# Conocimiento

## Representación del Conocimiento

http://www.youtube.com/watch?v=IhVu2hxm07E&feature=related

# Conocimiento y razonamiento

- Las **personas** conocen cosas y realizan razonamientos de forma *automática*.
- ¿Y los **agentes artificiales**?
- Conocimiento y razonamiento en forma de **estructuras de datos** y **algoritmos**.
- Para que el conocimiento sea accesible para los ordenadores, se necesitan **sistemas basados en el conocimiento** (SBCs).

# Conocimiento y razonamiento

- En los SBCs se usan **lenguajes declarativos**:
  - Expresiones más cercanas a los leguajes humanos
- Los SBCs expresan el conocimiento en una forma que tanto los humanos como los ordenadores puedan entender.
- Esta parte de la asignatura analiza cómo expresar el conocimiento sobre el mundo real en una forma computacional.

## Ejemplo: Familia

```
/* BASE DE CONOCIMIENTO INICIAL */
/* Initial beliefs and rules */
yo(fred).                                   // Hecho

/* Relaciones paternales */
es_padreDe(fred,erik).                      // Hecho
es_padreDe(fred,tom).                       // Hecho
…
es_padreDe(john,fred).                      // Hecho

/* Relaciones maternales */
es_madreDe(liz,erik).                       // Hecho
es_madreDe(liz,tom).                        // Hecho
es_madreDe(sally,liz).                      // Hecho
…
/* Relaciones fraternales */
es_hermanoDe(X,Y) :-  es_padreDe(Z,X) & es_padreDe(Z,Y)        // Regla
                      & es_madreDe(W,X)
                      & not es_madreDe(W,Y) & not X=Y.
…
```

## Ejemplo: Familia

```
// Primero reglas para "X es_antepasadoDe Y"
es_antepasadoDe(X,Y) :- es_padreDe(X,Y).
es_antepasadoDe(X,Y) :- es_madreDe(X,Y).
es_antepasadoDe(X,Y) :- es_padreDe(X,Z) & es_antepasadoDe(Z,Y).
…
es_antepasadoIndirecto_de(X,Y) :- es_hermanoDe(X,Z) & es_antepasadoDe(Z,Y).

// Luego reglas para "X es_descendienteDe Y"
es_descendienteDe(X,Y) :- es_padreDe(Y,X).
es_descendienteDe(X,Y) :- es_madreDe(Y,X).
es_descendienteDe(X,Y) :- es_padreDe(Y,Z) & es_descendienteDe(X,Z).
…

// Por último las reglas para "X es_parienteDe Y"
es_parienteDe(X,Y) :-  (   es_antepasadoDe(X,Y)
                         |  es_antepasadoIndirecto_de(X,Y)
                         |  es_descendienteDe(X,Y))
                        & not X=Y.
```

## Ejemplo: Familia

/* Initial goals */

// Se introduce un objetivo *"pendiente"* para el agente como si fuera su MAIN

!start.

// Definimos la planificación ("*estilo imperativo*") para resolver el objetivo

```
+!start : yo(X)
     <-   .findall(Y,  es_parienteDe(Y, X), L);
          .print("Mis parientes son:", L);
          for (.member(Z, L)) {
               if (es_descendienteDe(Z, X)) {
                    .print(Z," es descendiente de ", X)};
               if (es_antepasadoIndirecto_de(Z, X) | es_antepasadoDe(Z, X)){
                    .print(Z, " es un antepasado de ", X)}
          }.
```

Y si queremos que el agente interaccione con el entorno ¿ ?

# Inferencia en lógica

- Se quieren conseguir algoritmos que puedan responder a preguntas expresadas en forma lógica.
- Tres grandes familias de algoritmos de inferencia:
  - **encadenamiento hacia delante** y sus aplicaciones en los **sistemas de producción**
  - **encadenamiento hacia atrás** y los sistemas de programación lógica
  - sistemas de demostración de teoremas basados en la resolución

## RAZONAMIENTO EN LÓGICA FORMAL

- Uso de **razonamiento deductivo**
  - Una aseveración será cierta si se puede demostrar su veracidad a partir de observaciones que se sabe que son ciertas
    $\longrightarrow$ todo lo que se infiere es verdadero
  - Pasar de conocimiento general a nuevo conocimiento específico

```
cuando hay gripe, tienes fiebre      Juan tiene gripe
Juan tiene gripe                     Juan tiene fiebre
------------------------------       ------------------------------
Juan tiene fiebre                    cuando hay gripe tienes fiebre
(Razonam. deductivo)   [LOGICA]      (Razonam. inductivo)   [APRENDIZAJE]
```

```
Juan tiene fiebre
cuando hay gripe, tienes fiebre
------------------------------
Juan tiene gripe
(Razonam. abductivo)      [RAZ. PROBABILISTICO]
```

$\longrightarrow$ Raz. inductivo y abductivo no aseguran que sus conclusiones sean siempre verdaderas

# Sistemas de producción

- La representación mediante formalismos lógicos es **declarativa** pero puede representar procedimientos.
- Se describen cuales son los pasos para resolver un problema como una **cadena de deducciones**.
- La representación se basa en dos elementos:
  - **hechos**: proposiciones o predicados
  - **reglas**: formulas condicionales

# Sistemas de producción

- Un problema queda definido por:
  - **Base de hechos**: que describen el problema concreto.
  - **Base de reglas**: que describen los mecanismos de razonamiento que permiten resolver problemas.
  - **Motor de inferencia**: que ejecuta las reglas y obtiene una cadena de razonamiento que soluciona el problema.

# Hechos: terminología

```
/* BASE DE CONOCIMIENTO INICIAL */
/* Initial beliefs and rules */
yo(fred).                          // Hecho

/* Relaciones paternales */
es_padreDe(fred,erik).             // Hecho
es_padreDe(fred,tom).              // Hecho
…
es_padreDe(john,fred).             // Hecho

/* Relaciones maternales */
es_madreDe(liz,erik).              // Hecho
es_madreDe(liz,tom).               // Hecho
es_madreDe(sally,liz).             // Hecho
…
```

- Base de hechos (BH):
  - Memoria de trabajo
  - Memoria a corto plazo
  - Aserciones

# Reglas: terminología

```
/* Relaciones fraternales */

es_hermanoDe(X,Y) :-
        es_padreDe(Z, X)        &
        es_padreDe(Z,Y)         &
        es_madreDe(W, X)        &
        not es_madreDe(W,Y)     &
        not X=Y.
…
```

**Si** **entonces**

- condiciones - acciones
- antecedentes - consecuentes
- premisas - conclusiones

- Base de reglas:
  - Base de conocimiento (BC)
  - Memoria a largo plazo
  - Implicaciones

# Motor de inferencia: terminología

- El motor de inferencia o mecanismo de control está compuesto de dos elementos:
  - **Interprete de reglas** o mecanismo de inferencia
    - Mecanismo de razonamiento que determina qué reglas de la BC se pueden aplicar para resolver el problema, y las aplica
  - **Estrategia de control** o estrategia de resolución de conflictos
- Función del motor de inferencia:
  - Ejecutar acciones para resolver el problema (objetivo) a partir de un conjunto inicial de hechos y eventualmente a través de una interacción con el usuario
  - La ejecución puede llevar a la deducción de nuevos hechos.

# Motor de inferencia

- Fases del ciclo básico:
  1. Detección (filtro): **Reglas pertinentes**
     - **Interprete de reglas:** Obtención, desde la BC, del conjunto de reglas aplicables a una situación determinada (estado) de la BH
       – formación del conjunto de conflictos
  2. Selección: **¿Qué regla?**
     - **Estrategia de control:** Resolución de conflictos
       – selección de la regla a aplicar

# Motor de inferencia

- Fases del ciclo básico:

3. Aplicación
   - Aplicación de la regla sobre una instancia de las variables: modificación de la memoria de trabajo
4. Vuelta al punto 1, o parada si el problema está resuelto
   - Si no se ha encontrado una solución y no hay reglas aplicables: **fracaso**.

# 1. Detección

- Construcción del conjunto de reglas aplicables
- El intérprete de reglas realiza los cálculos necesarios para obtener las instancias que son posibles en cada estado de resolución del problema (**comparación** o *matching*).
- **Una regla se puede instanciar más de una vez**, caso de existir variables que lo permitan.

# 2. Selección

- Las reglas son o no aplicadas dependiendo de la **estrategia de control**:
  - –estrategia fija
  - –estrategia dinámica prefijada
  - –estrategia guiada por meta-reglas
- Selección de la "mejor" instancia
- Posible combinación de criterios

# 2. Selección

## Ejemplos de estrategia de control:

- 1ª regla por orden en la base de conocimiento
- la regla más/menos utilizada
- la regla más específica (con más literales)
- la regla más general (con más variables)
- la regla que tenga el grado de certeza más alto

# Hechos: terminología

```
/* BASE DE CONOCIMIENTO INICIAL */
/* Initial beliefs and rules */
yo(fred).

/* Relaciones paternales */
es_padreDe(fred,erik).          // 1ª Regla
es_padreDe(fred,tom).           // 2ª Regla
…
es_padreDe(john,fred).          // Nª Regla

/* Relaciones maternales */
es_madreDe(liz,erik).           // 1ª Regla
es_madreDe(liz,tom).            // 2ª Regla
es_madreDe(sally,liz).          // 3ª Regla
…
```

# Hechos: terminología

```
/* BASE DE CONOCIMIENTO INICIAL */
/* Initial beliefs and rules */
yo(fred).                          // Otra Interpretación

/* Relaciones paternales */
es_padreDe(fred,erik).             // Nª Regla
es_padreDe(fred,tom).              // N-1ª Regla
…
es_padreDe(john,fred).             // 1ª Regla

/* Relaciones maternales */
es_madreDe(liz,erik).              // Nª Regla
es_madreDe(liz,tom).               // N-1ª Regla
es_madreDe(sally,liz).             // N-2ª Regla
…
```

# 2. Selección

Otros ejemplos de estrategia de control:

- la instancia que satisfaga los hechos:
  - más prioritarios
  - más antiguos (instancia más antigua)
  - más nuevos (instancia más reciente)
- aplicación de todas las reglas (sólo si se quieren todas las soluciones posibles)
- la regla usada más recientemente
- meta-reglas, que indican dinámicamente como seleccionar las reglas a aplicar

## Ejemplo: Familia

```
// Primero reglas para "X es_antepasadoDe Y"
es_antepasadoDe(X,Y) :- es_padreDe(X,Y).          // 2ª Regla - 1ª Menos general
es_antepasadoDe(X,Y) :- es_madreDe(X,Y).          // 3ª Regla - 2ª Menos general
es_antepasadoDe(X,Y) :- es_padreDe(X,Z) &
                        es_antepasadoDe(Z,Y). // 1ª Regla - Más general
…
es_antepasadoIndirecto_de(X,Y) :- es_hermanoDe(X,Z) & es_antepasadoDe(Z,Y).
```

```
// Luego reglas para "X es_descendienteDe Y"
es_descendienteDe(X,Y) :- es_padreDe(Y,X).
es_descendienteDe(X,Y) :- es_madreDe(Y,X).
es_descendienteDe(X,Y) :- es_padreDe(Y,Z) & es_descendienteDe(X,Z).
…

// Por último las reglas para "X es_parienteDe Y"
es_parienteDe(X,Y) :-  (   es_antepasadoDe(X,Y)
                         | es_antepasadoIndirecto_de(X,Y)
                         | es_descendienteDe(X,Y))
                        & not X=Y.
```

# 3. Aplicación

- Ejecución de la regla ⇒
  - Modificación de la base de hechos (en el razonamiento hacia delante)
  - Nuevos cálculos, nuevas acciones, preguntas al usuario
  - Nuevos sub-objetivos (en el razonamiento hacia atrás)
- Propagación de las instancias
- Propagación del grado de certeza
- El proceso de deducción acaba cuando:
  - se encuentra la conclusión (el objetivo) buscado ⇒ éxito
  - no queda ninguna regla aplicable ⇒ éxito? / fracaso?

# Tipos de razonamiento

- Deductivo, progresivo, **encadenamiento hacia delante**, dirigido por hechos
  - evidencias, síntomas, datos ⇒ conclusiones
- Inductivo, regresivo, **encadenamiento hacia atrás**, dirigido por objetivos
  - conclusiones ⇒ datos, evidencias, síntomas
- Mixto, encadenamiento híbrido

# Encadenamiento hacia delante

- Basado en *modus ponens*: A, A⇒B |- B
- La base de hechos (BH) se inicializa con los hechos conocidos inicialmente.
- Se obtienen las consecuencias derivables de la BH:
  - se comparan los hechos de la BH con la parte izquierda de las reglas; se seleccionan las reglas aplicables: las que tienen antecedentes conocidos (que están en la BH);
  - las nuevas conclusiones de las reglas aplicadas se añaden a la BH (hay que decidir cómo);
  - se itera hasta encontrar una condición de finalización.

# Encadenamiento hacia delante

- Problemas:
  - No focaliza en el objetivo
  - Explosión combinatoria
- Ventajas:
  - Deducción intuitiva
  - Facilita la formalización del conocimiento al hacer un uso natural del mismo
- Ejemplo de lenguaje: CLIPS

# Encadenamiento hacia atrás

- Basado en el *método inductivo*:
  - –guiado por un objetivo que es la conclusión que se trata de validar reconstruyendo la cadena de razonamiento en orden inverso.
- Cada paso implica nuevos sub-objetivos: *hipótesis* que han de validarse.

# Encadenamiento hacia atrás

- Funcionamiento:
  - se inicializa la BH con un conjunto inicial de hechos;
  - se inicializa el conjunto de hipótesis (CH) con los objetivos a verificar;
  - mientras existan hipótesis a validar en CH se escoge una de ellas y se valida:
    - se comparan los hechos de la BH y la parte derecha de las reglas con las hipótesis;
    - si una hipótesis está en BH eliminarla de CH;
    - si no: buscar reglas que tengan como conclusión la hipótesis; seleccionar una y añadir las premisas a CH.

# Encadenamiento hacia atrás

- El encadenamiento hacia atrás es un tipo de **razonamiento dirigido por el objetivo**.
- Sólo se considera lo necesario para la resolución del problema.
- El proceso de resolución consiste en la exploración de un árbol.
- Ejemplo de lenguaje: Prolog
  - W. F. Clocksin y C. S. Mellish . *Programming in Prolog: Using the ISO Standard*. Springer, 2003 (primera edición de 1981).

# Encadenamiento híbrido

- Partes de la cadena de razonamiento que conduce de los hechos a los objetivos se construyen **deductivamente** y otras partes **inductivamente**: exploración bi-direccional
- El cambio de estrategia suele llevarse a cabo a través de **meta-reglas**. Ejemplos:
  - función del número de estados iniciales y finales
  - función de la dirección de mayor ramificación
  - función de la necesidad de justificar el proceso de razonamiento
- Se evita la explosión combinatoria del razonamiento deductivo.

# Factores para decidir el sentido del encadenamiento

- Número de estados iniciales y finales
  - Preferible del conjunto más pequeño hacia el más grande
- Factor de ramificación
  - Preferible en el sentido del factor más pequeño
- Necesidad de justificar el proceso de razonamiento
  - Preferible el sentido de razonamiento habitual del usuario

# Comparación (matching)

- Es más complicada en el razonamiento hacia delante.
  - Si las condiciones de una regla se cumplen, una vez aplicada se puede entrar en un ciclo.
- Existen mecanismos eficientes de comparación y selección que evitan repasar todas las reglas de la BC:
  - OPS-5 usa el algoritmo RETE
  - Prolog indexa las cláusulas según los predicados

## Ejemplo: Familia

```
/* BASE DE CONOCIMIENTO INICIAL */
/* Initial beliefs and rules */
yo(fred).                                   // Hecho

/* Relaciones paternales */
es_padreDe(fred,erik).                      // Hecho
es_padreDe(fred,tom).                       // Hecho
…
es_padreDe(john,fred).                      // Hecho

/* Relaciones maternales */
es_madreDe(liz,erik).                       // Hecho
es_madreDe(liz,tom).                        // Hecho
es_madreDe(sally,liz).                      // Hecho
…
/* Relaciones fraternales */
es_hermanoDe(X,Y) :-  es_padreDe(Z,X) & es_padreDe(Z,Y)        // Regla
                      & es_madreDe(W,X)
                      & not es_madreDe(W,Y) & not X=Y.
…
```

## Ejemplo: Familia

```
// Primero reglas para "X es_antepasadoDe Y"
es_antepasadoDe(X,Y) :- es_padreDe(X,Y).
es_antepasadoDe(X,Y) :- es_madreDe(X,Y).
es_antepasadoDe(X,Y) :- es_padreDe(X,Z) & es_antepasadoDe(Z,Y).
…
es_antepasadoIndirecto_de(X,Y) :- es_hermanoDe(X,Z) & es_antepasadoDe(Z,Y).

// Luego reglas para "X es_descendienteDe Y"
es_descendienteDe(X,Y) :- es_padreDe(Y,X).
es_descendienteDe(X,Y) :- es_madreDe(Y,X).
es_descendienteDe(X,Y) :- es_padreDe(Y,Z) & es_descendienteDe(X,Z).
…

// Por último las reglas para "X es_parienteDe Y"
es_parienteDe(X,Y) :-  (   es_antepasadoDe(X,Y)
                         |  es_antepasadoIndirecto_de(X,Y)
                         |  es_descendienteDe(X,Y))
                        & not X=Y.
```

## Ejemplo: Familia

/* Initial goals */

// Se introduce un objetivo *"pendiente"* para el agente como si fuera su MAIN

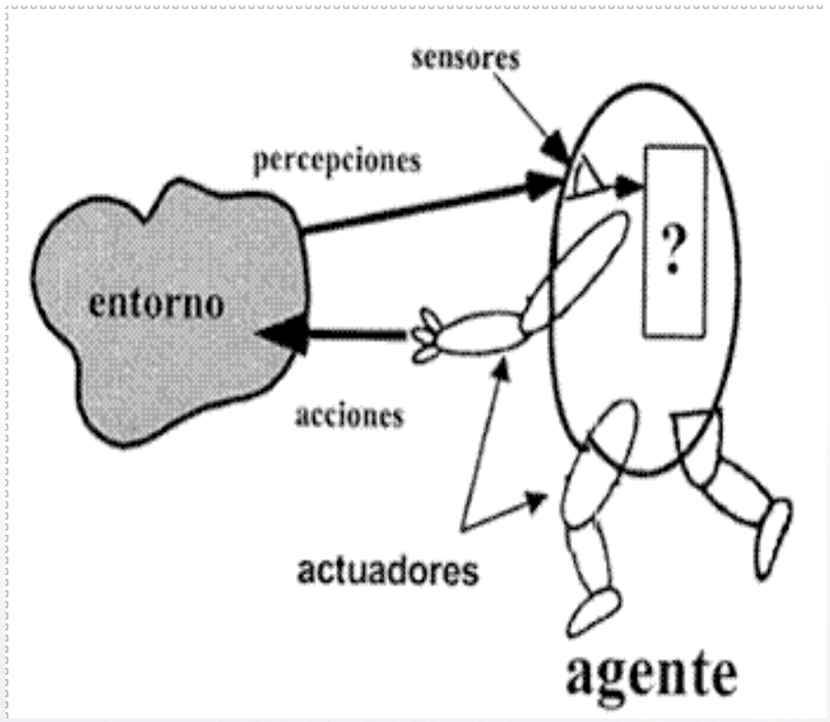
!start.

// Definimos la planificación ("*estilo imperativo*") para resolver el objetivo

```
+!start : yo(X)
     <-   .findall(Y,  es_parienteDe(Y, X), L);
          .print("Mis parientes son:", L);
          for (.member(Z, L)) {
               if (es_descendienteDe(Z, X)) {
                    .print(Z," es descendiente de ", X)};
               if (es_antepasadoIndirecto_de(Z, X) | es_antepasadoDe(Z, X)){
                    .print(Z, " es un antepasado de ", X)}
          }.
```

Y si queremos que el agente interaccione con el entorno ¿ ?

**Los Agentes pueden ser robots-humanos, softbots, dispositivos como el termostato y muchos otros.**




Esquema


Siri

# Ejemplo: Aspiradora

```
// Implementation of the example described in chapter 2
// of the Jason's manual

MAS mars {

    infrastructure: Centralised

    environment: MarsEnv

    agents: r1; r2;
}


// mars robot 2
+garbage(r2) : true <- burn(garb).
```

# Ejemplo: Aspiradora

```
// mars robot 1

/* LOGIC: Initial beliefs */

at(P) :- pos(P,X,Y) & pos(r1,X,Y).

/* LOGIC: Initial goal */

!check(slots).

/* LOGIC: Plans */

+!check(slots) : not garbage(r1)
    <-  next(slot);
        !!check(slots).
+!check(slots).


+garbage(r1) :
    not .desire(carry_to(r2))
    <-  !carry_to(r2).
```

```
+!carry_to(R) <-
    // remember where to go back
    ?pos(r1,X,Y);
    -+pos(last,X,Y);
    !take(garb,R);
    // carry garbage to r2
    !at(last);
    // goes back and continue to check
    !!check(slots).

+!take(S,L) : true
    <- !ensure_pick(S);
       !at(L);
       drop(S). // On the Environment

+!ensure_pick(S) : garbage(r1)
    <- pick(garb); // On the Environment
       !ensure_pick(S).
+!ensure_pick(_).

+!at(L) : at(L).
+!at(L) <-
    ?pos(L,X,Y);
    move_towards(X,Y); // On the Environment
    !at(L).
```

# Ejemplo: Aspiradora

```
. . .
public class MarsEnv extends Environment {

    public static final int GSize = 8; // grid size
    public static final int GARB  = 16; // garbage code in grid model

    public static final Term    ns = Literal.parseLiteral("next(slot)");
    public static final Term    pg = Literal.parseLiteral("pick(garb)");
    public static final Term    dg = Literal.parseLiteral(“drop(garb)");
    . . .
    @Override
    public boolean executeAction(String ag, Structure action) {
        logger.info(ag+" doing: "+ action);
        try {
            if (action.equals(ns)) {
                model.nextSlot();
            } else if (action.getFunctor().equals("move_towards")) {
                int x = (int)((NumberTerm)action.getTerm(0)).solve();
                int y = (int)((NumberTerm)action.getTerm(1)).solve();
                model.moveTowards(x,y);
            } else if (action.equals(pg)) {
. . .
```

## Ejemplo: Aspiradora

```
. . .
        void moveTowards(int x, int y) throws Exception {
            Location r1 = getAgPos(0);
            if (r1.x < x)
                r1.x++;
            else if (r1.x > x)
                r1.x--;
            if (r1.y < y)
                r1.y++;
            else if (r1.y > y)
                r1.y--;
            setAgPos(0, r1);
            setAgPos(1, getAgPos(1)); // just to draw it in the view
        }
. . .
        void dropGarb() {
            if (r1HasGarb) {
                r1HasGarb = false;
                add(GARB, getAgPos(0));
            }
        }
```