PRÁCTICA 2 - SISTEMAS BASADOS EN REGLAS

INFORME

Sistemas Inteligentes
3º Grupo 3.1
Alfredo Marquina Meseguer
Práctica 2/ 17-12-2023
Universidad de Murcia

Índice

1 Encadenamiento Hacia Atrás

A la hora de crea un procedimiento para ejecutar un encadenamiento hacia atrás lo que se ha pensado a sido una estructura de datos que simule la red de inferencia mediante objetos y punteros.

La idea es tener una serie de objetos que implemente la interfaz “Elemento” que tenga una función “evaluar” que nos permita calcular el factor de certeza de las clases que implementen esta interfaz: Hechos (son las hipótesis) y Reglas.

Por un lado, tenemos las clases de hechos donde existiría una distinción entre hechos simples, AND y OR. Cada una tendría una lista de precondiciones, un factor de certeza y un nombre.

En caso de los hechos simples se podría asignar directamente un factor de certeza o se puede calcular a partir de las reglas que implican dicho hecho. Si se careciera tanto de factor de certeza como de antecedentes desde lo que calcularlo, no encontraríamos con comportamiento no definido y se podría asumir cero, aunque lanzar un error por encontrarse en una red de inferencia mal formada. En caso de existir más de un antecedente se aplicaría el caso 2 a la hora de calcular el factor de certeza.

Sin embargo, en el caso de los hechos AND y OR, o hechos compuestos, este factor de certeza se calcularía siempre, no es asignable de forma directa y, de guardarse, sería unicamente para evitar que se calcularan los mismos valores varias veces. La forma de calcular los factores de certeza para estos hechos es aplicar el caso 1 entre los hechos Simples que los forman.

Por el otro lado, las reglas simplemente tienen un antecedente, un consecuente, un factor de certeza y un nombre. El consecuente no se guarda en la regla, sino que se añade a los antecedentes de este en el constructor. En caso de faltar cualquiera de estas propiedades el cálculo fallaría, menos el nombre que funciona a modo de identificador.

El Pseudo-código sería el siguiente:

```
interfaz Elemento {  real verificar(); }

constante real SIN_CALCULAR = // Cualquier valor que no utilicemos

clase Hecho impl Elemento {
    lista<Regla> antecedentes
    real factorCerteza
    cadena nombre
    // Constructor
  real verificar(){
    si factorCerteza == SIN_CALCULAR{
      //Utilizo iteradores para que sea más conciso
      factorCerteza = caso2(antecedentes.itr().map(t→ t.verificar()).collect())
    }
       return factorCerteza;
```

```
clase HechoAND impl Elemento{
  lista<T impl Elemento> antecedentes

    real verificar(){ // Caso 1
    return antecedente.itr().map(m → m.verificar()).max();
    }
}
clase HechoOR impl Elemento{
  lista<T impl Elemento> antecedentes

  real verificar(){// Caso 1
    return antecedente.itr().map(m → m.verificar()).min();
  }
}
clase Regla impl Elemento{
  <T impl Elemento> antecedente
  real factorCerteza
  cadena nombre

  real verificar() { // Caso 3
    return factorCerteza * max(0, antecedente.verificar())
  }
}
```

Dado este código y teniendo la red de inferencia completamente inicializada, solo sería necesario llamar a la función verificar del hecho objetivo para que se realizara todo el proceso. Además, como se ha mencionado anteriormente, se podría editar la función verificar para guardar el valor del factor de certeza calculado y evitar cálculos repetidos.

En el pseudo-código, durante “verifcar” de Hecho llama a una función “caso2” la implementación de esta sería la siguiente:

Pseudo-código

```
real caso2Simple(double valor1, double valor2){
    si valor1 ⩾ 0 && valor2 ⩾ 0 :
        return valor1 + valor2 * (1 - valor1)

    si valor1 ⩽ 0 && valor2 ⩽ 0 :
        return valor1 + valor2 * (1 + valor1)

  return (valor1 + valor2) / (1.0 - min(absoluto(valor1), absoluto(valor2)))
}
```

Pseudo-código

```
real caso2(list<real> antecedentes){
    si antecedentes.vacia()
    {
        imprimir("Error, comportamiento incierto en caso2")
        return 0
    }

    si antecedentes.longitud() == 1 :
        return  antecedentes[0]

    si antecedentes.longitud() == 2 :
        return  caso2Simple(antecedentes[0], antecedentes[1])

    double valor = reglas[0].verificar();
  para (entero i = 1; i < antecedentes.tamanno(); ++i)
  {
    valor = caso2Simple(valor,reglas[i].verificar())
  }
  return valor
}
```

2 PRUEBA 3

a) La Formalizamos la información a hechos:

1 antiguedad23: “El conductor tiene entre 2 y tres años de antigüedad.”

2 antiguedadmas3: ”El conductor tiene más de tres años de antigüedad.”

3 experimentado: “El conductor es experimentado.”

4 conducir23horas: “El conductor ha conducido entre dos y tres horas.”

5 conducirmas3horas: “El conductor ha conducido más de tres horas.”

6 cansado: “El conductor está cansado.”

7 noSolo: “El conductor no está solo”

8 causante: “El conductor es el causante del accidente”

9 joven: “El conductor es joven”

10 alcohol: “El conductor ha consumido alcohol”

b) REGLAS:

1 R1: Si antiguedad23 Entonces experimentado, FC=0.5

2 R2: Si antiguedadmas3 Entonces experimentado, FC=0.9

3 R3: Si conducir23horas Entonces cansado, FC=0.5

4 R4: Si conducirmas3horas Entonces cansado, FC=1

5 R5: Si experimentado y noSolo Entonces causante, FC=-0.5

6 R6: Si cansado Entonces causante, FC=0.5

7 R7: Si joven y alcohol Entonces causante, FC=0.7

c) HECHOS

1 FC(antiguedad23)=1

2 FC(antiguedadmas3)=-1

3 FC(conducirmas3horas)=1

4 FC(conducir23horas)=-1

5 FC(noSolo)=-1

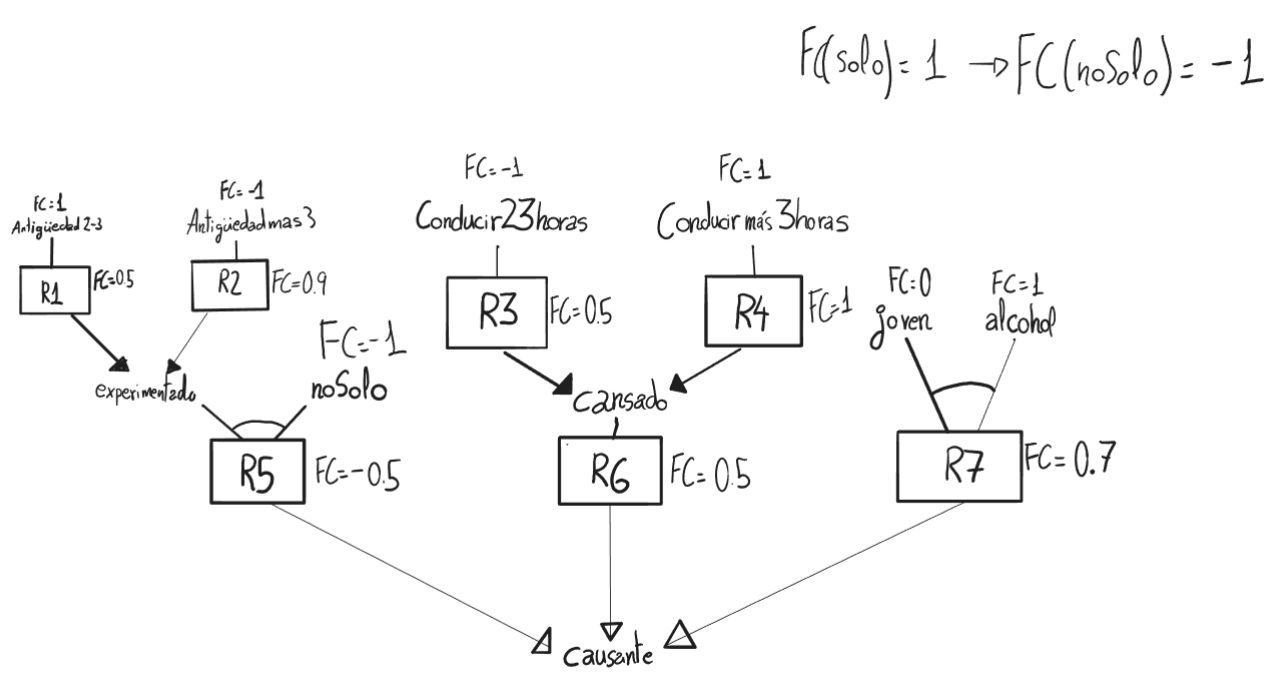
a) Se menciona FC(solo)=1, pero esa no se encuentra en la formalización, solo su complementaria.

6 FC(alcohol)=0.5

7 FC(joven)=0

a) No es mencionada, se pone cero porque no sabemos nada sobre ella.

d) Red de inferencia:



2.1 Base de Conocimiento

7

R1: Si antiguedad23 Entonces experimentado, FC=0.5

R2: Si antiguedadmas3 Entonces experimentado, FC=0.9

R3: Si conducir23horas Entonces cansado, FC=0.5

R4: Si conducirmas3horas Entonces cansado, FC=1

R5: Si experimentado y noSolo Entonces causante, FC=-0.5

R6: Si cansado Entonces causante, FC=0.5

R7: Si joven y alcohol Entonces causante, FC=0.7

2.2 BASE de Hechos

7

antiguedad23, FC=1

antiguedadmas3, FC=-1

conducirmas3horas, FC=1

conducir23horas, FC=-1

noSolo, FC=-1

alcohol, FC=0.5

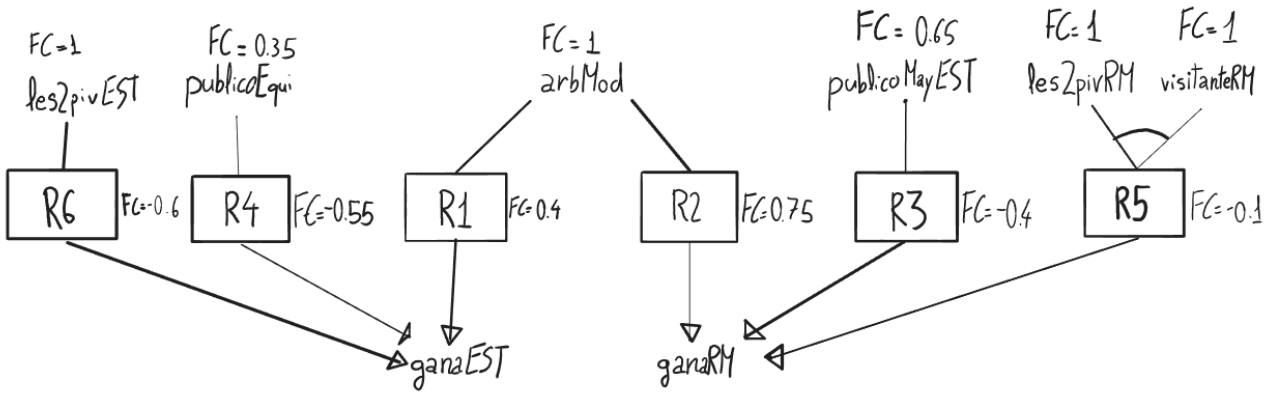
joven, FC=0

Objetivo

causante

3 PRUEBA 2

Su red de inferencia es:



3.1 BASE DE CONOCIMIENTO

6

R1: Si arbMod Entonces ganaEST, FC=0.4

R2: Si arbMod Entonces ganaRM, FC=0.75

R3: Si publicoMayEST Entonces ganaRM, FC=-0.4

R4: Si publicoEqui Entonces ganaEST, FC=-0.55

R5: Si les2pivRM y visitanteRM Entonces ganaRM, FC=-0.1

R6: Si les2pivEST Entonces ganaEST, FC=-0.6

3.2 BASE DE HECHOS

7

localEST, FC=1

visitanteRM, FC=1

arbMod, FC=1

publicoMayEST, FC=0.65

publicoEqui, FC=0.35

les2pivEST, FC=1

```
les2pivRM, FC=1

Objetivos

ganaEST, ganaRM
```

4 PRUEBA A

Akaikos es un granjero pobre que vive cerca de Atenas. Le encanta dar paseos, pero últimamente ha tenido muy mala suerte y quiere saber si es un buen día para dar un paseo:

1. Si la economía es mala y los caminos no se encuentran protegidos por el ejercito entonces es posible que se cruce con bandidos (0,3).
2. Si se encuentra con bandidos, como suele llevar dinero encima para estas ocasiones, probablemente no saldrá herido (0,5).
3. Si Zeus un nuevo amante o su esposa, Hera, no le descubre, Zeus estará feliz (0,9).
4. Si Zeus está feliz, lo más probable es que no hiera a Akaikos con un rayo (1).
5. Si no va a ser herido y hace un buen día entonces lo más probable es que salga a pasear (0,9).

Últimamente la economía está un poco mal por la reciente guerra (0,8), los caminos no están protegidos del todo (0,6), Zeus a tenido éxito con su nuevo amante (1), su esposa está feliz porque no lo ha descubierto (1) y hace un día esplendido (1). ¿Va a salir a pasear?

4.1 FORMALIZACIÓN

Construimos la signatura:

∑ = {economíaMal, caminosNoProtegidos, bandidos, noSerHerido, fracasoAmoroso, broncaHera, ZeusEnfadado, buenDia, pasear} donde:

- economíaMal = “economía es mala”
- caminosNoProtegidos = “caminos no se encuentran protegidos”
- bandidos = “Akaikos se encuentra con bandidos”
- noSerHerido = “no va a ser herido”
- exitoAmoroso = “Zeus consigue un nuevo amante”
- HeraFeliz = “Hera está feliz porque no sabe de la amante”
- ZeusFeliz = “Zeus está feliz”

- buenDia = “Hace un buen día”
- pasear = “Akaikos sale a pasear”

4.1.1 Base De Conocimiento

Las reglas son:

- Si economíaMal y caminosNoProtegidos Entonces bandidos, FC=0.3
- Si bandidos Entonces noSerHerido, FC=0.5
- Si exitoAmoroso o HeraFeliz Entonces ZeusFeliz, FC=0.9
- Si ZeusFeliz Entonces noSerHerido, FC=1
- Si noSerHerido y buenDia Entonces pasear, FC=0.9

Se traducen en el fichero:

```
5
R1: Si economiaMal y caminosNoProtegidos Entonces bandidos, FC=0.3
R2: Si bandidos Entonces noSerHerido, FC=0.5
R3: Si exitoAmoroso o HeraFeliz Entonces ZeusFeliz, FC=0.9
R4: Si ZeusFeliz Entonces noSerHerido, FC=1
R5: Si noSerHerido y buenDia Entonces pasear, FC=0.9
```

4.1.2 Base De Hechos

Últimamente la economía está un poco mal por la reciente guerra (0,8), los caminos no están protegidos del todo (0,6), Zeus a tenido éxito con su nuevo amante (1), su mujer lo ha descubierto y le ha echado un bronca suave (0,3) y hace un día esplendido (1). ¿Va a salir a pasear?

Los hechos son:

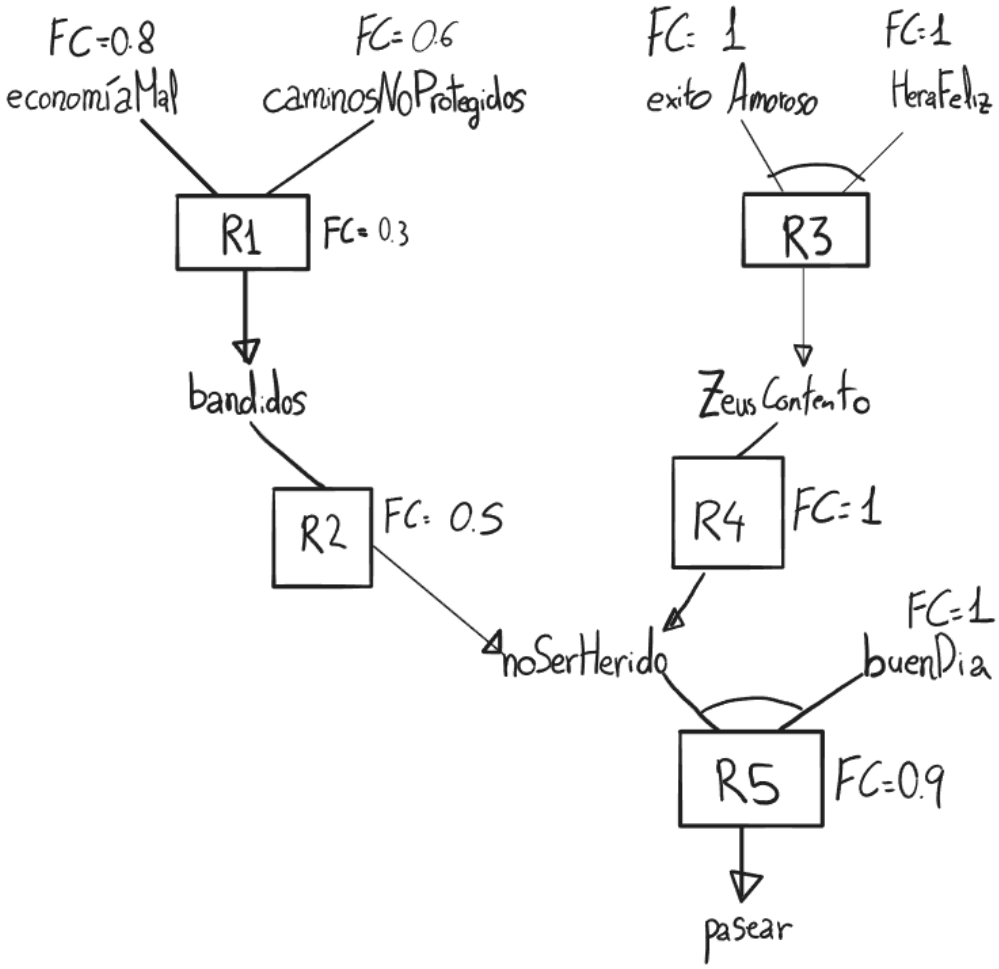
- FC(economíaMal)=0,8
- FC(caminosNoProtegidos)=0,6
- FC(exitoAmoroso)=1
- FC(HeraFeliz)=1
- FC(buenDia)=1

Que considerando que su objetivo es pasear, se traducen en el fichero con contenido:

```
5
economiaMal, FC=0.8
caminosNoProtegidos, FC=0.6
exitoAmoroso, FC=1
HeraFeliz, FC=1
buenDia, FC=1
Objetivo
pasear
```

4.1.3 Red de Inferencia

La red de inferencia es la siguiente



5 Notas Implementación

En esta sección quiero explicar un par de cuestiones de la implementación.

Durante la implementación del código se implementó primero el motor de resolución, después el objeto logger que genera el log y finalmente el programa de parseo de ficheros y escritura en fichero de la salida, así como un log del parseo implementado por las dificultades encontradas durante esta última parte.

La primera cuestión a mencionar es que la salida por consola se trata de este log del parseo y que el log pedido durante la práctica se escribirá en el fichero de salida especificado.

La segunda es que el programa ha sido desarrollado en Linux por lo que se añade el fichero Makefile y el programa generado, por si fueran de algún uso.

La tercera es que durante la implementación se interpretó que crear un log del proceso de inferencia como en el de la diapositiva 15, se refería exactamente igual a ese y no un log como la diapositiva 15 que también tiene log. Esta cuestión explica la gran complejidad del código, ya que el motor sin dicho log se reduce al comentado en la parte de pseudo-código y la inicialización mediante lectura de ficheros.

Finalmente, comentar que la existencia de un menor número de ficheros de prueba es porque el programa, está implementado para que cuando en la prueba dos pida cual equipo va a ganar, ejecute el árbol para ambos objetivos y devuelva el mayor.