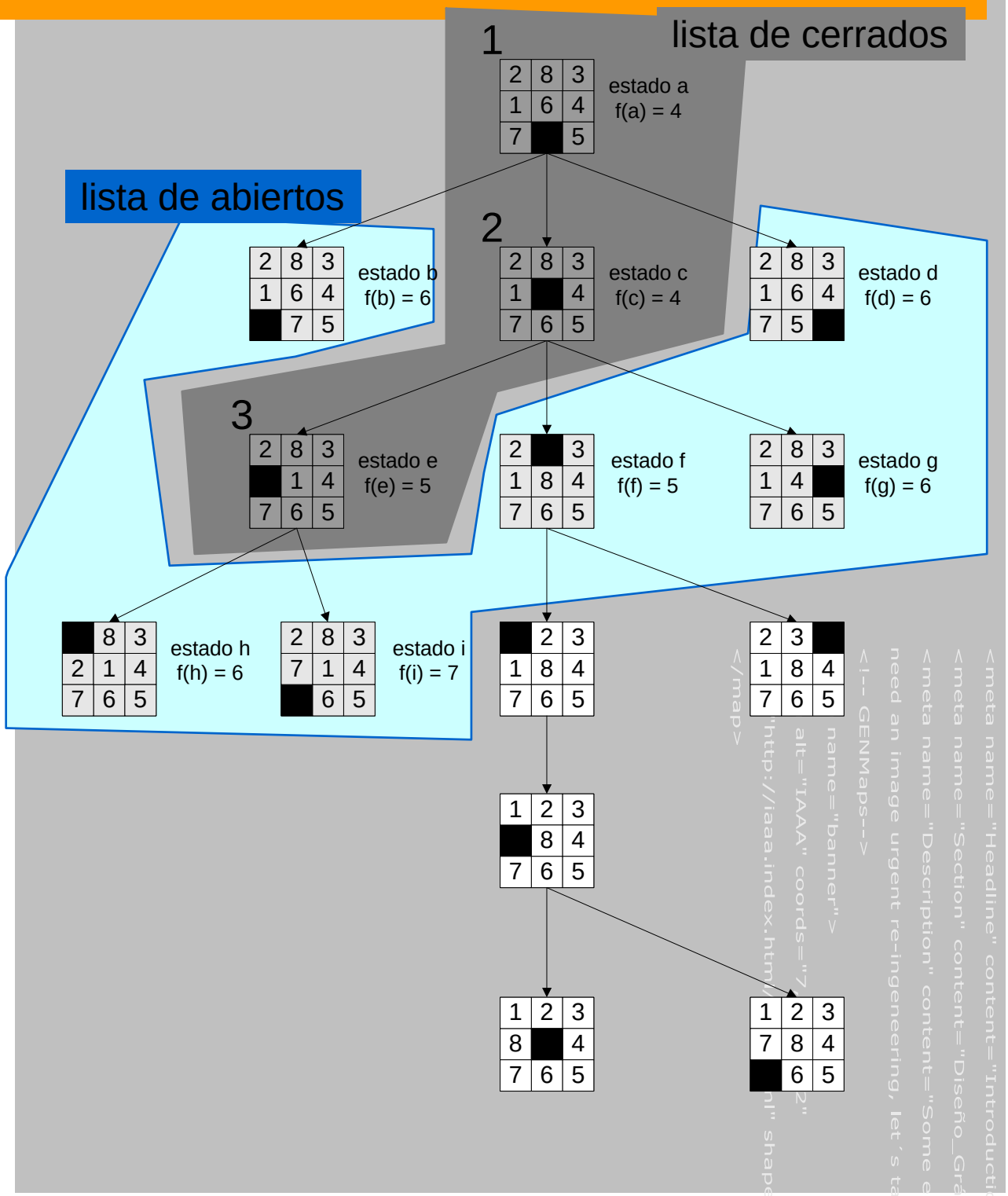
Espacio de estados del A* en el problema del 8-puzzle

f'(n) = g(n) + h'(n)
g(n) = distancia real del estado inicial a n
h'(n) = número de piezas fuera de lugar

nivel de búsqueda g(n) =



Listas de abiertos y cerrados en el proceso de búsqueda



¿Cuando una heurística es mejor?

- Dadas dos heurísticas A* h_1 y h_2,
  - se dice que h_2 está ***mejor informada*** que h_1
  - si para cualquier estado *n* del espacio de búsqueda
  - $h_1(n) <= h_2(n)$

- Comparaciones en el 8-puzzle
  - La búsqueda en anchura es el peor A* ($h'(n) = 0$)
  - La herurística suma de piezas fuera de lugar es un A*
  - La heurística con la distancia Manhattan es A* y está mejor informada que la anterior

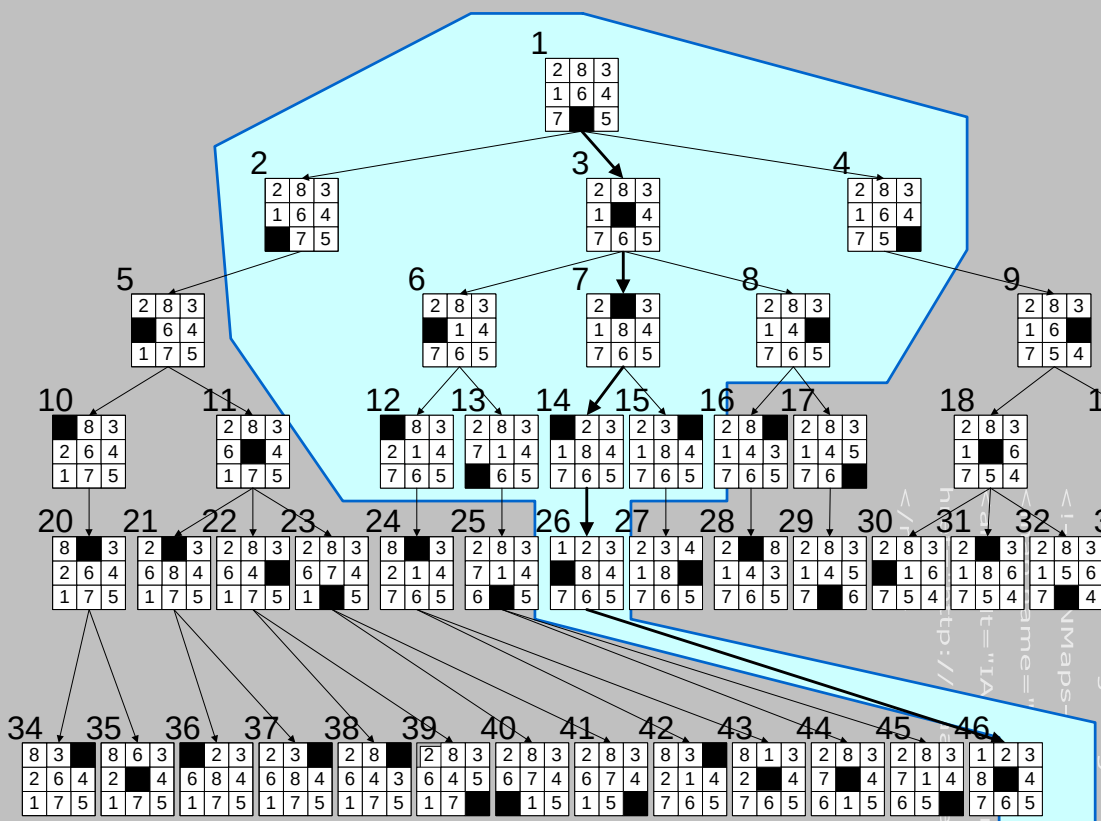
- Potncia heurística:
  - Promedio entre el coste dl camino solución encontrado y el coste de la búsqueda realizada
  - Es lo mismo que: relación CALIDAD-PRECIO
  - En problemas donde no se requiera la solución óptima se pueden utilizar heurísticas no admisibles

. Experimentando con el problema del 8-puzzle

```
> (setq *estado-inicial*
       (movimientos-aleatorios 30
                *estado-objetivo*))
#2A((1 5 2) (7 SPACE 3) (8 4 6))
> (distancia-estimada-al-objetivo
    *estado-inicial*)
8

>(steepest-ascent-hill-climbing-search
      *estado-inicial* *eight-puzzle-
operators*)
NIL

> (best-first-graph-search *estado-inicial*
                  *eight-puzzle-operators*)
#S(HNODE STATE #2A((1 2 3) (4 5 6) (7 8 SPACE))
    PATH (MOVE-UP MOVE-RIGHT MOVE-DOWN
          MOVE-DOWN MOVE-LEFT MOVE-LEFT
          MOVE-UP MOVE-RIGHT ...)
    ESTIMATED-DISTANCE-FROM-GOAL 0
    COST-OF-PLAN-SO-FAR 38)


> (a-star *estado-inicial*
        *eight-puzzle-operators*)
<Node :State #2A((1 2 3) (4 5 6) (7 8 SPACE))
 :path (MOVE-DOWN MOVE-LEFT MOVE-UP MOVE-RIGHT
        MOVE-UP MOVE-RIGHT MOVE-DOWN MOVE-
DOWN)
 :cost-of-plan-so-far  8.0
 :estimated-total-cost  8.0
 :depth 9>
>
```

Comparación del espacio de estados en el problema del 8-puzzle

- Comparación del espacio de estados entre los algoritmos A* y primero en anchura
  - Heurística utilizada: distancia Manhattan



Información heurística vs eficiencia

- Más información heurística en A*
  - menos nodos es necesario expandir
    - menos coste computacional
  - normalmente mayor esfuerzo para aplicar la heurística

- El ajedrez es un ejemplo típico
  - Una escuela usa heurísticas simples y dedica el esfuerzo a la búsqueda
    - usan hardware especializado para aumentar la profundidad
  - Otra escuela usa heurísticas sofisticadas para reducir el espacio de búsqueda (pero que a su vez pueden involucrar complejidad exponencial)
    - cálculos de las ventajas de las piezas
    - control de la geografía del tablero
    - estrategias de ataque posibles
    - estrategias defensivas ...
  - Tiempo limitado > coste computacional limitado
  - Que es lo mejor es todavía una cuestión abierta

Medidas de rendimiento

- N : número total de nodos expandidos por A* para un problema dado
- d : profundidad de la solución

- P : penetración
  - $P = d/N$

- b^* : factor de ramificación efectivo

es el factor de ramificación que un árbol uniforme de profundidad d debería tener para contener N nodos,

- $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

- *Ejemplo:*
  - Si $d = 5$ y $N = 52$ entonces $b^* = 1.91$

- es una medida más independiente de la longitud del camino óptimo que P
- b^* suele ser bastante uniforme
- La heurística es mejor cuanto más se acerca a *1*

Comparación de rendimiento en el problema del 8-puzzle

- Comparación de costes de búsqueda y factores de ramificación efectiva para los algoritmos
  - Profundización iterativa
  - A* con heurística a (número de piezas en posición erronea)
  - A* con heurística b (distancia Manhattan)

	Coste de la búsqueda			Factor ramific. efectivo		
d	IDS	A*(h_a)	A*(h_b)	IDS	A*(h_a)	A*(h_b)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

- Son datos medios sobre 100 ejemplos del 8-puzzle, para varias longitudes de solución

Usos prácticos de los algoritmos de búsqueda

- Aquí hay algunos otros punteros sobre aplicaciones de los algoritmos de búsqueda
  - Representación del conocimiento
    - Mejor primero, A*, satisfación de restricciones y análisis de medios-fines
  - Razonamiento con incertidumbre
    - En profundidad, en anchura, satisfación de restricciones
  - Razonamiento distribuido
    - A* y satisfación de restricciones
  - Comprensión del lenguaje
    - satisfación de restricciones
  - Planificación
    - A*, AO*, satisfación de restricciones y análisis de medios-fines
  - Aprendizaje
    - satisfación de restricciones y análisis de medios-fines
  - Visión
    - en profundidad, en anchura, heurísticas, enfriamiento simulado, satisfación de restricciones
  - Robótica
    - satisfación de restricciones y análisis de medios-fines

Usos prácticos de los algoritmos de búsqueda 2

- A*
- Búsqueda en profundidad
  - Prueba de teoremas lógicos
  - Procesamiento del lenguaje natural
- Escalada
  - Utilizado normalmente para minimizar alguna función objetivo cuando el objetivo no es claramente conocido
    - Aprendizaje
    - .....

<meta name="HTTP-EQUIV" content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32" href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Trabajo para casa 4 (parte 1)

1. (50). Resuelve tu problema de las torres de Hanoi con búsqueda primero en anchura, búsqueda primero en profundidad (si es posible), búsqueda primero en profundidad con límite (de 10) (ambas con y sin detección de nodos duplicados). Registra la cantidad de tiempo (nodos visitados) y espacio (máxima longitud de la lista de nodos) para cada algoritmo.

Escribe cuales son las ventajas y desventajas de cada una en este problema y en general. Tu discusión se debe focalizar en

- Completitud (¿Puede encontrar siempre una solución si existe?)
- Optimalidad (¿Puede encontrar siempre la solución más corta?)
- Complejidad en espacio (Número de nodos visitados)
- Complejidad en timepo (Máxima longitud de la lista de nodos)

El código para todas las funciones de búsqueda puedes encontrarlo en el fichero blind-search.lsp del directorio uci171pc.

2. (50) Implementa la búsqueda de profundización iterativa y pruebala en el problema de las torres de Hanoi. Si tu solución ocupa más de 5 líneas, lo estas haciendo demasiado complicado.

Trabajo para casa 4 (parte 2)

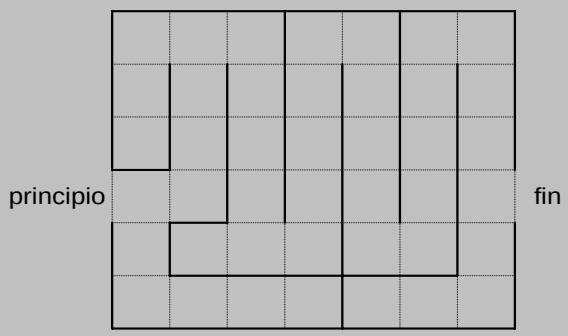
3. (150). Implementa la estrategia de búsqueda bidireccional. Problemas potenciales

- No todos los problemas tienen un único estado objetivo
- No todos los operadores tienen inversa (p.e. no siempre puedes coger tres litros de agua de una garrafa de 4 litros)

Implementalo para cada uno de los siguientes problemas, indica si es aplicable, y si lo es compara sus complejidades en tiempo y espacio con la búsqueda en anchura.

- problema del tren fichero: train-problem.lisp
- problema del granjero fichero: farmer-wolf-goat-cabbage.lisp
- problema de las garrafas fichero: jug-problem.lisp
- problema del 8-puzzle fichero: 8puzzle.lisp
- y el problema de las torres de Hanoi

Problema - Laberinto



Una búsqueda informada puede resolver este laberinto, encontrando un camino desde la localización inicial hasta la final. Los posibles operadores son: **arriba**, **abajo**, **izquierda**, **derecha,** que son sólo válidos si hay una línea de puntos entre las celdas del laberinto. Considerar que la heurística utilizada es la distancia en línea recta entre la posición actual y la final del laberinto.

1. Muestra que la búsqueda en escalada (ascenso de la colina) no podrá encontrar una solución óptima.

2. ¿Que algoritmo de búsqueda garantizaría encontrar una solución óptima?

3. Indica los seis primeros cuadrados buscados en el laberinto utilizando A*, con la heurística dada, y un coste unitario por cada movimiento.

ESTRATEGIAS DE BUSQUEDA EN JUEGOS CON ADVERSARIO

- Objetivos:
  - Aprender sobre juegos con dos jugadores
  - Aprender sobre funciones de evaluación de juegos
  - Aprender sobre la búqueda minimax
  - Aprender sobre alfa beta, una heurística de búsqueda admisible para minimax
  - Aprender a utilizar una implementación en Lisp de búsqueda minimax con alfa beta
  - Aprender a identificar el mejor movimiento de un árbol de juego y nodos podados por cortes de alfa-beta

- Indice:
  - 1.- Introducción al problema
  - 2.- Minimax
    - Limitación de la profundidad de busqueda
  - 3.- Poda Alfa-Beta

- Lecturas:
  - E. Rich y K. Knight, Cap 12.1, 12.2, 12.3, 12.6
  - G. Luger y W. Stubblefield, Cap. 4.3
  - N. Nilsson, Cap. 3.4

1.- Contexto de juego

- Juegos que se consideran:
  - Juegos con dos jugadores {computador, hombre}, que juegan alternativamente.
  - Los jugadores mueven por turno y los dos quieren ganar.
    - El oponente introduce incertidumbre {no sabemos que va a hacer}.
  - Uno de los jugadores gana (y el otro pierde) o el resultado queda en empate (tablas).
  - Perfectamente informados:
    - NO influye la suerte. NO se consideran juegos en los que el resultado está determinado parcialmente por el azar (aunque el estudio podría generalizarse a alguno de estos juegos).
      - ejemplos: dados, muchos juegos de cartas
    - Las reglas son conocidas, bien definidas y limitadas.
    - Cada jugador conoce perfectamente la evoluci'on pasada del juego y lo que puede hacer 'el y su oponente.
  - Ejemplos:
    - damas, ajedrez, tic-tac-toe, go, nim
- Objetivo:
  - Determinar la mejor jugada para que se maximicen las posibilidades de ganar.

Los juegos como problemas de búsqueda

- Un juego puede definirse como un problema de búsqueda con los siguientes componentes:

Estado inicial
Posición inicial del tablero
Test terminal
Determina si el juego ha terminado
Conjunto de operadores
Movimientos legales que puede hacer un jugador

MIN
Jugador 2 Queremos que pierda
MAX
Jugador 1 Mueve primero Queremos que gane

Función de utilidad
Da un valor numérico para la finalización del juego (depende del juego) p.e. en el ajedrez: +1 gana MAX -1 pierde MAX 0 empatan

- La estrategia a seguir depende de lo que quiera hacer MIN
  - estrategia que conduzca a un estado terminal ganador, independientemente de lo que haga MIN

Arbol (parcial) de búsqueda del 3-en-raya (Tic-Tac-Toe)

Problema de utilizar la búsqueda

- PROBLEMA: En la mayoría de los juegos la búsqueda es impracticable
  - ejemplo del **ajedrez**:
    - factor de ramificaciòn medio: 35
    - media de 50 movimientos por jugador
    - complejidad
      - NO por falta de informaciòn
      - SINO por no ser posible calcular las consecuencias exactas de un movimiento

35^{100} nodos
sólo 10^{40} posiciones legales

  - ejemplo **damas**: arbol completo 10^{40} nodos

si 1 nodo en 1/3 de nanosegundo
10^{21} siglos

  - ejemplo **Tic-Tac-Toe**:
    - 9! = 362.880 (computador juega 1º)
    - 8! = 40.320 (computador juega 2º)
  - Normalmente tienen límites de tiempo por lo que se penaliza severamente la ineficiencia

¿Que es lo que se intenta?

- Esto implica que hay que modificar
  - Utilizar
    - una **función de evaluación heurística** (estimación de la utilidad esperada del juego para una posición dada),
    - en lugar de la **función de utilidad** (evalua nodos terminales)
  - Las condiciones de terminación:
    - basadas en tiempo limitado
    - espacio de almacenamiento limitado
    - máxima profundidad de los nodos

- Una buena estratégia puede ser hacer una *profundización iterativa* tratando de mirar más hacia delante si el tiempo lo permite

Ejemplo de movimiento (p)

- Se podría elegir el movimiento que maximize la situación:
  - NO se consideran las respuestas posibles del oponente

Solución (p)

- Elegir un movimiento tal que se maximize el tablero después del mejor movimiento del oponente

2.- Búsqueda MINIMAX

- El algoritmo MINIMAX está diseñado para determinar la estratégia óptima para MAX, y así decidir que movimiento es el mejor

- IDEA: El computador hará un movimiento tal que, cuando MIN realice su mejor movimiento, la configuración del tablero estará en la mejor posición para MAX

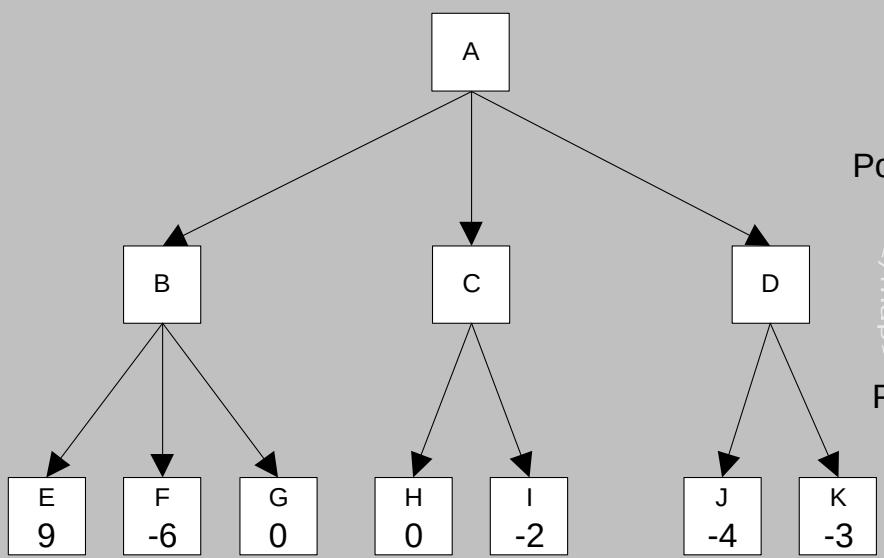
asume que MIN juega lo mejor posible

- ¿Como llevarla a cabo?
  - Para saber el mérito de un movimiento:
    - 1.- aplicar la función de evaluación a los inmediatos sucesores
    - 2.- mirar hacia delante y anticiparse a características que aparecerán más adelante
      - avanzar y luego propagar hacia atrás las valoraciones

- El jugador haría el movimiento que maximice las posibilidades del jugador (p.e. utilizando la función de evaluación heurística):
  - Generar todos los movimientos legales posibles
  - Evaluar las configuraciones del tablero resultantes
  - Realizar el movimiento con la configuración del tablero con el máximo valor

Algoritmo MINIMAX

- El algoritmo tiene 5 pasos:
  - 1.- Mirar varios movimientos hacia delante:
    - Generar todo el árbol hasta los estados terminales
      - Con una búsqueda en profundidad
  - 2.- Aplicar la función de utilidad a cada estado terminal para obtener su valor



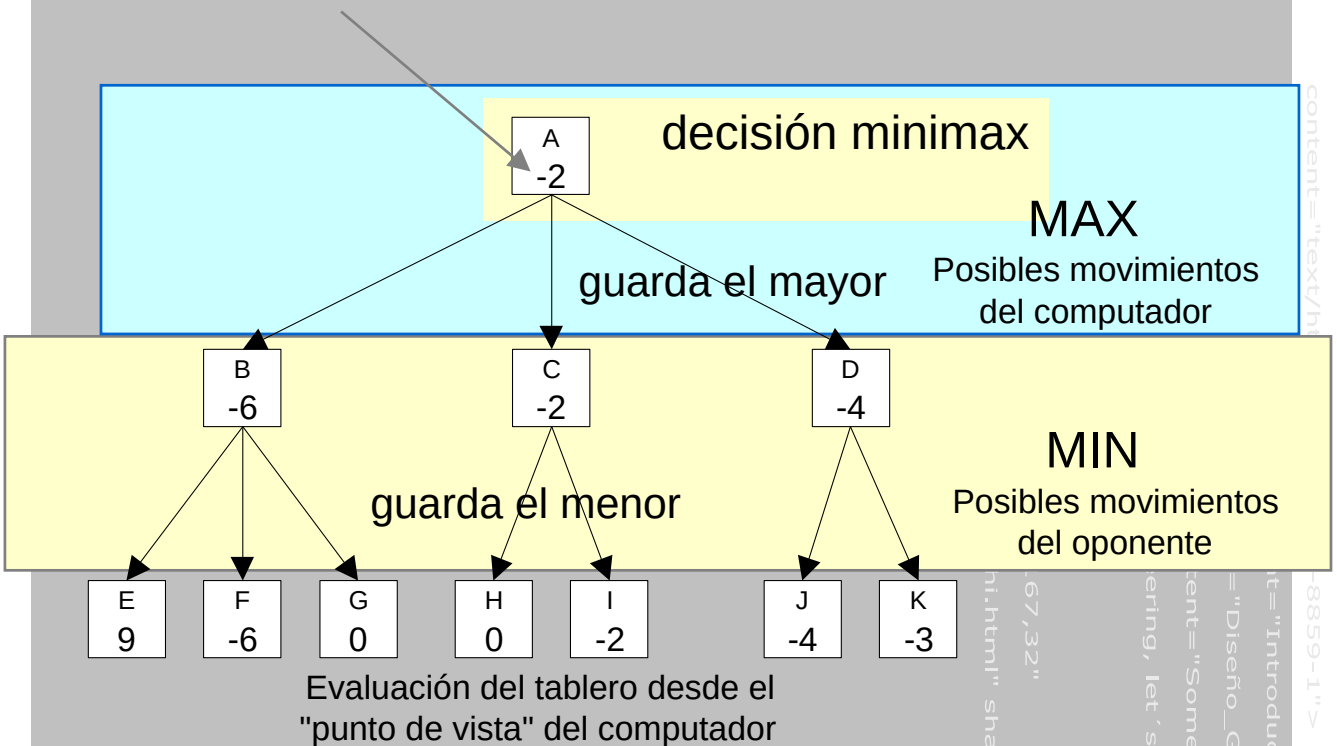
MAX
Posibles movimientos del computador

MIN
Posibles movimientos del oponente

Evaluación del tablero desde el "punto de vista" del computador

Algoritmo MINIMAX 2

- 3.- Usar la utilidad de los nodos terminales para determinar la utilidad de los nodos inmediatamente superiores del árbol.
  Valor del padre:
  - El hijo es un movimiento de MIN:
    Mínimo de todos los hijos inmediatos
  - El hijo es un movimiento de MAX:
    Máximo de todos los hijos inmediatos
- 4.- Continuar ascendiendo hasta la raiz (un nivel cada vez)
- 5.- MAX elige el movimiento que conduce al valor más alto



Evaluación del tablero desde el "punto de vista" del computador

Características del minimax

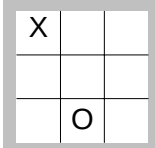
- Complejidad en tiempo:
  - Hay que explorar todos los nodos hasta la profundidad decidida
  - $O(b^m)$
    - b = factor de ramificación
    - m = profundidad alcanzada
- Complejidad en espacio
  - Si se realiza una implementación recursiva es como el algoritmo de búsqueda en profundidad
    - complejidad lineal con b y m
- El coste en tiempo lo hace totalmente impracticable.
  - Sólo útil como base para métodos más realísticos o para el análisis matemático de juegos

Ejemplos de funciones de evaluación heurística para juegos

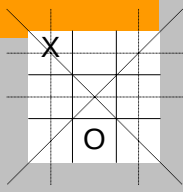
- Significaría estimar como de buena es la configuración corriente del tablero para un jugador dado
  - Tipicamente, significaría estimar como de buena es su situación y como es para su oponente y entonces restar las puntuaciones de los jugadores
  - Los valores típicos podrían variar de -infinito (pierde) a +infinito (gana) o [-1,+1]
  - Ajedrez: valor todas las blancas - - valor todas las negras
  - Damas: num. blancas - num. negras
  - 3-en-raya?

- Características de las funciones de evaluación heurística:
  - Debe estar de acuerdo con la función de utilidad para los nodos terminales
  - No debe ser muy compleja
  - Debe reflejar de manera fiable las posibilidades actuales de ganar

Ejemplo de heur'istica para el 3-en-raya

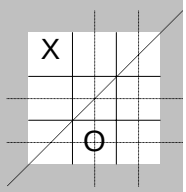


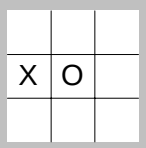
X tiene 6 posibilidades ganadoras



e(p) = 6 - 5 = 1

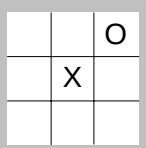
O tiene 5 posibilidades ganadoras





X tiene 4 posibilidades ganadoras
O tiene 6 posibilidades

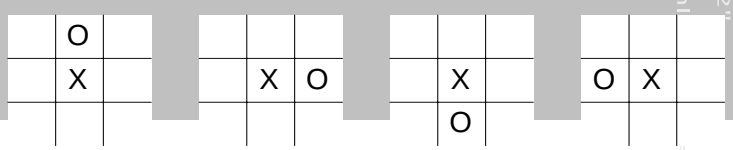
e(p) = 4 - 6 = -2



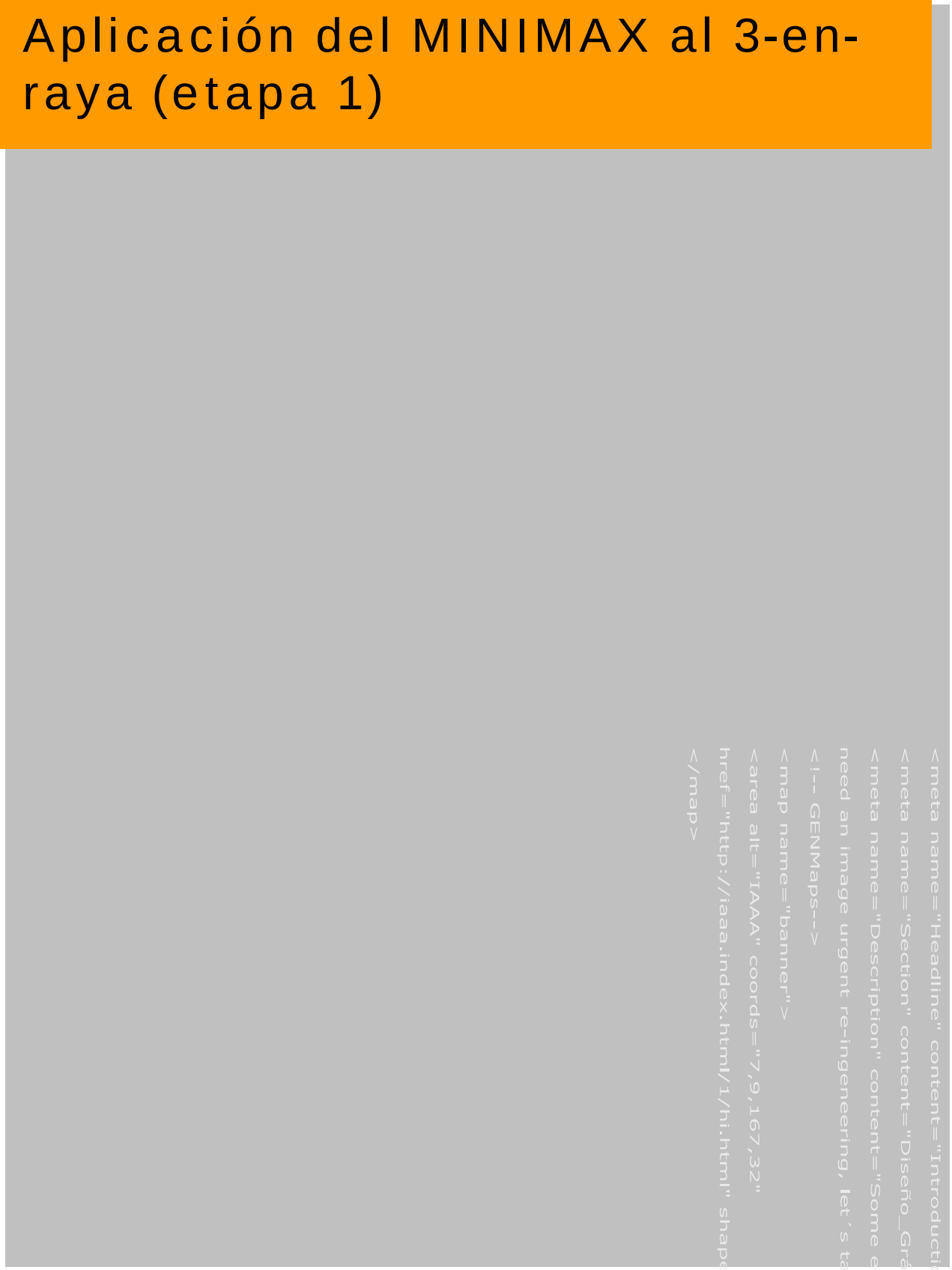
X tiene 5 posibilidades ganadoras
O tiene 4 posibilidades

e(p) = 5 - 4 = 1

- Función de evaluación *e(p)* para una posición *p*:
  - En *p* no gana ninguno
    - *e(p)* = (num. filas, columnas y diagonales completas utilizables por MAX) - (num. filas, columnas y diagonales completas utilizables por MIN)
  - En *p* gana MAX
    - *e(p) = +infinito*
  - En *p* gana MIN
    - *e(p) = -infinito*
  - Para disminuir el factor de ramificación se consideran posiciones simétricas como idénticas



Aplicación del MINIMAX al 3-en-raya (etapa 1)



Aplicación del MINIMAX al 3-en-raya (etapa 2)

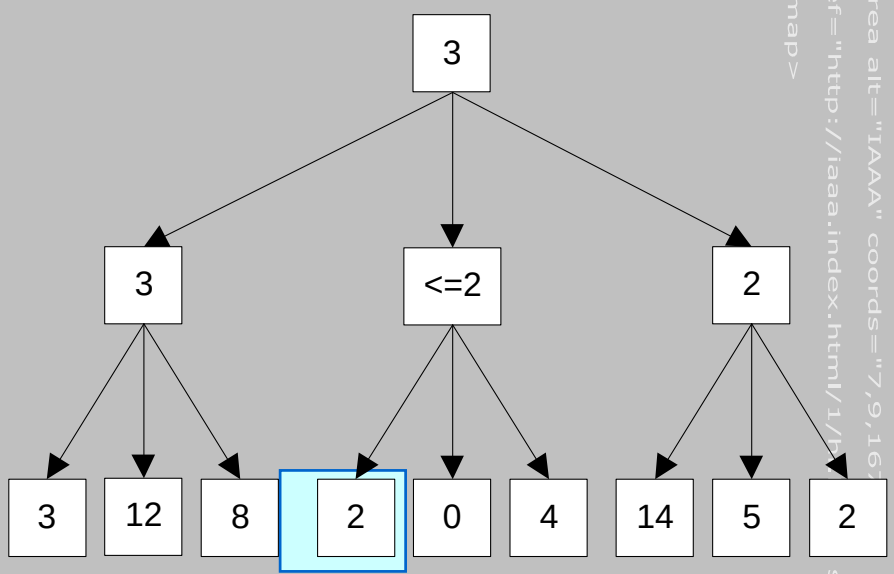
Aplicación del MINIMAX al 3-en-raya (etapa 3)

Ejemplo de heurística para Conecta-4

- 1. Para cada jugador, da puntos para cada posible
línea de longitud 4
  - 1. Si el adversario tiene cualquier pieza en la línea, 0 puntos
  - 2. Si todos son blancos en la línea, 0 puntos
  - 3. Si hay 1 pieza del jugador en la línea y 3 blancos, 1 punto
  - 4. Si hay 2 piezas del jugador en la línea y 2 blancos, 1 punto
  - 5. Si hay 3 piezas del jugador en la línea y 1 blanco, 4 puntos
  - 6. Si hay 4 piezas del jugador en la línea, 1000 puntos
- 2. Resta la puntuación del adversario de la puntuación del jugador

3.- Explorando menos estados

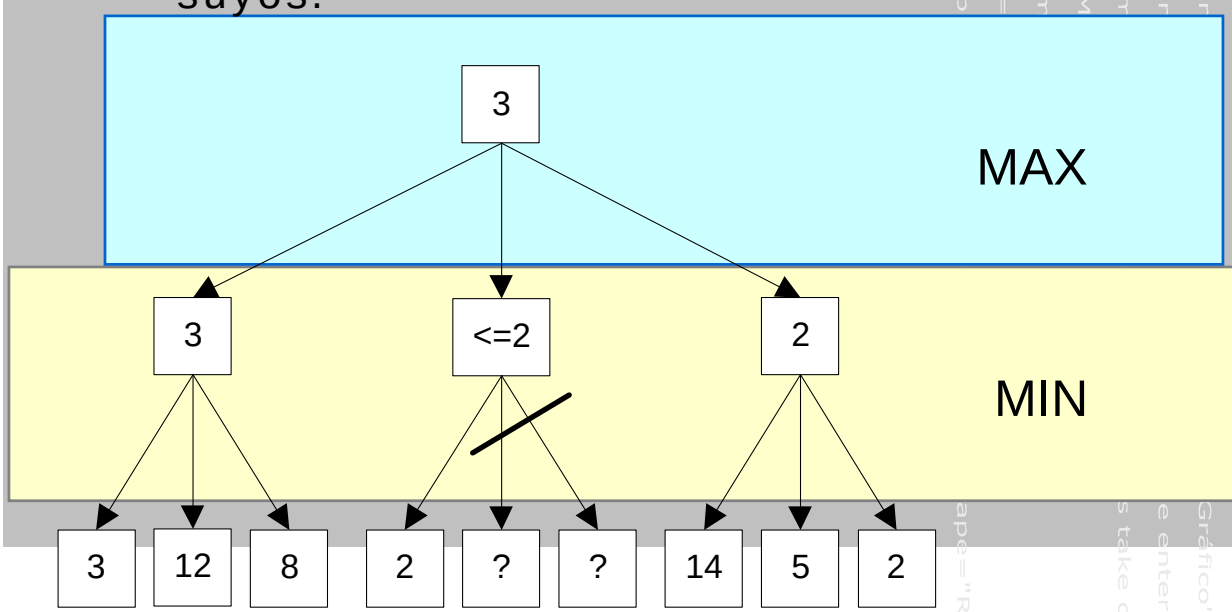
- Minimax es todavía impracticable
  - p.e. buen programa 1000 posiciones/seg
    - ajedrez 150 seg/jugada 150000 pos examinadas
    - con un b=35 unos 3 o 4 plys
    - un jugador medio puede prever de 6 a 8
- Sin embargo, expandir el árbol de juego completo en Minimax es hacer algo más que lo necesario
- Es posible encontrar la decisión minimax sin mirar todos los nodos del árbol
- Idea
  - recuerdese que la búsqueda minimax es en profundidad



Podas de alfa-beta en Minimax

- **Alfa**: (para un nodo MAX) es el valor más **alto** visto hasta el momento de los valores finales, calculados hacia atrás, de sus sucesores.
- **Beta**: (para un nodo MIN) es el valor más **bajo** visto hasta el momento de los valores finales, calculados hacia atrás, de sus sucesores.

- Poda (fin de la llamada recursiva)
  - Puede suspenderse la exploración por debajo de cualquier nodo MIN que tenga valores de Beta menores o iguales que el valor Alfa de cualquiera de sus nodos MAX ascendientes suyos.
  - Puede suspenderse la exploración por debajo de cualquier nodo MAX que tenga valores de Alfa mayores o iguales que el valor Beta de cualquiera de sus nodos MIN ascendientes suyos.



Efectividad de la poda alfa-beta

- La efectividad depende del orden en que se generan los nodos
  - Si se prueba primero con movimientos buenos, mas poda
  - Si se prueba primero con movimientos malos, menos poda
  - Puede ser útil tratar de examinar al principio los caminos más prometedores
- Peor caso como minimax
- Mejor caso
  - Si se puede elegir perfectamente el orden de los caminos
  - $O(b^{d/2})$ en lugar de $O(b^{d})$ nodos de minimax
  - factor de ramificación efectivo es la raiz cuadrada
    - p.e. ajedrez 6 en lugar de 35
      - se pueden mirar 8 pasos adelante en lugar de 4

Ejemplo para ilustrar la poda alfa-beta

Estatus de los programas de juegos

- Tic Tac Toe
  - Al nivel del mejor jugador del mundo
- Othello
  - El computador mejor que cualquier humano. Los campeones ahora rechazan jugar.
  - Campeon mundial IAGO de Rosenbloom de CMU
- Damas
  - El computador (Chinook) ganó al segundo del ranking mundial pero perdió con el primero (1992)
- Ajedrez
  - El computador (Deep Blue) con una puntuación de 2560 está entre los 100 mejores del mundo, Kasparov tiene 2805
- Backgammon
  - Campeon mundial BKG de Berliner de CMU

REDES SEMANTICAS Y FRAMES

- Indice:
  - 1 Redes semánticas
    - 1.1 Conceptos de asociación
    - 1.2 Conceptos de herencia
    - 1.3 Relaciones especiales
    - 1.4 Conclusiones
  - 2 Frames
    - 2.1 Los frames de Minsky
    - 2.2 Los frames después de minsky
    - 2.3 Conclusiones

- Lecturas:
  - E. Rich y K. Knight, Cap 4.1, 4.2, 9.1, 9.2

<meta name="HTTP-EQUIV" content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32" href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

1.- Contexto en que se han desarrollado

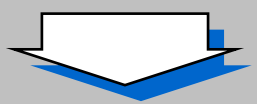
- A través de los años los filósofos y matemáticos han propuesto muchas formas diferentes de lógica. Objetivo:
  - Caracterizar los principios del razonamiento correcto (formalidad, completitud, ...)
    - Su énfasis está en conseguir operaciones en las que se preserve la verdad sobre expresiones bien formadas

- Las líneas de trabajo de psicologos y linguistas tratan de caracterizar la naturaleza del entendimiento humano. Objetivo:
  - Describir la forma en que los humanos adquieren y usan su conocimiento del mundo

(Los informáticos no han acordado todavía que forma de lógica es, o sería, apropiada para representar conocimiento del mundo real.)

1.1.- Conceptos de asociación

- Teorias asociacionistas
  - Definen el significado de un objeto en términos de una red de asociaciones con otros objetos
  - Cuando se percibe o razona sobre un objeto, primero se hace una correspondencia con un concepto en nuestra mente
  - Este concepto es parte de nuestro conocimiento del mundo y está conectado a otros objetos a través de apropiadas relaciones
  - Ejemplo: “la nieve es blanca”

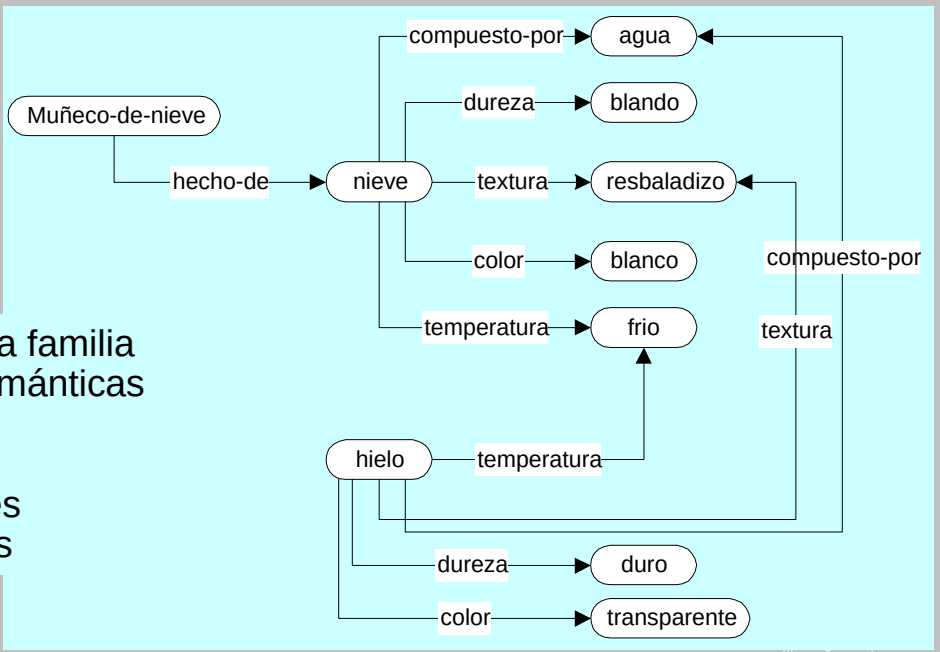


NIEVE
fria, blanca, resbaladiza, muñecoDeNieve, hielo

- Almacenar objetos y sus asociaciones con otros objetos

¿Como es una red semántica?

- Una red semántica es una estructura en forma de grafo utilizada para representar conocimiento:
  - El conocimiento puede ser codificado como nodos y conexiones en una red
  - **Toda** la información asociada a un nodo (objeto) está relacionada con él a través de conexiones (relaciones).
  -



hay toda una familia de redes semánticas varían en:
- nodos
- conexiones
- inferencias

- A diferencia de la LPO, no existe ningún formalismo de redes semánticas "estandar", ni una notación "estandar". Se crea a medida que se va trabajando.
- La investigación se centra en
  - que tipos de nodos y conexiones debe haber y
  - como deben ser manipulados.

Redes semánticas de Quillian

- Quillian, CMU, 1966
  - Aplicación: comprensión del lenguaje natural
  - Creador del concepto de redes semánticas
  - Introdujo conexiones superset/subset (relaciones taxonómicas)
  - Nodos codifican objetos y atributos
  - Conexiones codifican relaciones entre objetos: and, or, modificación

áquí dibujo de Quillian

Tres planos representando tres definiciones de la palabra “planta”

Inferencia: Búsqueda de intersección

- Quillian 1968

Un tipo de inferencia para contestar a preguntas sobre relaciones entre objetos

(utilizado en las primeras redes semánticas)

- Se activa cada uno de los nodos y observa donde se encuentran las activaciones

- Ejemplos
  - ¿Cual es la relación entre muñeco-de-nieve y frio?
  - Quillian ¿relación entre dos palabras?

1.2.- La mente organiza el conocimiento jerárquicamente

- Los humanos organizan su conocimiento jerárquicamente, con la información asociada en los niveles apropiados más altos en la jerarquía

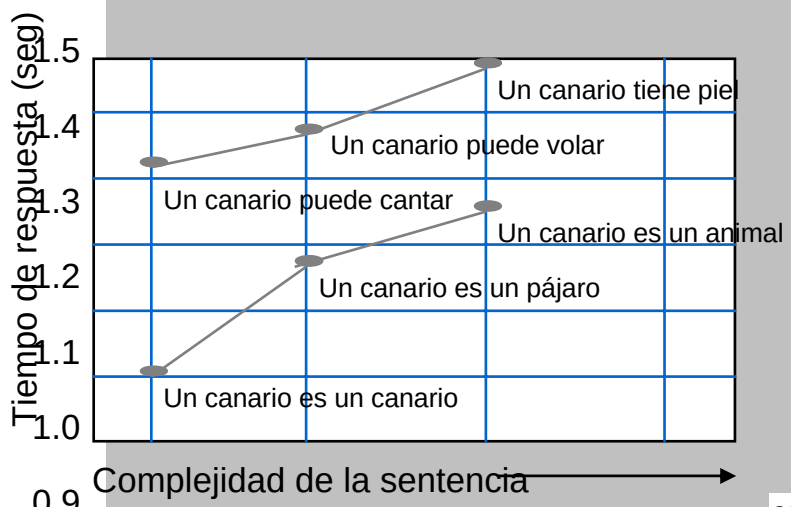
  - Collins y Quillian (1969)
    Objetivo modelar gestión y almacenamiento de información en humanos.
    - Test ejemplo: ciertas propiedades de los pájaros:
      - ¿Un canario es un pájaro? t1
      - ¿Puede cantar un canario? t2
      - ¿Puede volar un canario? t3

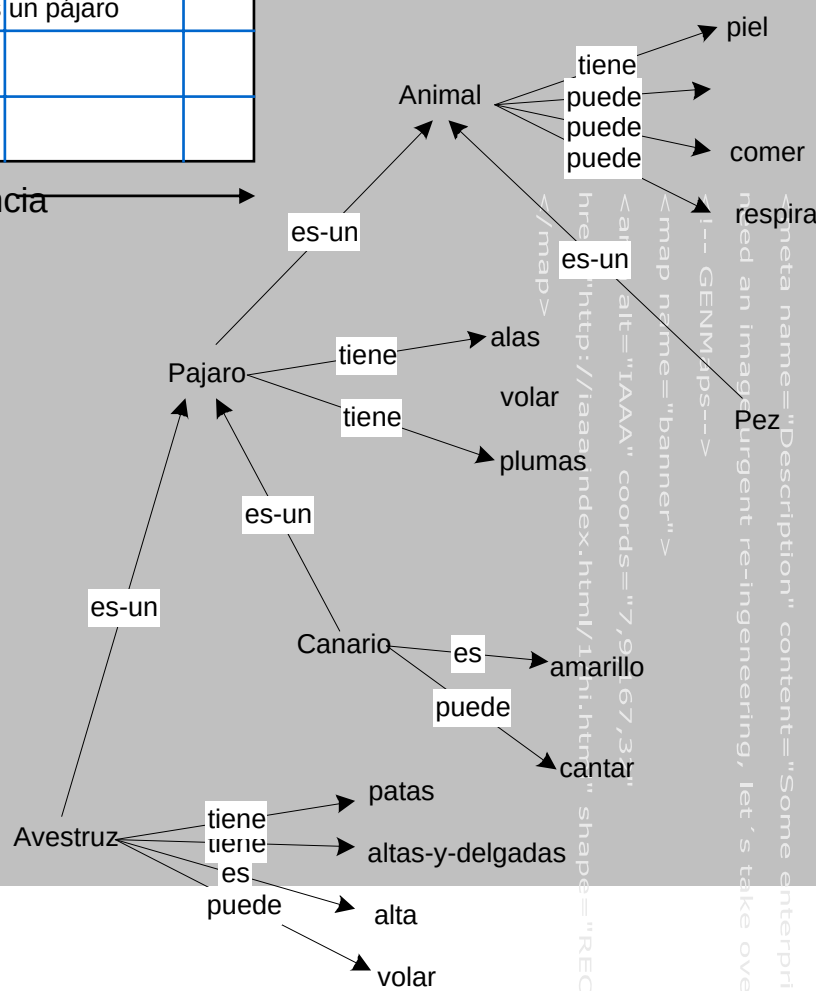
    tiempos de respuesta

    - t2 < t3
      lo mas rápido fué lo más específico del canario
    - Cuando se preguntó si un avestruz podía volar, la respuesta fué más rápida que a la pregunta de si podía respirar

Resultado

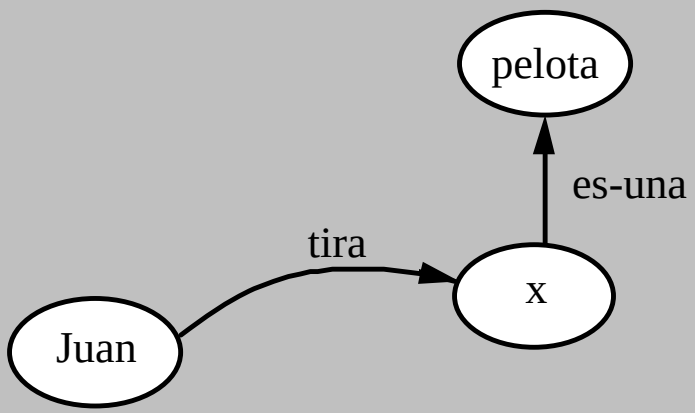
- Red semántica desarrollada por Collins y Quilian en su investigación sobre almacenamiento de información humana y tiempos de respuesta





Juan tira la pelota

- Red:

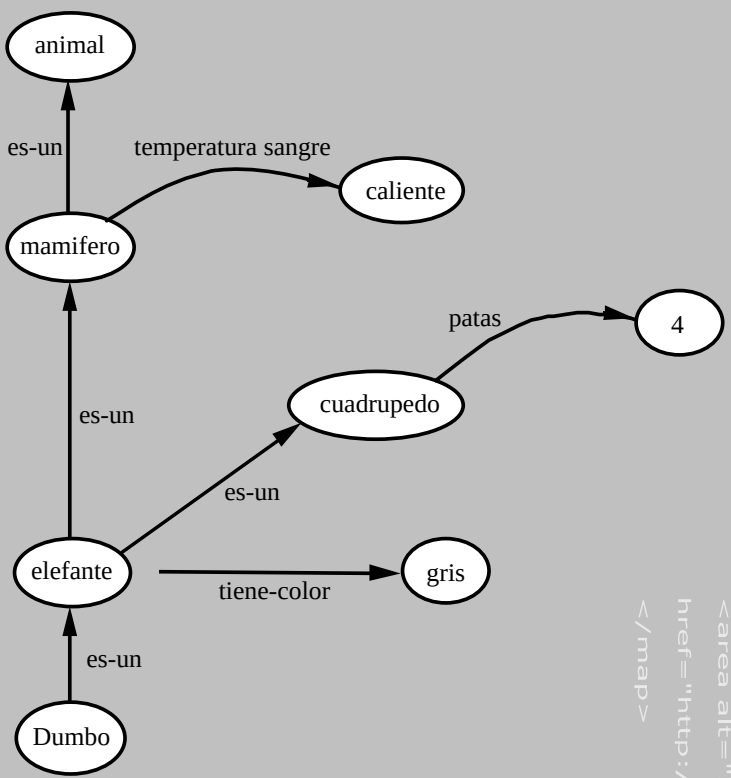


- Lógica:
  - $\exists\ x.\ Tira(Juan, x) \wedge Pelota(x)$

- Problema:
  - Nos gustaría aportar información extra sobre el evento "Tira", tal como el tiempo, lugar, o intensidad.

Inferencia: Herencia de propiedades

- Herencia es el algoritmo de inferencia más común en los sistemas de redes semánticas.
  - A través de relaciones es-un e instancia



- Las preguntas son de la forma: "cuál es el valor de la propiedad *p* para el nodo *x*"
  - Respuesta cuando
    - *x* tiene la propiedad *p*
    - *x es-un y* e *y* tiene la propiedad *p*
    - *x es-un* z_1, z_1 *es-un* z_2, ... , z_{n-1} *es-un* z_n, y z_n tiene la propiedad *p*
  - En otro caso, la respuesta está indefinida

Razonamiento por defecto

- ¿Y si hay múltiples posibles respuesta?

$\forall x\ Mamifero(x) \rightarrow NumPatas(x) = 4$

$\forall x\ Humano(x) \rightarrow Mamifero(x)$

$\forall x\ Humano(x) \rightarrow NumPatas(x) = 2$

- Problema en Lógica:
  - Esto es inconsistente, puesto que de *Humano(Juan)* podemos deducir que 2 = 4.
  - Solución:
    - No podemos usar LPO.
    - Debemos usar lógica por defecto o lógica no monótona.
      ¡ Pero estas lógicas no son mecanizables !

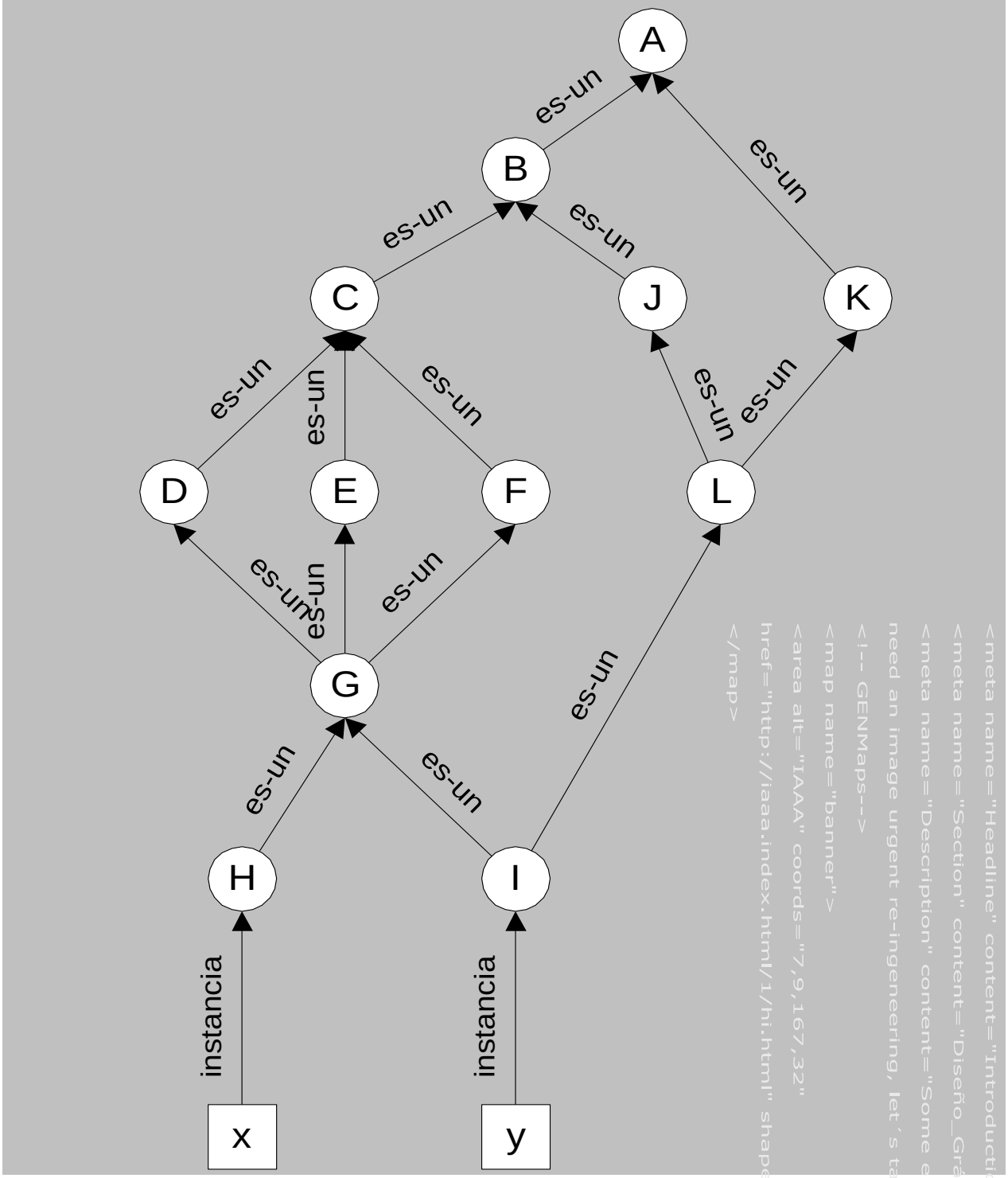
- Solución en redes semánticas:
  distancia inferencial

Distancia inferencial

- Touretzky 1986
  "The mathematics of inheritance systems"

  - Si α y β son antecesores de x, y α es antecesor de β , entonces β es inferencialmente mas cercano a *x* que α.
    - Si α y β tienen ambos un valor para la propiedad *p* , *x* heredará su valor para la propiedad *p* de β (e ignorará completamente el valor para *p* de α)
  - Si α no es un antecesor de β y β no es un antecesor de α , entonces α y β estan en conflicto en lo que se refiere a la herencia de *x* : si α y β tienen diferentes valores para la propiedad *p* , entonces *x* no tendrá un valor para la propiedad *p*
  - Los nodos que no son antecesores de *x* son completamente irrelevantes en lo que se refiere a la herencia de *x*.

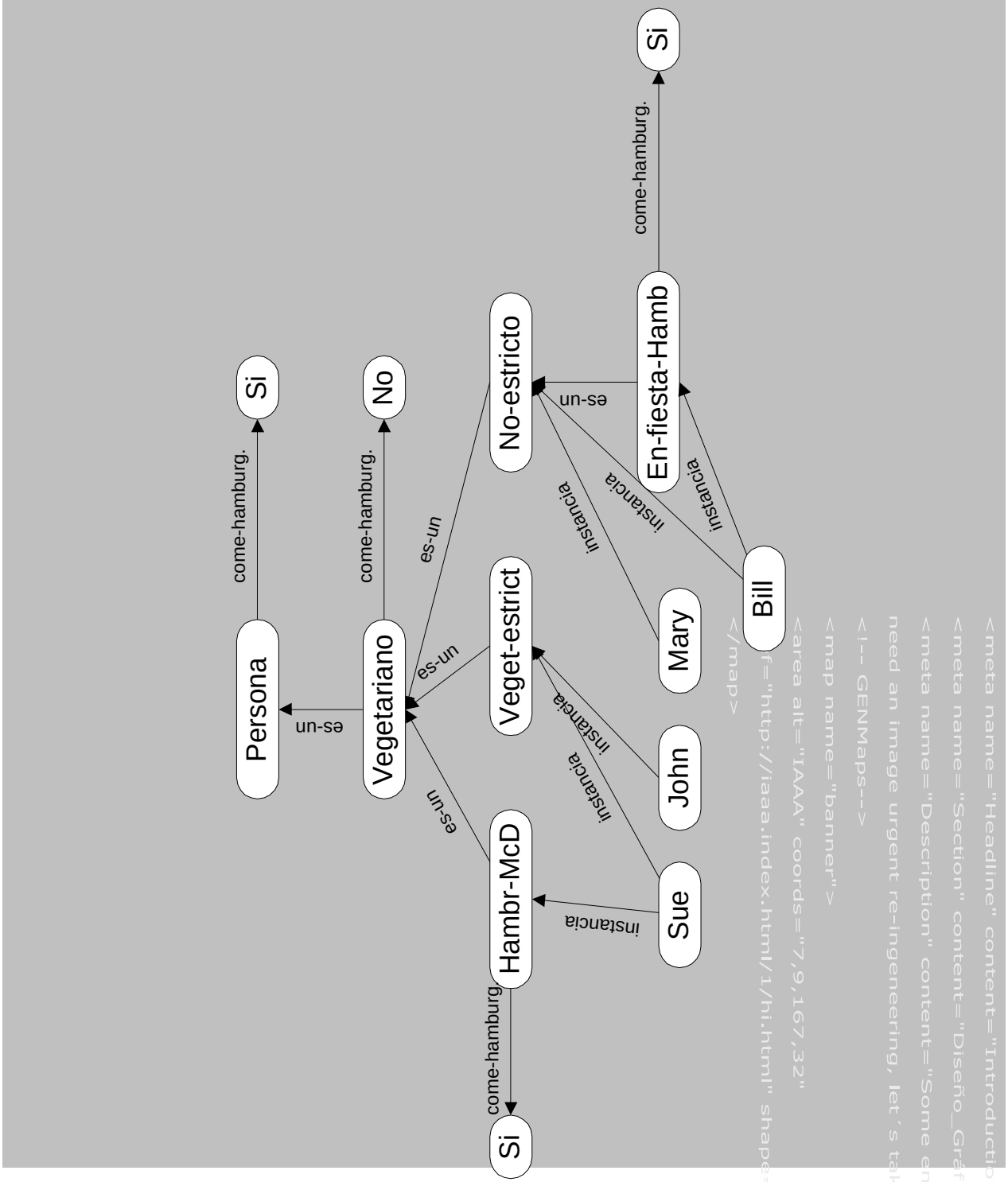
Ejemplo distancia inferencial



Un ejemplo más grande de redes semánticas

- Dibujar la una red semántica que represente la siguiente información:
  - La mayoría de las personas come hamburguesas
  - Algunas personas son vegetarianas
  - Los vegetarianos no comen hamburguesas, a menos que se encuentren hambrientos-en-frente-de-un-McDonalds
  - Los vegetarianos puen ser estrictos y no-estrictos
  - Los vegetarianos-no-estrictos comen hamburguesas si se encuentran en una fiesta-de-sólo-hamburguesas
  - John y Sue son vegetarianos-estrictos
  - Mary y Bill son vegetarianos-no-estrictos
  - Bill está en una fiesta-de-sólo-hamburguesas
  - Sue se encuentra hambrienta-en-frente-de-un-McDonalds

Red semántica del ejemplo



Inferencia en redes semánticas

- ¿Comió John alguna hamburguesa?
- ¿Comió Sue alguna hamburguesa?
- ¿Comió Mary alguna hamburguesa?
- ¿Comió Bill alguna hamburguesa?

1.3.- Relaciones especiales

- Capacidad expresiva de redes semánticas parecida al CPPO
- La notación gráfica de las relaciones facilita un acceso más eficiente que la resolución
- Ciertas relaciones tienen asociadas sus propias reglas de inferencia
  - Herencia:
    - permite almacenar información al mayor nivel de abstracción
    - reduce el tamaño de la base de conocimiento
    - reduce incosistencias en la actualización
    - asociado un algoritmo especializado que conoce y apovecha las conexiones
- Aunque el formalismo es altamente expresivo es poco restringido (como CPPO) y deja gran carga de trabajo al programador
  - Solución: más rico conjunto de asociaciones con semántica del dominio e implementarlas como parte del formalismo para que no las tenga que crear el diseñador

Añadiendo relaciones al formalismo

- Simmons 1973 (basado en Fillmore 1968) (trabajos en el contexto del lenguaje natural)

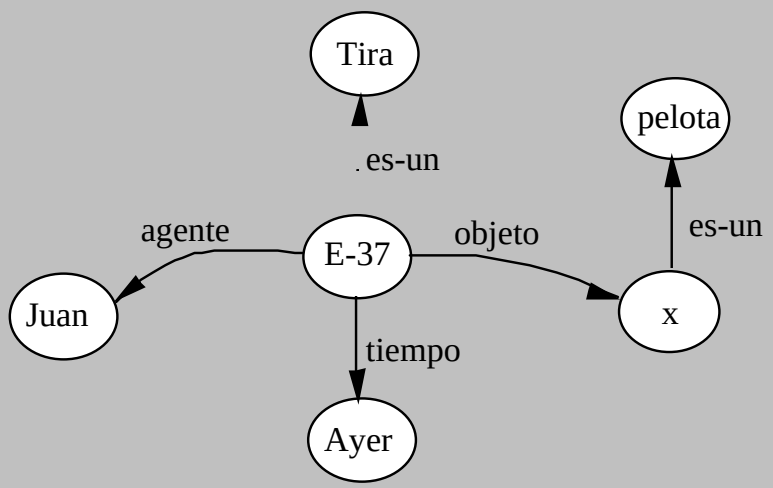
Abordan la necesidad de relaciones en la estructura de las frases

- la sentencia se representa por un nodo verbo con conexiones a los participantes definiendo relaciones del tipo:
  - agente,
  - objeto,
  - instrumento,
  - localización,
  - tiempo

- El lenguaje de representación captura gran parte de la estructura profunda
  - relaciones entre verbo y su sujeto
  - relaciones entre verbo y su objeto

Juan tiró la pelota ayer

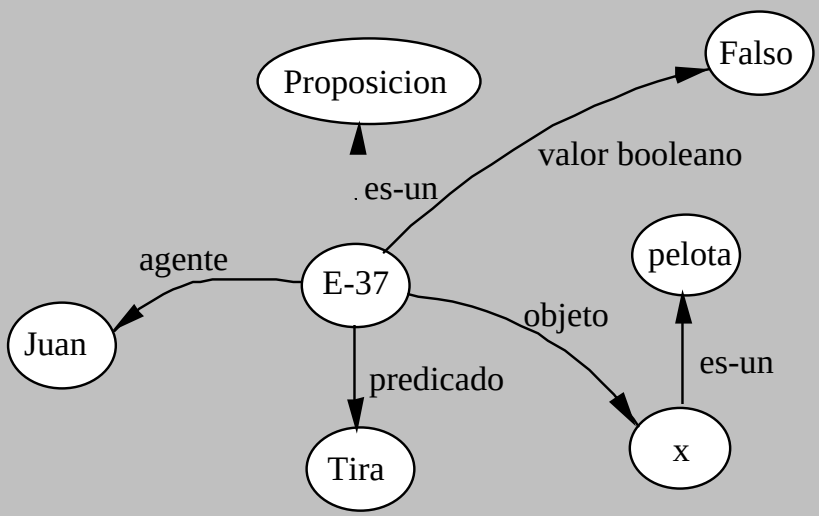
- Red:



- Lógica:
  - $\exists x\ z\ w.\ Tira(Juan, x, Ayer, z, w) \wedge Pelota(x)$ *donde z y w denotan el lugar y la intensidad.*
  - ¡Resulta un poco engorroso!
- Problema:
  - ¿Como podemos decir que Juan NO tiró la pelota?
  - Es fácil en lógica pero no con la formulación de redes semánticas.

Juan no tiró la pelota

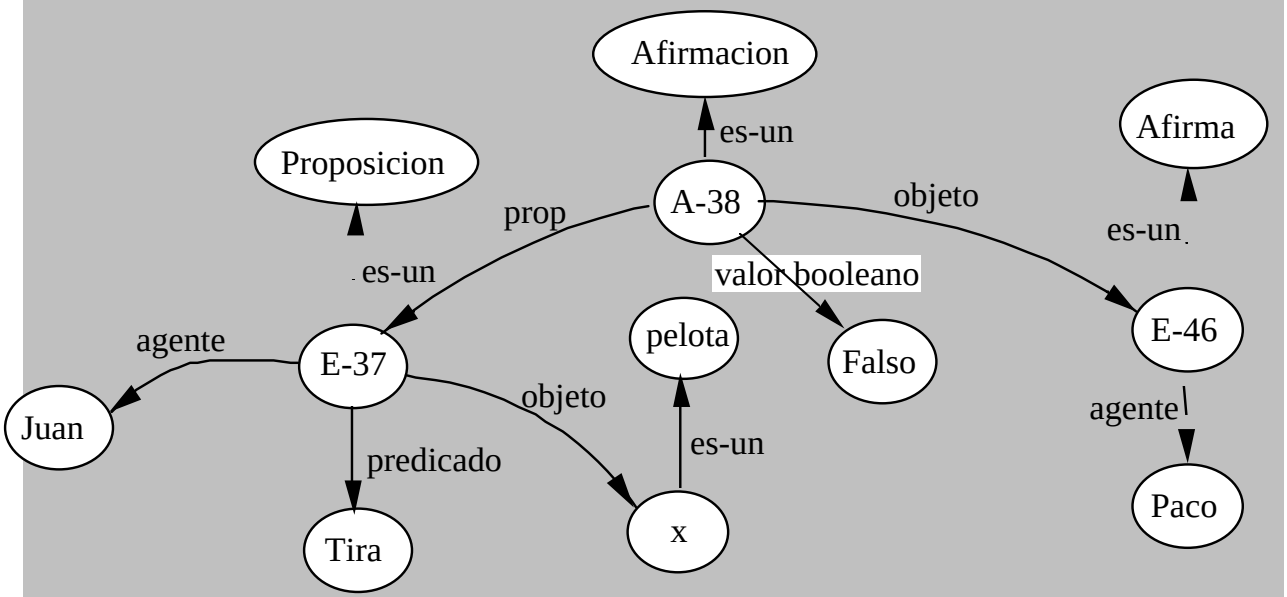
- Red:



- Lógica:
  - *∃x. ~ ∃y z w. Tira(Juan, x, y, z, w)∧ Pelota(x) donde y, z y w denotan el tiempo, el lugar y la intensidad.*
- Problema:
  - ¿Como podemos hablar de una proposición tal como: "Paco insiste en que Juan tiró la pelota"?

Paco afirma que Juan tiró la pelota

- Red:



- Lógica:
  - No hay forma de decir esto en CPPO.
  - Se debe recurrir a algún de tipo de lógica modal.

- Observación:
  - Cuando se incrementan nuestras demandas representacionales debemos inventar continuamente nueva maquinaria de red semántica.
  - En lógica, debemos inventar nuevas piezas de máquinaria lógica (p.e. operadores modales o valores de verdad extra).

Lenguajes basados en redes semánticas

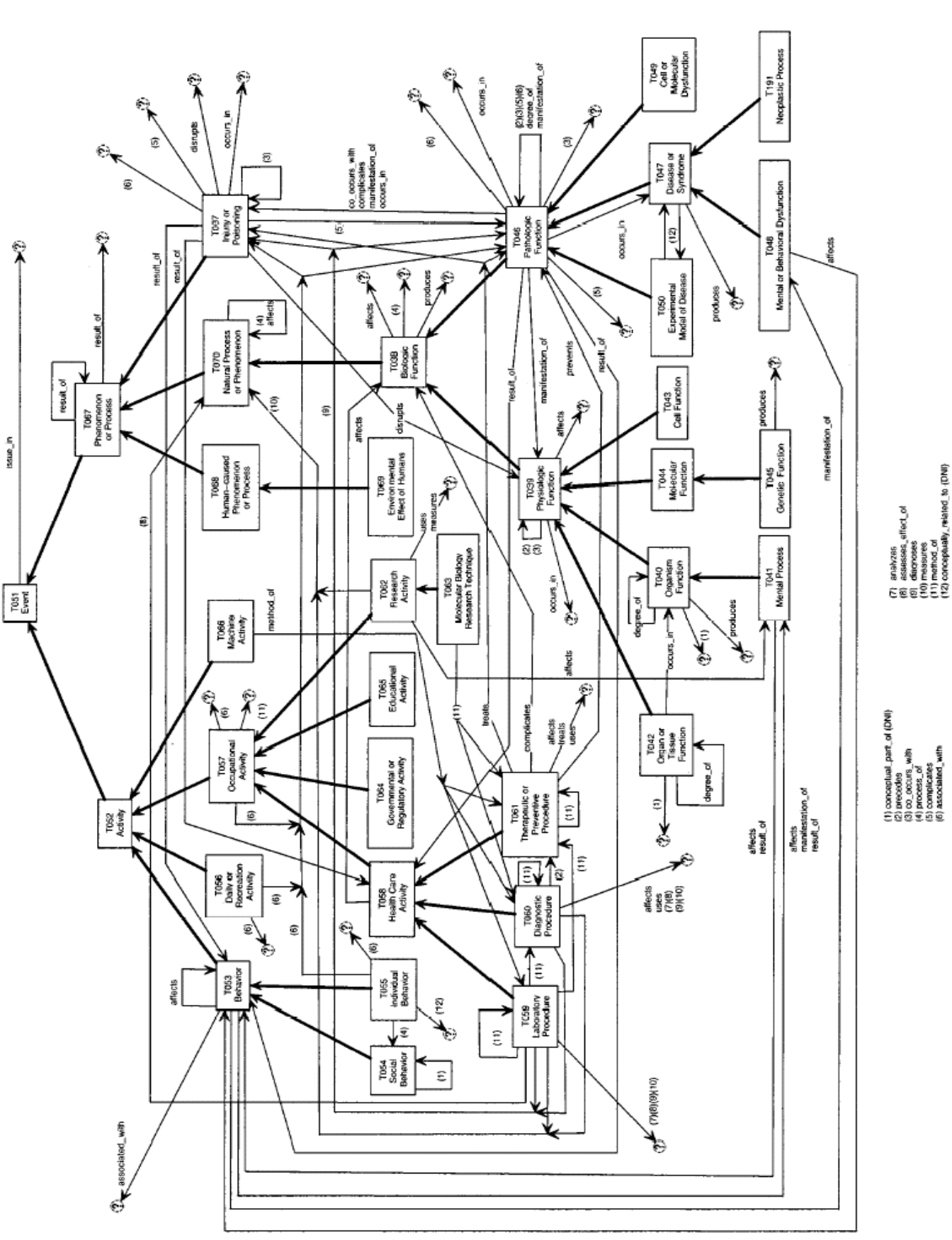
- La alternativa más común encontrada en IA son las redes semánticas.
  Constituye la teoría central en la aproximación de Inteligencia Artificial para representar el conocimiento.

- Provienen de las teorias asociacionistas (memoria asociativa)

- La mayor parte de las aplicaciones se han desarrollado en el área de comprensión del lenguaje natural

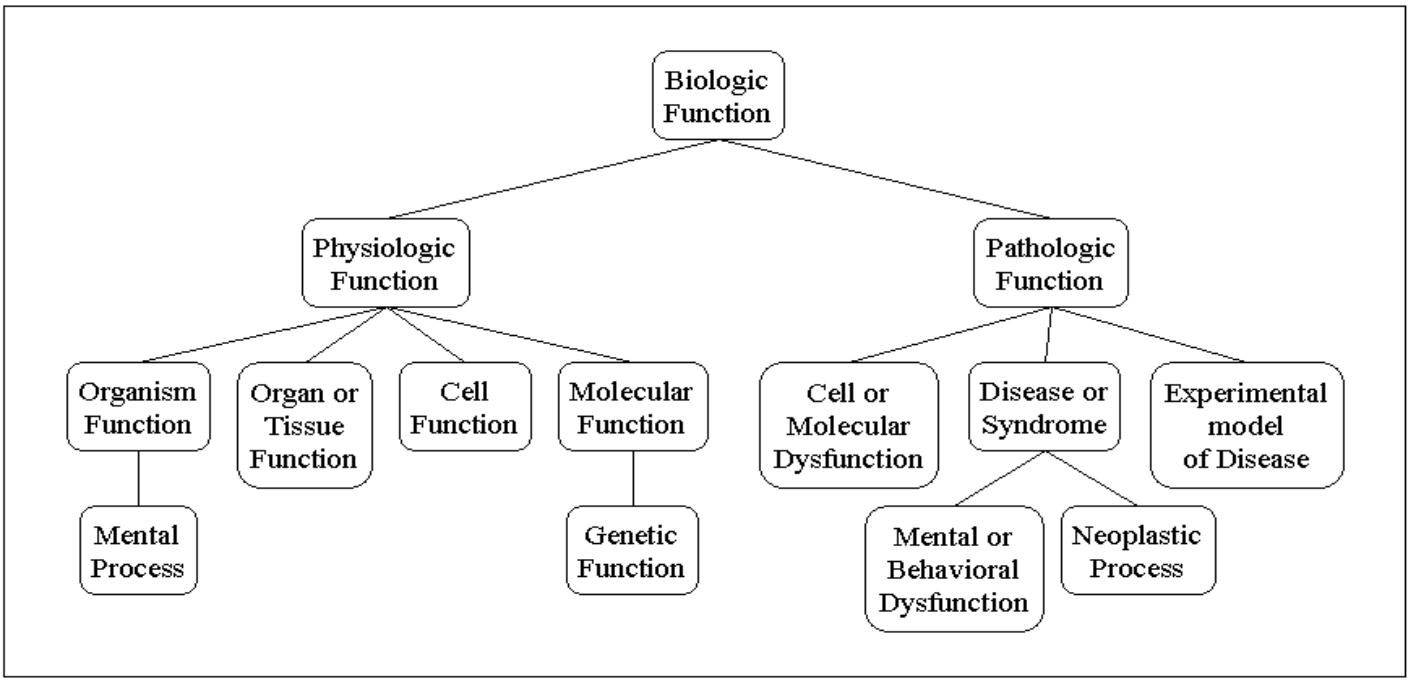
- Por ejemplo:
  - La mayoría de los lenguajes de IA, tales como FRL, KRL, KLONE, AIMDS, y SRL están basados en el paradigma de las redes semánticas.
- Excepciones:
  - Lenguajes de programación ad hoc (LISP, OPS5), y lenguajes basados en la lógica tales como PROLOG. (Por supuesto, sobre ellos es posible construir lenguajes basados en redes semánticas)

UMLS

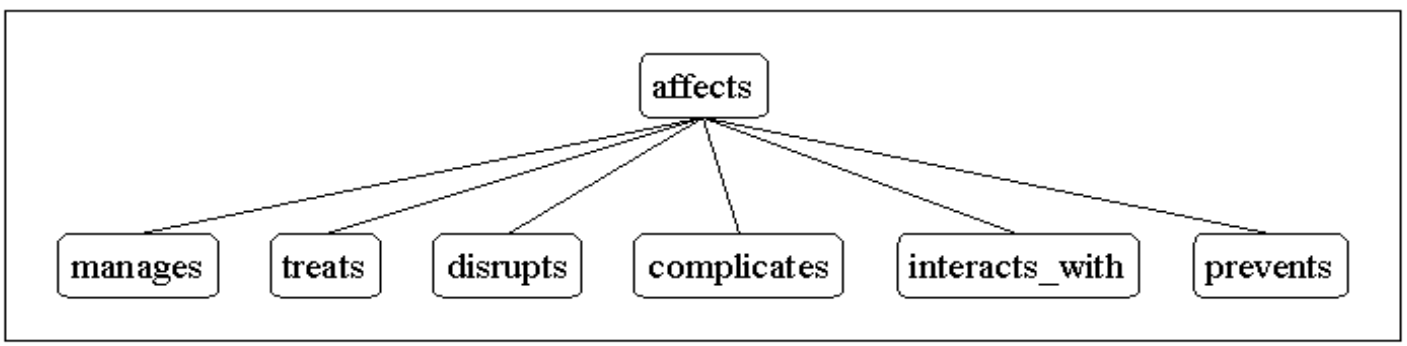

T037 Injury or Poisoning
T046 Pathologic Function
T047 Disease or Syndrome
T049 Cell or Molecular Dysfunction
T191 Neoplastic Process
T070 Natural Process of Phenomenon
T038 Biologic Function
T050 Experimental Model of Disease
T048 Mental or Behavioral Dysfunction
T067 Phenomenon or Process
T043 Cell Function
T068 Human-caused Phenomenon or Process
T069 Environmental Effect of Humans
T039 Physiologic Function
T044 Molecular Function
T045 Genetic Function
T062 Research Activity
T063 Molecular Biology Research Technique
T040 Organism Function
T041 Mental Process
T051 Event
T066 Machine Activity
T065 Educational Activity
T042 Organ or Tissue Function
T057 Occupational Activity
T064 Governmental or Regulatory Activity
T061 Therapeutic or Preventive Procedure
T052 Activity
T056 Daily or Recreation Activity
T058 Health Care Activity
T060 Diagnostic Procedure
T053 Behavior
T055 Individual Behavior
T059 Laboratory Procedure
T054 Social Behavior
issue_in
result_of
disrupts
occurs_in
co_occurs_with
complicates
manifestation_of
degree_of
produces
affects
prevents
uses
measures
method_of
treats
complicates
associated_with
(1) conceptual_part_of (DNI)
(2) precedes
(3) co_occurs_with
(4) process_of
(5) complicates
(6) associated_with
(7) analyzes
(8) assesses_effect_of
(9) diagnoses
(10) measures
(11) method_of
(12) conceptually_related_to (DNI)

UMLS – Jerarquías de tipos y relaciones

- Porción de jerarquía de tipos

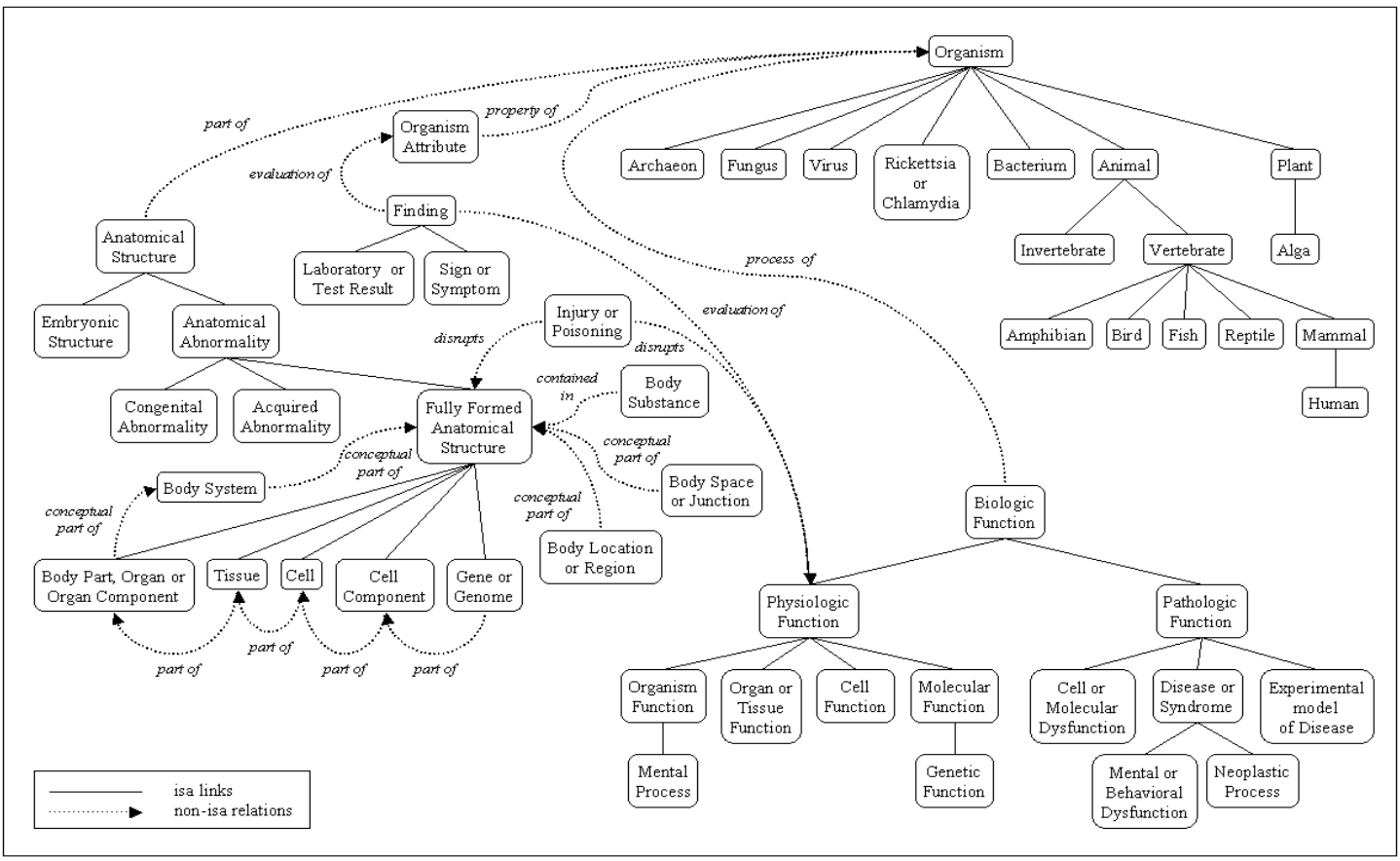


- Porción jerarquía de relaciones



UMLS – relaciones entre tipos

- Porción de red semántica que ilustra relaciones (jerárquicas o asociativas) que exiten entre tipos semánticos



UMLS – Listado de tipos

- **Entity**
  - Physical Object
    - Organism
      - Plant
        - Alga
      - Fungus
      - Virus
      - Rickettsia or Chlamydia
      - Bacterium
      - Archaeon
      - Animal
        - Invertebrate
        - Vertebrate
          - Amphibian
          - Bird
          - Fish
          - Reptile
          - Mammal
            - Human
    - Anatomical Structure
      - Embryonic Structure
      - Anatomical Abnormality
        - Congenital Abnormality
        - Acquired Abnormality
      - Fully Formed Anatomical Structure
        - Body Part, Organ, or Organ Component
        - Tissue
        - Cell
        - Cell Component
        - Gene or Genome
    - Manufactured Object
      - Medical Device
      - Research Device
      - Clinical Drug

- [Entity] (continued)
  - [Physical Object] (continued)
    - Substance
      - Chemical
        - Chemical Viewed Functionally
          - Pharmacologic Substance
            - Antibiotic
          - Biomedical or Dental Material
          - Biologically Active Substance
            - Neuroreactive Substance or Biogenic Amine
            - Hormone
            - Enzyme
            - Vitamin
            - Immunologic Factor
            - Receptor
          - Indicator, Reagent, or Diagnostic Aid
          - Hazardous or Poisonous Substance
        - Chemical Viewed Structurally
          - Organic Chemical
            - Nucleic Acid, Nucleoside, or Nucleotide
            - Organophosphorus Compound
            - Amino Acid, Peptide, or Protein
            - Carbohydrate
            - Lipid
              - Steroid
              - Eicosanoid
          - Inorganic Chemical
          - Element, Ion, or Isotope
      - Body Substance
      - Food

UMLS – Listado de relaciones

- isa
- associated_with
  - physically_related_to
    - part_of
    - consists_of
    - contains
    - connected_to
    - interconnects
    - branch_of
    - tributary_of
    - ingredient_of
  - spatially_related_to
    - location_of
    - adjacent_to
    - surrounds
    - traverses
  - functionally_related_to
    - affects
      - manages
      - treats
      - disrupts
      - complicates
      - interacts_with
      - prevents
    - brings_about
      - produces
      - causes

- [associated_with] (continued)
  - [functionally_related_to] (continued)
    - performs
      - carries_out
      - exhibits
      - practices
    - occurs_in
      - process_of
    - uses
    - manifestation_of
    - indicates
    - result_of
  - temporally_related_to
    - co occurs_with
    - precedes
  - conceptually_related_to
    - evaluation_of
    - degree_of
    - analyzes
      - assesses_effect_of
    - measurement_of
    - measures
    - diagnoses
    - property_of
    - derivative_of
    - developmental_form_of
    - method_of
    - conceptual_part_of
    - issue_in

1.4.- Conclusiones sobre redes semánticas

- Ideas que aporta
  - Memoria asociativa
  - Herencia: relaciones **is-a** e **instancia**

- Ventajas
  - Agrupación eficiente de piezas de conocimiento relacionadas.
    Todas las sentencias sobre Juan existen como arcos conectados a Juan
  - Algoritmos de inferencia eficientes (básicamente a través de las soportadas relaciones de herencia).
    - En muchos casos es suficiente con simples algoritmos de grafos.
    - Esto es mucho más eficiente que el proceso de resolución.
  - Notación gráfica conveniente
  - Las redes semánticas pueden representar cualquier cosa puesto que la notación se puede crear sobre la marcha

Más conclusiones

- Desventajas
  - No podemos decir que significan las primitivas de la red semánticasin mirar a los algoritmos de inferencia.
  - No existen garantías de consistencia, completidud o corrección de las primitivas ad oc y los algoritmos de inferencias que surgen (esto requeriría formalización)
  - No hay forma de decir:
    - “Paris está en algún Pais” sin decir cual??

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

2.- Frames (marcos)

- Bartlett 1932, Minsky MIT 1975
  Los sistemas basados en frames fueron propuestos originariamente por Marvin Minsky del MIT, en un artículo denominado:
  - "A Framework for Representing Knowledge"
  - Su trabajo está relacionado con el del psicólogo Jean Piaget.
  - Minsky estaba interesado en la ayuda que un sistema de frames podia proporcionar para controlar el razonamiento en un sistema de percepción en el campo de visión por computador o en la comprensión del lenguaje natural
- Análisis de una situación por un humano
  - 1) consulta las estructuras de su memoria sobre situaciones anteriores,
  - 2) selecciona la estructura que parece más cercana a la situación actual y la modifica, si es necesario, para que encaje lo mejor posible
- Minsky: formalismo basado en la reutilización de estructuras (frames) que representan patrones de situaciones.

2.1.- ¿Que son los frames?

- Los frames sirven como descripciones estereotípicas de situaciones u objetos.
  - Redes semánticas como estructuras de registro
  - Foco en expectaciones/por defecto

- Frame de una "habitación"
  - El frame de una "habitación" debería contener componentes como puerta, suelo, paredes, y techo.
  - El frame de la "habitación" debería incluir también cierta información esperada: "las paredes de la habitación son planas y se juntan en ángulos rectos".
  - Ciertos tipos de puertas dependen de ciertos tipos de habitaciones. Incluso antes de entrar en una habitación podríamos decir si se entra en un armario, una habitación o se sale al exterior.
  - Podrían crease subconceptos de "habitación", p.e. frames para "comedor", "dormitorio" o "baño".
  - En el frame del "baño", esperariamos no encontrar una puerta de cristal.

Ejemplo de situación

- Conexión de conceptos relacionados:
  - Gracias a la compartición de terminales, no es necesario dar las descripciones de cada objeto en cada frame nuevo.

gráfico habitación minsky

<meta name="HTTP-EQUIV"
content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises
need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32"
href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

Como hacer frente a la violación de espectativas

- PROCESO DE MATCHING
  - 1) se selecciona un frame usando heurísticas y predicciones
  - 2) se intenta establecer una correspondencia entre la información observada y la del frame: valores por defecto son sustituidos por valores observados
  - 3) Si el frame seleccionado no corresponde con la realidad, la red en la que se encuentra localizado puede sugerir un frame alternativo para describir el objeto o situación. Por ejemplo:
    - ¿como puede recuperarse un sistema de visión si durante el análisis de una escena se invoca el frame "silla" de forma incorrecta?

2.2.- Los frames después de Minsky

- Los sistemas actuales reflejan el espíritu anterior pero su forma ha evolucionado considerablemente
  - Objetivo adicional: manipular conocimiento difícil de manipular usando cálculo de predicados
    - tipicalidad,
    - valores por defecto,
    - excepciones e
    - información redundante o incompleta
    - Se quiere una estructura flexible, con capacidad para evolucionar
  - Nos interesa su aspecto computacional

Aspectos de interés para la programación

- Los frames son prototipos:
  - Representa información por defecto que caracteriza una familia de entidades.
  - Referencia para comparar objetos a ser reconocidos, analizados, clasificados.

Herencia
Clases
Instancias

- Generalmente están organizados en una estructura a tres niveles:

frame	slot	facet

Frame: entidad
Slot: propiedades de la entidad
Facet: características descriptivas o de comportamiento del slot

Y además
Programación orientada al acceso

- ¿Como se consigue la funcionalidad de los facets?
  Ataduras procedurales llamad

demons

  - Proporciona un mecanismo que permite a las funciones definidas por el usuario manipular los valores de los slots cuando son necesarios, y chequear restricciones.
    - Procedimientos IF-NEEDED, IF-ADDED, ...

se disparan al acceder a los slots

- al leer
- al borrar
- al añadir
- al sustituir

Paradigma orientado a objeto

OBJETOS Y ESPECIALIZACION
=
Clases + Instancias + Atributos
+
Relaciones, herencia
+
Razonamiento por defecto, por herencia

ENCAPSULACION
=
Estructuras de datos ocultas
+
Operaciones ocultas
+
Acceso exclusivo por envio de mensajes

Implementaciones de frames

- Pronto se dearrollaron algunas implementaciones de la teoría de frames. Uno de los primeros, un lenguaje llamado FRL (Frame Representation Language), fué desarrollado por Roberts y Goldstein del MIT.

- Implementaciones más recientes de frames:
  - KLONE, KRL, SRL, CRL
  - units (KEE)
  - objects (LOOPS)
  - schemata (Knowledge Craft)

<meta name="HTTP-EQUIV" content="text/html;charset=iso-8859-1">
<meta name="Headline" content="Introduction">
<meta name="Section" content="Diseño_Gráfico">
<meta name="Description" content="Some enterprises need an image urgent re-ingeneering, let´s take over.">
<!-- GENMaps-->
<map name="banner">
<area alt="IAAA" coords="7,9,167,32" href="http://iaaa.index.html/1/hi.html" shape="RECT">
</map>

2.3.- Conclusiones sobre frames

- Ventajas
  - Incluye provisión para la asociación de conocimiento procedural con el conocimiento declarativo almacenado como valores de slots (esto permite permite expresar conocimiento más allá del puramente lógico)
  - Estructura de control flexible; el comportamiento de FRL es facil de extender

- Desventajas
  - No hay forma de decir:
    - “Tversky no es el tutor de Jones” sin decir quien es
  - Conduce a una frágil teoría de la representación; hace difícil la especificación de la "corrección" del sistema.

3.- KEE. Clases

- Clase: descripción de una familia de objetos con la misma estructura y comportamiento

- Tienen dos componentes:
  - **Estático** : datos - *atributos* caracterizan estado
  - **Dinámico** : procedimientos - *métodos* comportamiento común de la clase

- Ejemplo ilustrativo:
  - gestión de artículos en unos grandes almacenes

Clase

Articulo

Atributos

referencia
descripcion
precioSinIVA
cantidad

Metodos

precio : precioSinIVA * 1.13
costeTransp : precioSinIVA * 0.05
eliminar (q) : cantidad <- cantidad - q
añadir (q) : cantidad <- cantidad + q

Clases (cont.)

- En KEE
  - Clases - UNIT
  - Atributos y métodos - SLOTS

```
UNIT Articulo

                                                { Campos }
MEMBER SLOT referencia

MEMBER SLOT descripcion

MEMBER SLOT precioSinIVA

MEMBER SLOT cantidad)
                                                 { Metodos }
MEMBER SLOT precio
       VALUE (LAMBDA (THISUNIT )
              (* (GET.VALUE THISUNIT 'precioSinIVA)
                 1.13))

MEMBER SLOT costeTransp
       VALUE (LAMBDA (THISUNIT )
              (* (GET.VALUE THISUNIT 'precioSinIVA)
                  0.05))

MEMBER SLOT eliminar
       VALUE RestaDeCantidad

MEMBER SLOT aniadir
       VALUE SumaACantidad
```

```
(DEFUN RestaDeCantidad (THISUNIT Q)
       (- (GET.VALUE THISUNIT 'cantidad) Q))

(DEFUN SumaACantidad (THISUNIT Q)
       (+ (GET.VALUE THISUNIT 'cantidad) Q))
```

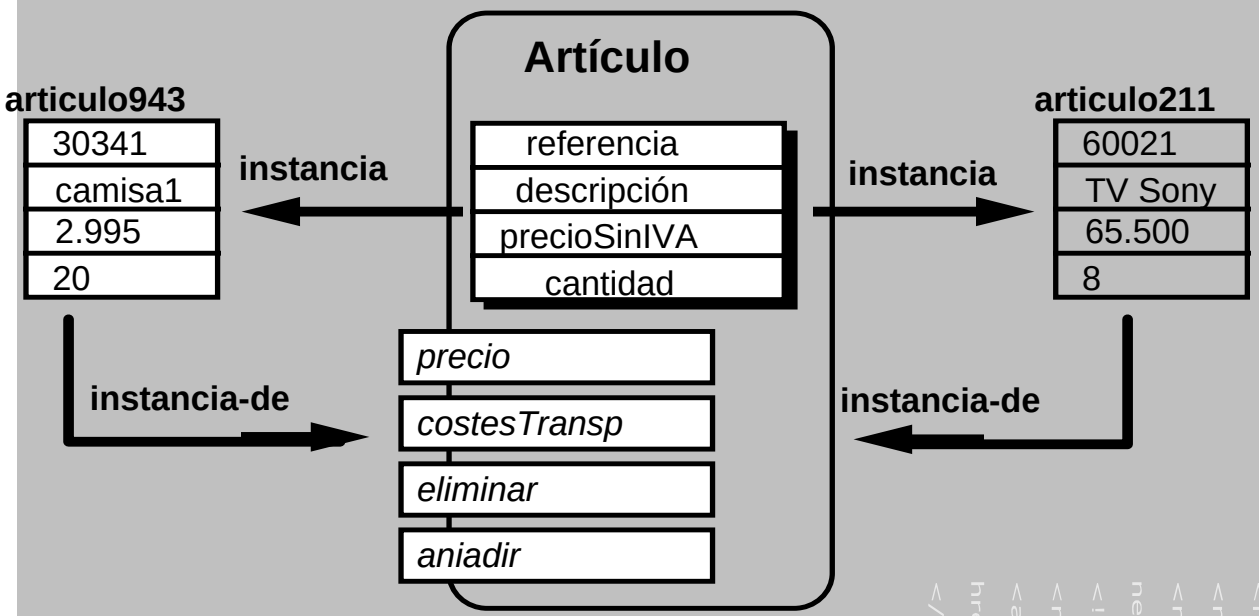
Algunos facets preexistentes

- en KEE:
  - VALUE
  - VALUECLASS
  - DEFAULT

```
UNIT Articulo

                                                    { Campos }
MEMBER SLOT referencia

MEMBER SLOT descripcion

MEMBER SLOT precioSinIVA

MEMBER SLOT cantidad)
       VALUECLASS     NUMBER
       DEFAULT        0
                                                    { Metodos }
MEMBER SLOT precio
       VALUE LAMBDA (THISUNIT )
              (* (GET.VALUE THISUNIT 'precioSinIVA) 1.13))

MEMBER SLOT costeTransp
       VALUECLASS METHOD
       VALUE LAMBDA (THISUNIT )
              (* (GET.VALUE THISUNIT 'precioSinIVA) 0.05)

MEMBER SLOT eliminar
       VALUE RestaDeCantidad

MEMBER SLOT aniadir
       VALUE SumaACantidad
```

Instancias

- Instancia: es un objeto específico creado de acuerdo con los planes de construcción impuestos por su clase



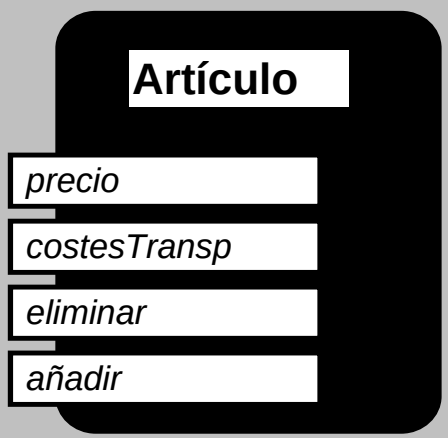

```
(CREATE.UNIT 'Articulo211   ;nombre de la unidad
   NIL NIL '(Articulo)      ;clase de la instancia
   NIL '((referencia 60021) ;informacion de campos
   (descripcion TVSony)
   (precioSinIVA 65.500)
   (cantidad 8)))
```

```
UNIT Articulo
MEMBERS articulo943 articulo211
```

```
UNIT articulo943
MEMBER-OF Articulo
```

Mecanismo de comunicacion entre objetos

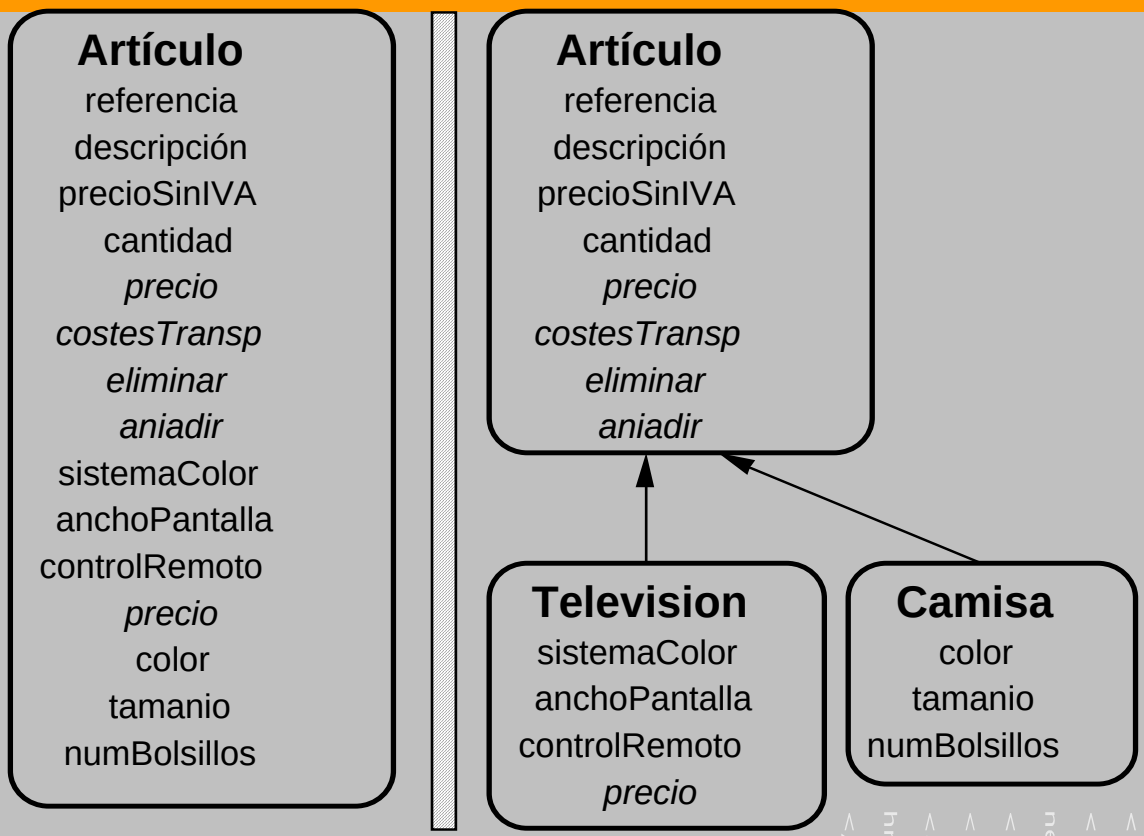
- Encapsulación: el objeto es una entidad independiente
  - sólo el objeto conoce su propia estructura
  - relación a través de un interfase: métodos



- Requerimiento para ejecutar un método: mensaje
  - parámetros necesarios:
    - objeto que recibe el mensaje
    - selector del método
    - argumentos

```
(UNITMSG ‘Articulo211          ;nombre del objeto
         ‘eliminar             ;nombre del mensaje
         1                     ;argumentos
         )
```

Herencia - especialización

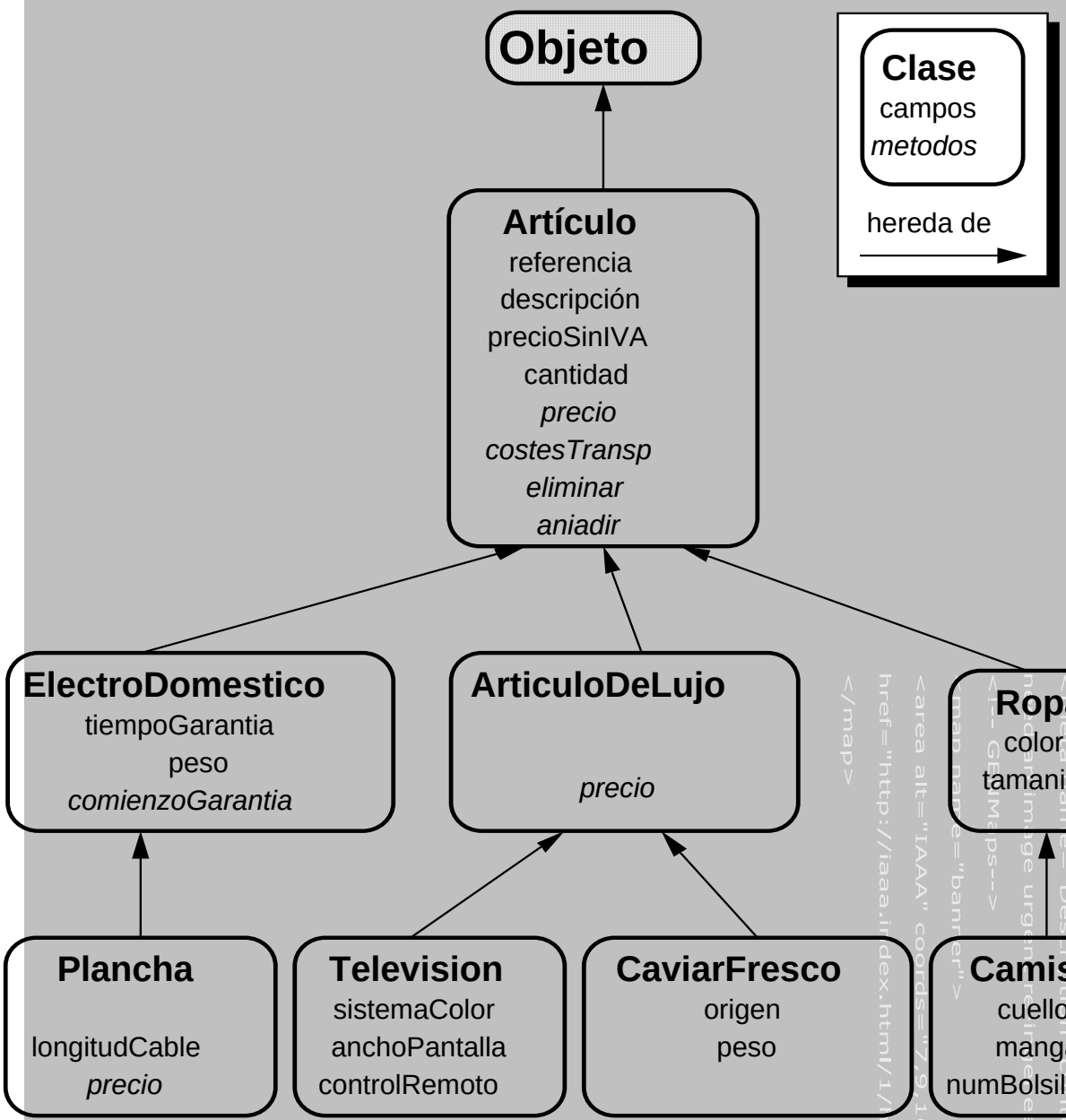


- Mecanismo para compartir conocimiento: herencia

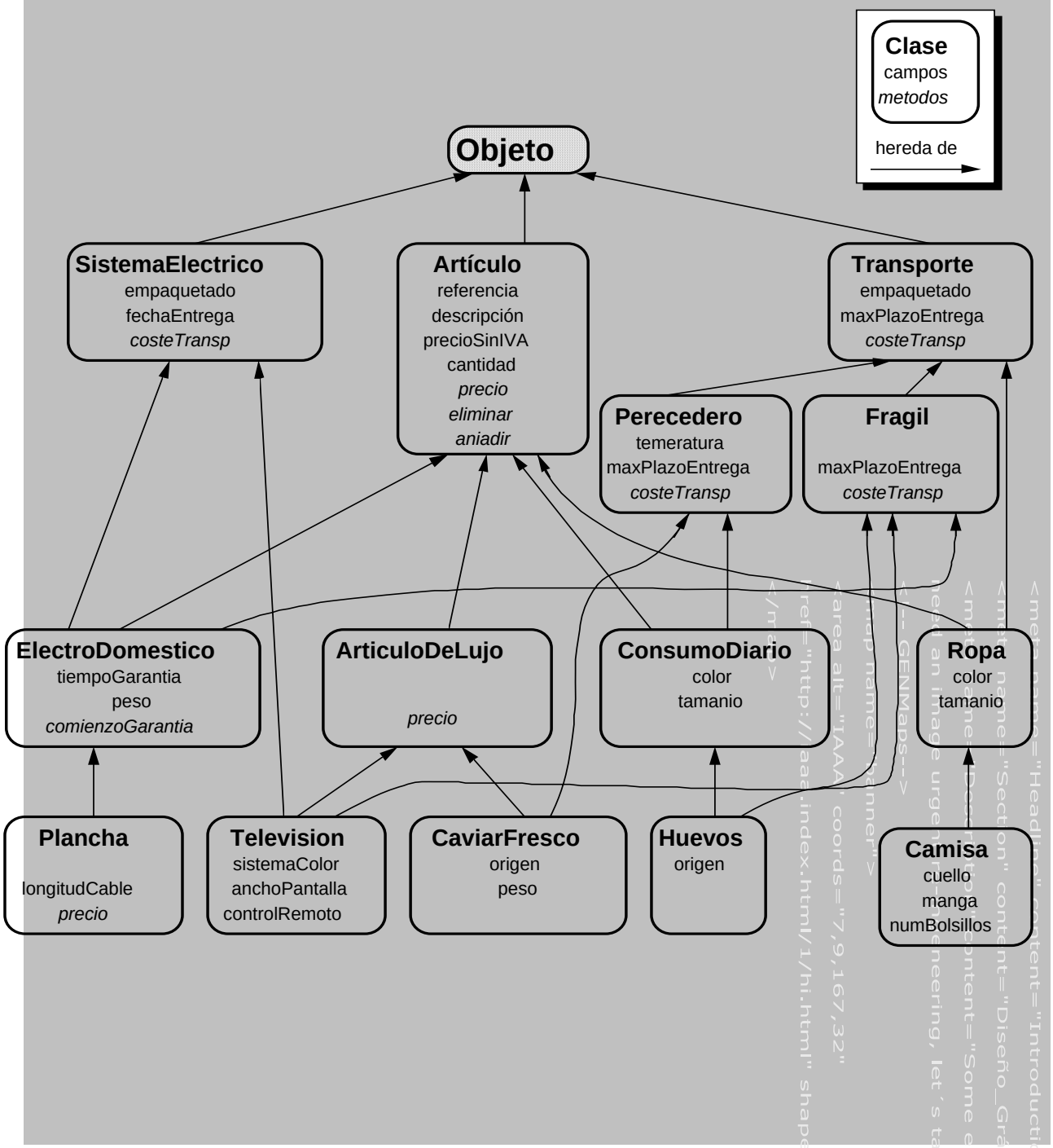
```
UNIT Television
SUPERCLASSES Articulo
SUBCLASSES articulo943

MEMBERSLOT sistemaColor                ;Adición
MEMBERSLOT anchoPantalla
MEMBERSLOT controlRemoto
MEMBERSLOT precio                      ;Sustitución
   VALUE LAMBDA (THISUNIT )
            (* (GET.VALUE THISUNIT 'precioSinIVA)
               1.33)
```

```
UNIT articulo943
MEMBER-OF Television
```

Grafo de herencia



Herencia múltiple



Herencia múltiple, problemas

- Cuando una clase tiene varias superclases directas

- Conflictos
  - mismo nombre distinta semántica - **fallo**
  - p.e.: precio precio con IVA
    precio coste del transporte
  - misma semántica distintos valores
  - se necesita: criterio para fijar la herencia, p.e.
    `maxPlazoEntrega` de la clase `Perecedero`
    `maxPlazoEntrega` de la clase `Fragil`

    el máximo de los dos

Herencia Múltiple en KEE

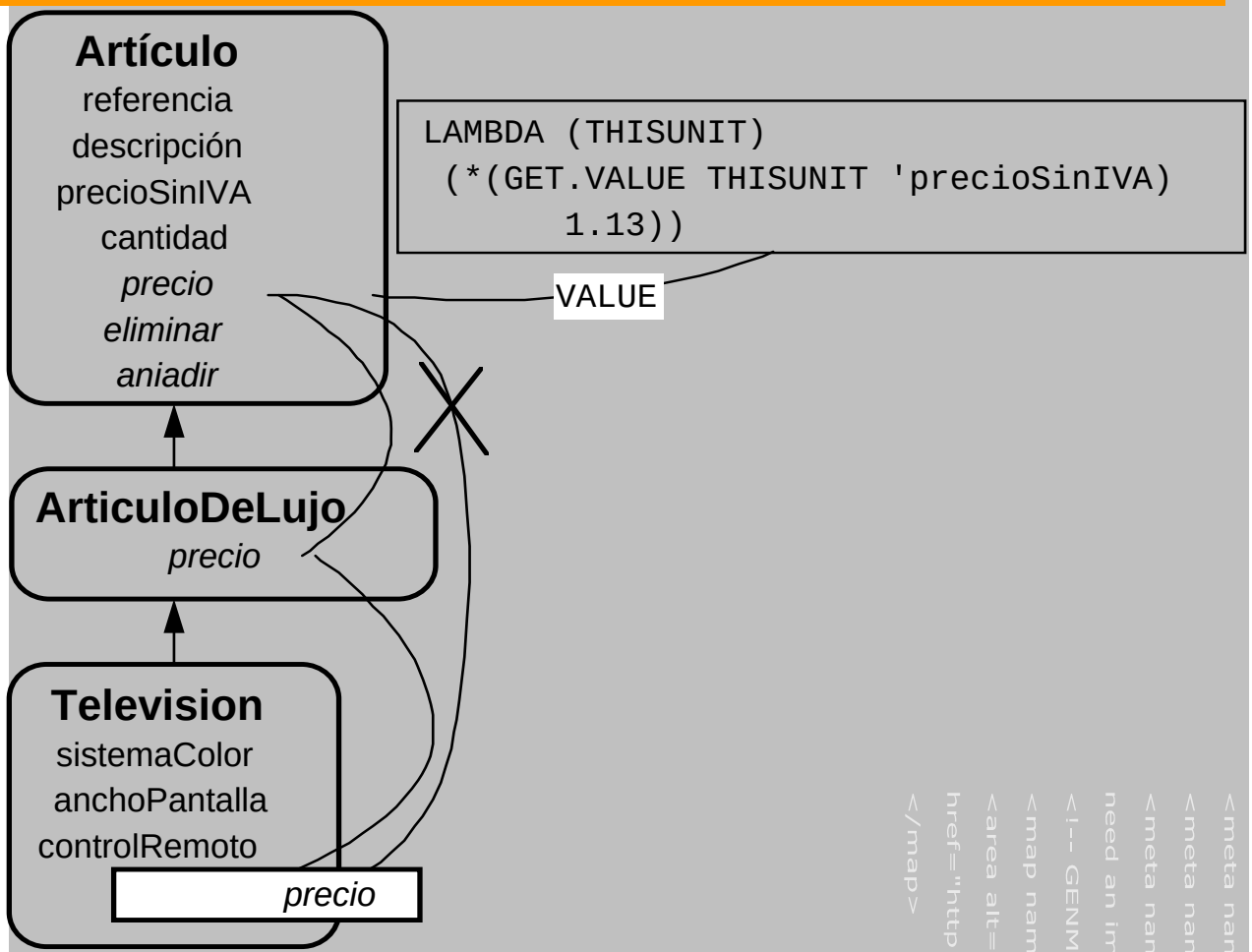
- En KEE la herencia se especifica a nivel de slot (es una propiedad del slot: FACET)
  - papel de herencia: INHERITANCE

- KEE proporciona distintos criterios para la herencia:
  - +++ el criterio por defecto: OVERRIDE.VALUES
    - 1 Si hay valor local, encubre al de los padres
    - 2 Sino, se toma el del primer padre que tiene valores

```
UNIT Huevos
SUPERCLASSES  ConsumoDiario  Fragil

MEMBERSLOT maxPlazoEntrega
     VALUECLASS NUMBER
     INHERITANCE NIL ;por defecto OVERRIDE.VALUES
```

    - 1 el valor local de Huevos
    - 2 sino, el valor de ConsumoDiario
    - 3 sino, el valor de Fragil
  - Otros criterios predefinidos:
    - MINIMUN el mínimo del local y padres
      p.e.: mínimo de Huevos, ConsumoDiario y Fragil
    - MAXIMUN, UNION ....

Herencia de métodos

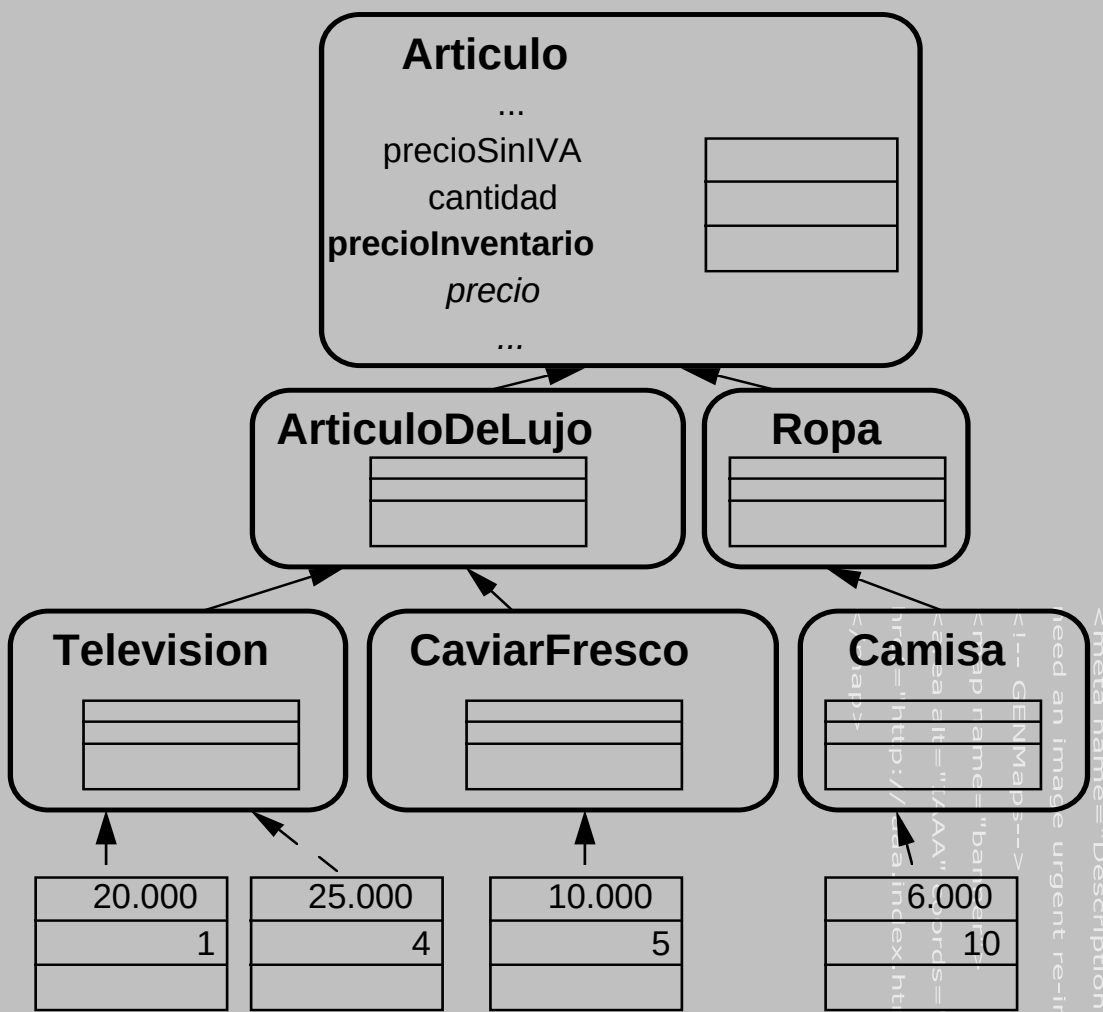


- Característica interesante de KEE (herencia especial de métodos):
  - Adición de "WRAPPERS (AFTER, BEFORE, WRAPPER)" a los métodos heredados

```
MEMBER SLOT precio
  INHERITANCE  METHOD
  VALUECLASS METHOD
  VALUE (WRAPPER
   (PROGN (FORMAT T
        "~&El articulo ~a paga impuesto de lujo" THISUNIT)
        (* WRAPPERBODY 1.33)))
```

Facets procedurales - Demons

- Objetivo: invocar una acción automáticamente al acceder/modificar el valor de un slot



Demons en KEE - Active Values

- Paso 1: Crear el valor activo como una unidad descendiente de ACTIVEVALUE

```
UNIT VAprecioInventario
MEMBER-OF  ACTIVEVALUE
                                {slots metodo heredados}
MEMBER SLOT  AVADD
MEMBER SLOT  AVREM
MEMBER SLOT  AVPUT
MEMBER SLOT  AVGET
       VALUECLASS     METHOD
       VALUE
 LAMBDA (THISUNIT )
 (cond ((NULL (UNIT.PARENTS THISUNIT 'MEMBER))
        (* (GET.VALUE THISUNIT 'cantidad)
           (GET.VALUE THISUNIT 'precioSinIVA)))
       (T (apply '+
           (MAPCAR
            #'(LAMBDA (elarticulo)
                (GET.VALUE elarticulo 'precioSinIVA))
                (append
                 (UNIT.CHILDREN THISUNIT 'MEMBER)
                 (UNIT.CHILDREN THISUNIT 'SUBCLASS))))))
```

- Paso 2: Asociar el valor activo al slot

```
UNIT Artículo
  ...
MEMBER SLOT precioInventario
       AVVALUE  VAprecioInventario
  ...
```