# OTTER: Tratamiento de la igualdad

Francisco J. Martín Mateos
José A. Alonso Jiménez

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

# Axiomas de igualdad

- Demostrar que si Francisco es igual a Curro y a Paco, entonces Curro y Paco son iguales
- Formalización en OTTER

**francisco-a.in**

```
set(binary_res).

list(sos).
 francisco = curro.
 francisco = paco.
 paco != curro.
end_of_list.
```

- OTTER: `Search stopped because sos empty.`

# Axiomas de igualdad

- Formalización en OTTER con axiomas de igualdad

## francisco-b.in

```
set(binary_res).

list(sos).
 x = x.                          % Reflexividad
 x != y | y = x.                 % Simetría
 x != y | y != z | x = z.        % Transisitividad
 francisco = curro.
 francisco = paco.
 paco != curro.
end_of_list.
```

# Axiomas de igualdad

## Prueba obtenida

```
 2 [] x != y | y = x.
 3 [] x != y | y != z | x = z.
 4 [] francisco = curro.
 5 [] francisco = paco.
 6 [] paco != curro.
 8 [binary,2.1,4.1] curro = francisco.
 9 [binary,2.2,6.1] curro != paco.
10 [binary,3.1,8.1] francisco != x | curro = x.
24 [binary,10.1,5.1] curro = paco.
25 [binary,24.1,9.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 10 | 35 | 18 | 10 | 0 |

# Axiomas de igualdad

- Formalización en OTTER con soporte y resolución UR

## francisco-c.in

```
set(ur_res).

list(usable).
 x = x.                          % Reflexividad
 x != y | y = x.                 % Simetría
 x != y | y != z | x = z.        % Transisitividad
 francisco = curro.
 francisco = paco.
end_of_list.

list(sos).
 paco != curro.
end_of_list.
```

# Axiomas de igualdad

## Prueba obtenida

```
 2 [] x != y | y = x.
 3 [] x != y | y != z | x = z.
 4 [] francisco = curro.
 5 [] francisco = paco.
 6 [] paco != curro.
 7 [ur,6,3,4] paco != francisco.
 9 [ur,7,2] francisco != paco.
10 [binary,9.1,5.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 2 | 7 | 3 | 4 | 0 |

# Axiomas de sustitución funcionales

- Demostrar que si la opuesta de la derecha es la izquierda y la opuesta de la izquierda es la derecha, entonces la opuesta a la opuesta de la derecha es la derecha
- Formalización en OTTER con axiomas de igualdad

**opuesta-a.in**

```
set(binary_res).

list(sos).
 x = x.                                  % Reflexividad
 x != y | y = x.                         % Simetría
 x != y | y != z | x = z.                % Transisitividad
 opuesta(derecha) = izquierda.
 opuesta(izquierda) = derecha.
 opuesta(opuesta(derecha)) != derecha.
end_of_list.
```

# Axiomas de sustitución funcionales

- Formalización en OTTER con un axioma de sustitución

**opuesta-b.in**

```
include("opuesta-a.in").

list(sos).
 x != y | opuesta(x) = opuesta(y).
end_of_list.
```

# Axiomas de sustitución funcionales

## Prueba obtenida

```
 2 [] x != y | y = x.
 3 [] x != y | y != z | x = z.
 4 [] opuesta(derecha) = izquierda.
 5 [] opuesta(izquierda) = derecha.
 6 [] opuesta(opuesta(derecha)) != derecha.
 7 [] x != y | opuesta(x) = opuesta(y).
 8 [binary,2.1,5.1] derecha = opuesta(izquierda).
 9 [binary,2.1,4.1] izquierda = opuesta(derecha).
10 [binary,2.2,6.1]
     derecha != opuesta(opuesta(derecha)).
11 [binary,7.1,9.1]
     opuesta(izquierda) = opuesta(opuesta(derecha)).
36 [binary,3.1,8.1]
     opuesta(izquierda) != x | derecha = x.
81 [binary,36.1,11.1]
     derecha = opuesta(opuesta(derecha)).
82 [binary,81.1,10.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 23 | 130 | 74 | 49 | 0 |

# Axiomas de sustitución funcionales

- Formalización en OTTER con soporte y resolución UR

## opuesta-c.in

```
set(ur_res).

list(usable).
 x = x.                                  % Reflexividad
 x != y | y = x.                         % Simetría
 x != y | y != z | x = z.                % Transisitividad
 x != y | opuesta(x) = opuesta(y).       % Sustitución
 opuesta(derecha) = izquierda.
 opuesta(izquierda) = derecha.
end_of_list.

list(sos).
 opuesta(opuesta(derecha)) != derecha.
end_of_list.
```

# Axiomas de sustitución funcionales

## Prueba obtenida

```
 3 [] x != y | y != z | x = z.
 4 [] x != y | opuesta(x) = opuesta(y).
 5 [] opuesta(derecha) = izquierda.
 6 [] opuesta(izquierda) = derecha.
 7 [] opuesta(opuesta(derecha)) != derecha.
 8 [ur,7,3,6]
     opuesta(opuesta(derecha)) != opuesta(izquierda).
10 [ur,8,4] opuesta(derecha) != izquierda.
11 [binary,10.1,5.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 3 | 8 | 3 | 5 | 0 |

# Axiomas de sustitución relacionales

- Demostrar que si los padres son mayores que los hijos y Luis es el padre de Juan, entonces Luis es mayor que Juan
- Formalización en OTTER con axiomas de sustitución relacionales

## mayor-a.in

```
set(binary_res).

list(sos).
 x = x.                                    % Reflexividad
 x != y | y = x.                           % Simetría
 x != y | y != z | x = z.                  % Transisitividad
 x != y | Padre(x) = Padre(y).               % Sustitución
 x1 != x2 | -Mayor(x1,y) | Mayor(x2,y). % Sustitución
 y1 != y2 | -Mayor(x,y1) | Mayor(x,y2). % Sustitución
 Mayor(Padre(x),x).
 Padre(Juan) = Luis.
 -Mayor(Luis,Juan).
end_of_list.
```

# Axiomas de sustitución relacionales

## Prueba obtenida

```
  1 [] x = x.
  2 [] x != y | y = x.
  3 [] x != y | y != z | x = z.
  5 [] x1 != x2 | -Mayor(x1,y) | Mayor(x2,y).
  7 [] Mayor(Padre(x),x).
  8 [] Padre(Juan) = Luis.
  9 [] -Mayor(Luis,Juan).
 26 [binary,3.1,8.1] Luis != x | Padre(Juan) = x.
 53 [binary,26.1,2.2] Padre(Juan) = x | x != Luis.
303 [binary,5.3,9.1] x != Luis | -Mayor(x,Juan).
312 [binary,303.1,53.1,unit_del,7,1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 55 | 671 | 302 | 351 | 0 |

# Axiomas de sustitución relacionales

- Formalización en OTTER con soporte y resolución UR

## mayor-b.in

```
set(ur_res).

list(usable).
 x = x.                                    % Reflexividad
 x != y | y = x.                           % Simetría
 x != y | y != z | x = z.                  % Transisitividad
 x != y | Padre(x) = Padre(y).               % Sustitución
 x1 != x2 | -Mayor(x1,y) | Mayor(x2,y). % Sustitución
 y1 != y2 | -Mayor(x,y1) | Mayor(x,y2). % Sustitución
 Mayor(Padre(x),x).
 Padre(Juan) = Luis.
end_of_list.

list(sos).
 -Mayor(Luis,Juan).
end_of_list.
```

# Axiomas de sustitución relacionales

## Prueba obtenida

```
 5 [] x1 != x2 | -Mayor(x1,y) | Mayor(x2,y).
 7 [] Mayor(Padre(x),x).
 8 [] Padre(Juan) = Luis.
 9 [] -Mayor(Luis,Juan).
10 [ur,9,5,7] Padre(Juan) != Luis.
11 [binary,10.1,8.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 1 | 2 | 1 | 1 | 0 |

# Razonamiento con igualdad

- Formalización de los axiomas de igualdad mediante fórmulas

```
all x (x = x).                                  % Reflexividad
all x y (x = y -> y = x).                       % Simetría
all x y z (x = y & y = z -> x = z).             % Transitividad

% Axiomas de sustitución de la función f/3
all x1 x2 x3 y (x1 = y -> f(x1,x2,x3) = f(y,x2,x3)).
all x1 x2 x3 y (x2 = y -> f(x1,x2,x3) = f(x1,y,x3)).
all x1 x2 x3 y (x3 = y -> f(x1,x2,x3) = f(x1,x2,y)).

% Axiomas de sustitución de la relación P/3
all x1 x2 x3 y (x1 = y & P(x1,x2,x3) -> P(y,x2,x3)).
all x1 x2 x3 y (x2 = y & P(x1,x2,x3) -> P(x1,y,x3)).
all x1 x2 x3 y (x3 = y & P(x1,x2,x3) -> P(x1,x2,y)).
```

# Razonamiento con igualdad

- Formalización de los axiomas de igualdad mediante cláusulas

```
x = x.                              % Reflexividad
x != y | y = x.                     % Simetría
x != y | y! = z | x = z.            % Transitividad

% Axiomas de sustitución de la función f/3
x1 != y | f(x1,x2,x3) = f(y,x2,x3).
x2 != y | f(x1,x2,x3) = f(x1,y,x3).
x3 != y | f(x1,x2,x3) = f(x1,x2,y).

% Axiomas de sustitución de la relación P/3
x1 != y | -P(x1,x2,x3) | P(y,x2,x3).
x2 != y | -P(x1,x2,x3) | P(x1,y,x3).
x3 != y | -P(x1,x2,x3) | P(x1,x2,y).
```

# Paramodulación

- Regla de paramodulación
  - Izquierda

$$\frac{t_1 = t_2 \vee A_1 \vee \ldots \vee A_n \qquad B[t] \vee B_1 \vee \ldots \vee B_m}{\sigma(B[t_2] \vee A_1 \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_m)}$$

    donde $\sigma = umg(t, t_1)$
  - Derecha

$$\frac{t_1 = t_2 \vee A_1 \vee \ldots \vee A_n \qquad B[t] \vee B_1 \vee \ldots \vee B_m}{\sigma(B[t_1] \vee A_1 \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_m)}$$

    donde $\sigma = umg(t, t_2)$

# Paramodulación

- `set(para_into)`: Paramodulación en la cláusula dada
- `set(para_from)`: Paramodulación desde la cláusula dada

## paramodulacion.in

```
set(para_into).
set(para_from).

list(sos).
 P(f(x,b),x) | Q(x).
 f(a,x) = x | R(x).
end_of_list.
```

# Paramodulación

## Proceso de búsqueda

```
given clause #1: (wt=7) 1 [] P(f(x,b),x) | Q(x).

given clause #2: (wt=7) 2 [] f(a,x) = x | R(x).
* KEPT 3 [para_into,2.1.1,2.1.1] x = x | R(x).
* KEPT 4 [para_from,2.1.1,1.1.1] P(b,a) | Q(a) | R(b).
```

# Paramodulación

- Explicación

```
3 [para_into,2.1.1,2.1.1] x = x | R(x).
```

| | |
|---|---|
| Into 2.1.1 | $f(a,x_1) = x_1 \mid R(x_1)$ |
| From 2.1.1 | $f(a,x_2) = x_2 \mid R(x_2)$ |
| Unificador | $\sigma = \{x_2/x_1\}$ |
| Paramodulante | $\sigma(x_2 = x_1 \mid R(x_2) \mid R(x_1))$ |
| | $\Longrightarrow x_1 = x_1 \mid R(x_1)$ |
| | $\Longrightarrow x = x \mid R(x)$ |

```
4 [para_from,2.1.1,1.1.1] P(b,a) | Q(a) | R(b).
```

| | |
|---|---|
| From 2.1.1 | $f(a,x_1) = x_1 \mid R(x_1)$ |
| Into 1.1.1 | $P(f(x_2,b),x_2) \mid Q(x_2)$ |
| Unificador | $\sigma = \{x_2/a, x_1/b\}$ |
| Paramodulante | $\sigma(P(x_1,x_2) \mid Q(x_2) \mid R(x_1))$ |
| | $\Longrightarrow P(b,a) \mid Q(a) \mid R(b)$ |

# Paramodulación

- Demostrar que si Francisco es igual a Curro y a Paco, entonces Curro y Paco son iguales

## francisco-d.in

```
set(para_into).

list(usable).
 francisco = curro.
 francisco = paco.
end_of_list.

list(sos).
 paco != curro.
end_of_list.
```

# Paramodulación

## Prueba obtenida

```
1 [] francisco = curro.
2 [] francisco = paco.
3 [] paco != curro.
4 [para_into,3.1.1,2.1.2] francisco != curro.
5 [binary,4.1,1.1] $F.
```

- Explicación de la cláusula

```
4 [para_into,3.1.1,2.1.2] francisco != curro.

  Into 3.1.1       paco != curro
  From 2.1.2       francisco = paco
```

# Paramodulación

- Demostrar que si la opuesta de la derecha es la izquierda y la opuesta de la izquierda es la derecha, entonces la opuesta a la opuesta de la derecha es la derecha

## opuesta-d.in

```
set(para_into).

list(usable).
 opuesta(derecha) = izquierda.
 opuesta(izquierda) = derecha.
end_of_list.

list(sos).
 opuesta(opuesta(derecha)) != derecha.
end_of_list.
```

# Paramodulación

## Prueba obtenida

```
1 [] opuesta(derecha) = izquierda.
2 [] opuesta(izquierda) = derecha.
3 [] opuesta(opuesta(derecha)) != derecha.
5 [para_into,3.1.1.1,1.1.1]
    opuesta(izquierda) != derecha.
6 [binary,5.1,2.1] $F.
```

- Explicación de la cláusula

```
5 [para_into,3.1.1.1,1.1.1] opuesta(izquierda) != derecha.

  Into 3.1.1.1   opuesta(opuesta(derecha)) != derecha
  From 1.1.1     opuesta(derecha) = izquierda
```

# Paramodulación

- Demostrar que si los padres son mayores que los hijos y Luis es el padre de Juan, entonces Luis es mayor que Juan

## mayor-c.in

```
set(para_into).

list(usable).
 Mayor(Padre(x),x).
 Padre(Juan) = Luis.
end_of_list.

list(sos).
 -Mayor(Luis,Juan).
end_of_list.
```

# Paramodulación

## Prueba obtenida

```
1 [] Mayor(Padre(x),x).
2 [] Padre(Juan) = Luis.
3 [] -Mayor(Luis,Juan).
4 [para_into,3.1.1,2.1.2] -Mayor(Padre(Juan),Juan).
5 [binary,4.1,1.1] $F.
```

- Explicación de la cláusula

```
4 [para_into,3.1.1,2.1.2] -Mayor(Padre(Juan),Juan).

  Into 3.1.1      -Mayor(Luis,Juan)
  From 2.1.2      Padre(Juan) = Luis
```

# Paramodulación

- Demostrar que si Juan está casado y es el tío de Pepe, entonces el hermano del padre de Pepe está casado

## casados-a.in

```
set(para_into).

list(usable).
 casado(juan).
 hermano(padre(x)) = tio(x).
 tio(pepe) = juan.
end_of_list.

list(sos).
 -casado(hermano(padre(pepe))).
end_of_list.
```

# Paramodulación

## Prueba obtenida

```
1 [] casado(juan).
2 [] hermano(padre(x)) = tio(x).
3 [] tio(pepe) = juan.
4 [] -casado(hermano(padre(pepe))).
5 [para_into,4.1.1,2.1.1] -casado(tio(pepe)).
6 [para_into,5.1.1,3.1.1] -casado(juan).
8 [binary,6.1,1.1] $F.
```

# Demodulación

- Regla de demodulación

$$\frac{\boldsymbol{A}[t] \vee \boldsymbol{B_1} \vee \ldots \vee \boldsymbol{B_n} \qquad \boldsymbol{t_1} = \boldsymbol{t_2}}{\boldsymbol{\sigma}(\boldsymbol{A}[\boldsymbol{t_2}] \vee \boldsymbol{B_1} \vee \ldots \vee \boldsymbol{B_n})}$$

  donde $\boldsymbol{\sigma} = \boldsymbol{umg}(\boldsymbol{t}, \boldsymbol{t_1})$

- Paramodulación izquierda en la cláusula dada desde una cláusula unitaria

# Demodulación

- Demostrar que si Francisco es igual a Curro y a Paco, entonces Curro y Paco son iguales
- **`demodulators`**: Conjunto de demoduladores
- **`set(process_input)`**: Procesamiento de la entrada

## francisco-e.in

```
set(process_input).

list(usable).
 x = x.
end_of_list.

list(sos).
 paco != curro.
end_of_list.

list(demodulators).
 curro = francisco.
 paco = francisco.
end_of_list.
```

# Demodulación

## Prueba obtenida

```
1 [] curro = francisco.
2 [] paco = francisco.
3 [] x = x.
4 [] paco != curro.
5 [copy,4,demod,2,1] francisco != francisco.
6 [binary,5.1,3.1] $F.
```

- Explicación de la cláusula

```
5 [copy,4,demod,2,1] francisco != francisco.
```

```
Copy 4        paco != curro
Demod 2       paco = francisco
              ⟹ francisco != curro
Demod 1       curro = francisco
              ⟹ francisco != francisco
```

# Demodulación

- Demostrar que si la opuesta de la derecha es la izquierda y la opuesta de la izquierda es la derecha, entonces la opuesta a la opuesta de la derecha es la derecha

## opuesta-e.in

```
set(process_input).

list(usable).
 x = x.
end_of_list.

list(sos).
 opuesta(opuesta(derecha)) != derecha.
end_of_list.

list(demodulators).
 opuesta(derecha) = izquierda.
 opuesta(izquierda) = derecha.
end_of_list.
```

# Demodulación

## Prueba obtenida

```
1 [] opuesta(derecha) = izquierda.
2 [] opuesta(izquierda) = derecha.
3 [] x = x.
4 [] opuesta(opuesta(derecha)) != derecha.
5 [copy,4,demod,1,2] derecha != derecha.
6 [binary,5.1,3.1] $F.
```

- Explicación de la cláusula

```
5 [copy,4,demod,1,2] derecha != derecha.
```

```
Copy 4      opuesta(opuesta(derecha)) != derecha
Demod 1     opuesta(derecha) = izquierda
            ⟹ opuesta(izquierda) != derecha
Demod 2     opuesta(izquierda) = derecha
            ⟹ derecha != derecha
```

# Demodulación

- Demostrar que si Juan está casado y es el tío de Pepe, entonces el hermano del padre de Pepe está casado

## casados-b.in

```
set(process_input).

list(usable).
 x = x.
 casado(juan).
end_of_list.

list(sos).
 -casado(hermano(padre(pepe))).
end_of_list.

list(demodulators).
 hermano(padre(x)) = tio(x).
 tio(pepe) = juan.
end_of_list.
```

# Demodulación

## Prueba obtenida

```
1 [] hermano(padre(x)) = tio(x).
2 [] tio(pepe) = juan.
4 [] casado(juan).
5 [] -casado(hermano(padre(pepe))).
6 [copy,5,demod,1,2] -casado(juan).
7 [binary,6.1,4.1] $F.
```

- Explicación de la cláusula

```
6 [copy,5,demod,1,2] -casado(juan).
```

| | |
|---|---|
| `Copy 5` | `-casado(hermano(padre(pepe)))` |
| `Demod 1` | `hermano(padre(x)) = tio(x)` |
| `Unificador` | $\sigma = \{x/pepe\}$ |
| | $\Longrightarrow$ `-casado(tio(pepe))` |
| `Demod 2` | `tio(pepe) = juan` |
| | $\Longrightarrow$ `-casado(juan)` |

# Operadores

- Sea $G$ un grupo y $e$ su elemento neutro. Demostrar que si, para todo $x$ de $G$, $x^2 = e$, entonces $G$ es conmutativo
- Formalización
  - Axiomas de grupo

    $$(\forall x)[e.x = x]$$
    $$(\forall x)[x.e = x]$$
    $$(\forall x)[x.x^{-1} = e]$$
    $$(\forall x)[x^{-1}.x = e]$$
    $$(\forall x)(\forall y)(\forall z)[(x.y).z = x.(y.z)]$$
  - Hipótesis

    $$(\forall x)[x.x = e]$$
  - Conclusión

    $$(\forall x)(\forall y)[x.y = y.x]$$

# Operadores

## grupo-a.in

```
set(para_into).
set(para_from).

op(400, xfy, *).
op(300, yf, ^).

list(usable).
 x = x.                                  % Reflexividad
 e * x = x.                              % Ax. 1
 x * e = x.                              % Ax. 2
 x^ * x = e.                             % Ax. 3
 x * x^ = e.                             % Ax. 4
 (x * y) * z = x * (y * z).              % Ax. 5
end_of_list.

list(sos).
 x * x = e.
 a * b != b * a.
end_of_list.
```

# Operadores

## Prueba obtenida

```
  2 [] e*x = x.
  3 [] x*e = x.
  6 [] (x*y)*z = x*y*z.
  7 [] x*x = e.
  8 [] a*b != b*a.
 19 [para_from,7.1.2,3.1.1.2] x*y*y = x.
 20 [para_from,7.1.2,2.1.1.1] (x*x)*y = y.
 31 [para_into,19.1.1,6.1.2] (x*y)*y = x.
167 [para_into,20.1.1,6.1.1] x*x*y = y.
170 [para_from,20.1.1,6.1.1] x = y*y*x.
496 [para_into,167.1.1.2,31.1.1] (x*y)*x = y.
755 [para_into,496.1.1.1,170.1.2] x*y = y*x.
756 [binary,755.1,8.1] $F.
```

# Operadores

- Explicación de las cláusulas

```
19 [para_from,7.1.2,3.1.1.2] x*y*y = x.

   From 7.1.2        x1*x1 = e
   Into 3.1.1.2      x2*e = x2
                     ⟹ x2*(x1*x1) = x2 {x2/x, x1/y}
                     ⟹ x*(y*y) = x

20 [para_from,7.1.2,2.1.1.1] (x*x)*y = y.

   From 7.1.2        x1*x1 = e
   Into 2.1.1.1      e*x2 = x2
                     ⟹ (x1*x1)*x2 = x2 {x1/x, x2/y}
                     ⟹ (x*x)*y = y
```

# Operadores

- Explicación de las cláusulas

```
31 [para_into,19.1.1,6.1.2] (x*y)*y = x.
```

| | |
|---|---|
| Into 19.1.1 | $x_1*(y_1*y_1) = x_1$ |
| From 6.1.2 | $(x_2*y_2)*z_2 = x_2*(y_2*z_2)$ |
| Unificador | $\sigma = \{x_1/x_2,\ y_1/y_2,\ z_2/y_2\}$ |
| Paramodulante | $\sigma((x_2*y_2)*z_2 = x_1)$ |
| | $\Longrightarrow (x_2*y_2)*y_2 = x_2\ \{x_2/x,\ y_2/y\}$ |
| | $\Longrightarrow (x*y)*y = x$ |

```
167 [para_into,20.1.1,6.1.1] x*x*y = y.
```

| | |
|---|---|
| Into 20.1.1 | $(x_1*x_1)*y_1 = y_1$ |
| From 6.1.1 | $(x_2*y_2)*z_2 = x_2*(y_2*z_2)$ |
| Unificador | $\sigma = \{x_2/x_1,\ y_2/x_1,\ z_2/y_1\}$ |
| Paramodulante | $\sigma(x_2*(y_2*z_2) = y_1)$ |
| | $\Longrightarrow x_1*(x_1*y_1) = y_1\ \{x_1/x,\ y_1/y\}$ |
| | $\Longrightarrow x*(x*y) = y$ |

# Operadores

- Explicación de las cláusulas

```
170 [para_from,20.1.1,6.1.1] x = y*y*x.
```

| | |
|---|---|
| From 20.1.1 | $(x_1*x_1)*y_1 = y_1$ |
| Into 6.1.1 | $(x_2*y_2)*z_2 = x_2*(y_2*z_2)$ |
| Unificador | $\sigma = \{x_2/x_1,\ y_2/x_1,\ z_2/y_1\}$ |
| Paramodulante | $\sigma(y_1 = x_2*(y_2*z_2))$ |
| | $\Longrightarrow\ y_1 = x_1*(x_1*y_1)\ \{y_1/x,\ x_1/y\}$ |
| | $\Longrightarrow\ x = y*(y*x)$ |

```
496 [para_into,167.1.1.2,31.1.1] (x*y)*x = y.
```

| | |
|---|---|
| Into 167.1.1.2 | $x_1*(x_1*y_1) = y_1$ |
| From 31.1.1 | $(x_2*y_2)*y_2 = x_2$ |
| Unificador | $\sigma = \{x_1/x_2*y_2,\ y_1/y_2\}$ |
| Paramodulante | $\sigma(x_1*x_2 = y_1)$ |
| | $\Longrightarrow\ (x_2*y_2)*x_2 = y_2\ \{x_2/x,\ y_2/y\}$ |
| | $\Longrightarrow\ (x*y)*x = y$ |

# Operadores

- Explicación de las cláusulas

```
755 [para_into,496.1.1.1,170.1.2] x*y = y*x.
```

| | |
|---|---|
| Into 496.1.1.1 | $(x_1 * y_1) * x_1 = y_1$ |
| From 170.1.2 | $x_2 = y_2 * (y_2 * x_2)$ |
| Unificador | $\sigma = \{x_1/y_2,\ y_1/y_2 * x_2\}$ |
| Paramodulante | $\sigma((x_2 * x_1) = y_1)$ |
| | $\Longrightarrow x_2 * y_2 = y_2 * x_2\ \{x_2/x,\ y_2/y\}$ |
| | $\Longrightarrow x * y = y * x$ |

- Estadísticas

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 72 | 5741 | 747 | 4994 | 45 |

# Operadores

- Mejora con demoduladores

## grupo-b.in

```
include("grupo-a.in").

list(demodulators).
 e * x = x.                          % Ax. 1
 x * e = x.                          % Ax. 2
 x^ * x = e.                         % Ax. 3
 x * x^ = e.                         % Ax. 4
 (x * y) * z = x * (y * z).          % Ax. 5
end_of_list.
```

# Operadores

## Prueba obtenida

```
  2 [] e*x = x.
  6 [] (x*y)*z = x*y*z.
  7 [] x*x = e.
  8 [] a*b != b*a.
 10 [] x*e = x.
 13 [] (x*y)*z = x*y*z.
 14 [para_into,7.1.1,6.1.2,demod,13,13,13] x*y*x*y = e.
 20 [para_from,7.1.2,2.1.1.1,demod,13] x*x*y = y.
494 [para_from,14.1.1,20.1.1.2,demod,10] x = y*x*y.
540 [para_from,494.1.2,20.1.1.2] x*y = y*x.
541 [binary,540.1,8.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 68 | 5562 | 527 | 5035 | 7 |

# Operadores

- Mejora con demoduladores dinámicos
- **`set(dynamic_demod)`**: Detección dinámica de demoduladores

**grupo-c.in**

```
set(dynamic_demod).

include("grupo-b.in").
```

# Operadores

## Prueba obtenida

```
 6 [] (x*y)*z =x*y*z.
 7 [] x*x = e.
 8 [] a*b != b*a.
 9 [] e*x = x.
10 [] x*e = x.
13 [] (x*y)*z = x*y*z.
14 [para_into,7.1.1,6.1.2,demod,13,13,13] x*y*x*y = e.
19 [para_from,7.1.1,6.1.1.1,demod,9,flip.1] x*x*y = y.
31 [para_from,14.1.1,19.1.1.2,demod,10,flip.1] x*y*x = y.
36 [para_from,31.1.1,19.1.1.2] x*y = y*x.
37 [binary,36.1,8.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás |
|---|---|---|---|---|
| 10 | 219 | 13 | 206 | 1 |

# Operadores

- Modo automático

## grupo-d.in

```
set(auto2).

op(400, xfy, *).
op(300, yf, ^).

list(usable).
 e * x = x.                          % Ax. 1
 x * e = x.                          % Ax. 2
 x^ * x = e.                         % Ax. 3
 x * x^ = e.                         % Ax. 4
 (x * y) * z = x * (y * z).          % Ax. 5
 x = x.                              % Ax. 6
 x * x = e.
 a * b != b * a.
end_of_list.
```

# Operadores

## Prueba obtenida

```
1 [] a*b!=b*a.
2 [copy,1,flip.1] b*a!=a*b.
4,3 [] e*x=x.
6,5 [] x*e=x.
11 [] (x*y)*z=x*y*z.
14 [] x*x=e.
18 [para_into,11.1.1.1,14.1.1,demod,4,flip.1] x*x*y=y.
24 [para_into,11.1.1,14.1.1,flip.1] x*y*x*y=e.
34 [para_from,24.1.1,18.1.1.2,demod,6,flip.1] x*y*x=y.
38 [para_from,34.1.1,18.1.1.2] x*y=y*x.
39 [binary,38.1,2.1] $F.
```

| Analiz. | Gener. | Reten. | Sub. adel. | Sub. atrás | Seg. |
|---|---|---|---|---|---|
| 12 | 90 | 20 | 87 | 8 | 0.18 |

# Bibliografía


- Alonso, J.A.; Fernández, A. y Pérez, M.J. *Razonamiento automático*. (en *Lógica formal (Orígenes, métodos y aplicaciones)*, Ed. Kronos, 1995)
- Chang, C.L. y Lee, R.C.T. *Symbolic logic and mechanical theorem proving*. (Academic Press, 1973)
  - Cap. 8 "The equality relation"
- Genesereth, M.R. y Nilsson, N.J. *Logical foundations of Artificial Intelligence* (Morgan Kaufmann, 1987)
  - Cap. 9: "Relational resolution"