

Calibrado de programas lógicos difusos mediante satisfacibilidad módulo teorías

José Antonio Riaza Valverde

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Universidad de Castilla-La Mancha

Director: Ginés Moreno Valverde

19 de febrero de 2019

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro

Concatenación de listas

```
append([],X,X)  
append([H|T],X,[H|S]) ← append(T,X,S)
```

Concatenación de listas

```
append([],X,X)  
append([H|T],X,[H|S]) ← append(T,X,S)
```

```
?- append([1,2],[3,4],X)
```

Concatenación de listas

```
append([],X,X)
append([H|T],X,[H|S]) ← append(T,X,S)
```

?- append([1,2],[3,4],X)
{X/[1,2,3,4]}

Concatenación de listas

```
append([],X,X)
append([H|T],X,[H|S]) ← append(T,X,S)
```

```
?- append([1,2],[3,4],X)
   {X/[1,2,3,4]}
```

```
?- append(X,Y,[1,2,3,4])
```

Concatenación de listas

```
append([],X,X)
append([H|T],X,[H|S]) ← append(T,X,S)
```

```
?- append([1,2],[3,4],X)
{X/[1,2,3,4]}
```

```
?- append(X,Y,[1,2,3,4])
{X/[], Y/[1,2,3,4]}
{X/[1], Y/[2,3,4]}
{X/[1,2], Y/[3,4]}
{X/[1,2,3], Y/[4]}
{X/[1,2,3,4], Y/[]}
```


El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
elegant/1 ~ vanguardist/1 = 0.6
    metro ~ bus = 0.5
    bus ~ taxi = 0.4
```

El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
elegant/1 ~ vanguardist/1 = 0.6
    metro ~ bus = 0.5
    bus ~ taxi = 0.4
```

?- good_hotel(hydropolis)

El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
elegant/1 ~ vanguardist/1 = 0.6
    metro ~ bus = 0.5
    bus ~ taxi = 0.4
```

```
?- good_hotel(hydropolis)
<0.38, {}>
```

El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
elegant/1 ~ vanguardist/1 = 0.6
    metro ~ bus = 0.5
    bus ~ taxi = 0.4
```

```
?- good_hotel(hydropolis)           ?- good_hotel(X)
<0.38, {}>
```

El buen hotel

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
elegant/1 ~ vanguardist/1 = 0.6
    metro ~ bus = 0.5
    bus ~ taxi = 0.4
```

```
?- good_hotel(hydropolis)
<0.38, {}>
```

```
?- good_hotel(X)
<0.38, {X/hydropolis}>
<0.4, {X/ritz}>
```



FASILL

Fuzzy Aggregators and Similarities Into a Logic Language



Fuzzy Aggregators and Similarities Into a Logic Language

- Lenguaje de programación **lógico difuso**

FASILL

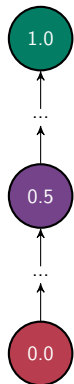
Fuzzy Aggregators and Similarities Into a Logic Language

- Lenguaje de programación **lógico difuso**
- Permite **relaciones de similitud**
 - Términos sintácticamente distintos *se parecen* con cierto grado
 $\hat{\mathcal{R}}(\text{elegant}(\text{taxi}), \text{vanguardist}(\text{bus})) = 0.4$

FASILL

Fuzzy Aggregators and Similarities Into a Logic Language

- Lenguaje de programación **lógico difuso**
- Permite **relaciones de similitud**
 - Términos sintácticamente distintos *se parecen* con cierto grado
 $\hat{\mathcal{R}}(\text{elegant}(\text{taxi}), \text{vanguardist}(\text{bus})) = 0.4$
- Trabaja con **retículos completos**
 - Cada programa puede definir su propia noción de verdad
($[0, 1], \leq$)



$$\dot{\&}_{luka}(x, y) \triangleq \text{máx}\{0, x + y - 1\}$$

$$\dot{\&}_{prod}(x, y) \triangleq xy$$

$$\dot{\&}_{godel}(x, y) \triangleq \text{mín}\{x, y\}$$

$$\dot{|}_{prod}(x, y) \triangleq x + y - xy$$

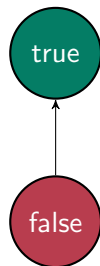
$$\dot{|}_{godel}(x, y) \triangleq \text{máx}\{x, y\}$$

$$\dot{|}_{luka}(x, y) \triangleq \text{mín}\{x + y, 1\}$$

$$\dot{\@}_{aver}(x, y) \triangleq (x + y)/2$$

$$\dot{\@}_{geom}(x, y) \triangleq \sqrt{xy}$$

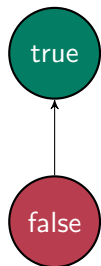
$$\dot{\@}_{very}(x) \triangleq x^2$$



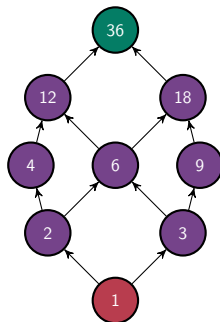
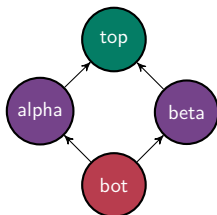
$$\&_{bool}(x, y) \triangleq x \wedge y \quad |_{bool}(x, y) \triangleq x \vee y$$

$$\@_{not}(x) \triangleq \neg x \quad \@_{xor}(x, y) \triangleq x \oplus y$$

Retículo booleano



$$\begin{aligned} \&_{bool}(x, y) &\triangleq x \wedge y & \vee_{bool}(x, y) &\triangleq x \vee y \\ \oplus_{not}(x) &\triangleq \neg x & \oplus_{xor}(x, y) &\triangleq x \oplus y \end{aligned}$$



```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

⟨good_hotel(X); id⟩

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

$\langle \text{good_hotel}(X); id \rangle$ $\overset{SS^{R4}}{\rightsquigarrow}$

$\langle @_{\text{aver}}(\text{elegant}(X1), @_{\text{very}}(\text{close}(X1, \text{metro}))); \{X/X1\} \rangle$ $\overset{SS^{R2}}{\rightsquigarrow}$

$\langle @_{\text{aver}}(0.8, @_{\text{very}}(\text{close}(\text{ritz}, \text{metro}))); \{X/ritz\} \rangle$ $\overset{FS}{\rightsquigarrow}$

$\langle @_{\text{aver}}(0.8, @_{\text{very}}(0)); \{X/ritz\} \rangle$

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

$\langle \text{good_hotel}(X); id \rangle$	SS^{R4}
	\rightsquigarrow
$\langle @\text{aver}(\text{elegant}(X1), @\text{very}(\text{close}(X1, \text{metro}))); \{X/X1\} \rangle$	SS^{R2}
	\rightsquigarrow
$\langle @\text{aver}(0.8, @\text{very}(\text{close}(\text{ritz}, \text{metro}))); \{X/\text{ritz}\} \rangle$	FS
	\rightsquigarrow
$\langle @\text{aver}(0.8, @\text{very}(0)); \{X/\text{ritz}\} \rangle$	IS
	\rightsquigarrow
$\langle @\text{aver}(0.8, 0); \{X/\text{ritz}\} \rangle$	IS
	\rightsquigarrow
$\langle 0.4; \{X/\text{ritz}\} \rangle$	

El entorno FLOPER: consola interactiva

```
fasill> consult('../sample/program/good_hotel.fpl').  
< 1.0, {} > ;  
  
fasill> good_hotel(X).  
< 0.4, {X/ritz} > ;  
  
fasill> consult_sim('../sample/sim/good_hotel.sim.pl').  
< 1.0, {} > ;  
  
fasill> good_hotel(X).  
< 0.38, {X/hydropolis} > ;  
< 0.4, {X/ritz} > ;  
  
fasill> findall(hotel(X,Y), truth_degree(good_hotel(X),Y), L).  
< 1.0, {X/X, Y/Y, L/[hotel(hydropolis, 0.38), hotel(ritz, 0.4)]} > ;  
  
fasill> taxi = metro ; taxi ~ metro.  
< 0.0, {} > ;  
< 0.4, {} > ;
```

Input

</> Program

```
1 vanguardist(hydropolis) <- 0.9.  
2 elegant(ritz) <- 0.8.  
3 close(hydropolis, taxi) <- 0.7.  
4 good_hotel(X) <- @aver(elegant(X), @very(close(X, metro))).  
5
```

Unfold program

● Lattice

```
1 % Elements  
2 member(X) :- number(X), 0 <= X, X <= 1.  
3 members([0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]).  
4  
5 % Distance  
6 distance(X,Y,Z) :- Z is abs(Y-X).
```

bool unit real

≡ Similarity Relation

```
1 elegant/1 ~ vanguardist/1 = 0.6.  
2 metro - bus = 0.5.  
3 bus - taxi = 0.4.  
4 -tnorm = godel.  
5
```

El entorno FLOPER: herramienta online (II)

Running Tuning

Goal

```
good_hotel(X).
```

Run

Output

Q Fuzzy Computed Answers

```
1 <0.38, {X/hydropolis}>
2 <0.4, {X/ritz}>
3 execution time: 3 milliseconds
```

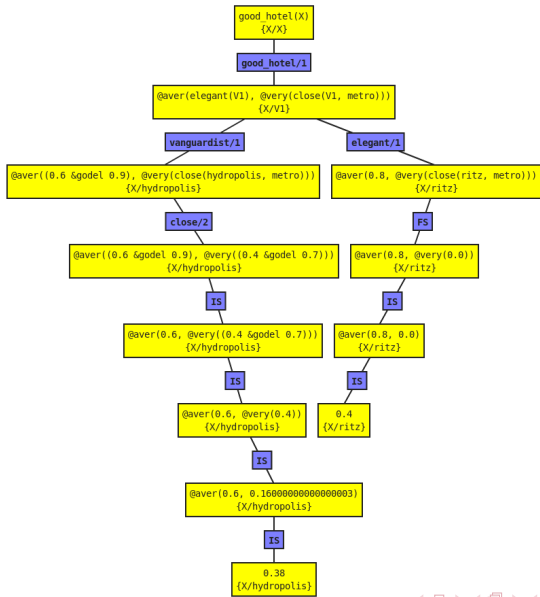
🌲 Derivation tree

```
1 GOAL <good_hotel(X), {X/X}>
2 good_hotel/1 <@aver(elegant(V1), @very(close(V1, metro))), {X/V1}>
3 vanguardist/1 <@aver((0.6 &godel 0.9), @very(close(hydropolis, metro))), {X/hydropolis}>
4 close/2 <@aver((0.6 &godel 0.9), @very((0.4 &godel 0.7))), {X/hydropolis}>
5 IS <@aver(0.6, @very((0.4 &godel 0.7))), {X/hydropolis}>
6 IS <@aver(0.6, @very(0.4)), {X/hydropolis}>
7 IS <@aver(0.6, 0.16000000000000003), {X/hydropolis}>
8 IS <0.38, {X/hydropolis}>
9 elegant/1 <@aver(0.8, @very(close(ritz, metro))), {X/ritz}>
10 FS <@aver(0.8, @very(0.0)), {X/ritz}>
11 IS <@aver(0.8, 0.0), {X/ritz}>
12 IS <0.4, {X/ritz}>
```

Draw derivation tree



El entorno FLOPER: herramienta online (III)



El entorno FLOPER: IDE

The screenshot displays the FLOPER IDE interface. At the top is a menu bar with options: File, Edit, Code, Run, Transformations, Options, and Help. Below the menu is a toolbar with various icons for file operations and execution. The main workspace is divided into three sections:

- Project:** A tree view showing a project named 'good_hotel' with subfolders 'fuzzy', 'lat', 'prolog', 'script', and 'sim'. The 'fuzzy' folder contains files 'good_hotel.fpl', 'lists.fpl', and 'pair.fpl'. Other files in the project include 'build.xml', 'tmp_fuzzy-goal.pl', 'tmp_fuzzy-inter.pl', 'tmp_fuzzy-prolog.pl', and 'tmp_pl-goal.pl'.
- lists.fpl:** A code editor showing the following Prolog code:

```
range(X, X, list(X,empty)).
range(X, Y, list(X, T)) <- X<Y & Z is X+1 & range
(Z, Y, T).

member(X, list(X,_)).
member(X, list(_,T)) <- member(X,T).

append(empty, X, X).
append(list(H,T), X, list(H,S)) <- append(T, X, S
).

reverse(empty, empty).
reverse(list(H,T), R) <- reverse(T, S) & append(S
, list(H,empty), R).

permutations(empty, empty).
```
- Console:** A text area showing the output of the IDE's initialization process:

```
Loading lattice file of "Good Hotel" project...
  Lattice "bool.lat.pl" specified
  Lattice "bool.lat.pl" file found
  Lattice "bool.lat.pl" file loaded correctly
Loading similarities file of "Good Hotel" project...
  Similarities file was not specified
Loading main fuzzy-prolog file of "Good Hotel" project...
  Main fuzzy-prolog "lists.fpl" file specified
  Main fuzzy-prolog "lists.fpl" file found
```

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos**
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro

Extensión simbólica FASILL (sFASILL)

¿El buen hotel?

```
vanguardist(hydropolis) ← 0.9
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

¿El buen hotel?

```
vanguardist(hydropolis) ← #s1
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← @aver(elegant(X),
                          @very(close(X, metro)))
```

¿El buen hotel?

```
vanguardist(hydropolis) ← #s1
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← #@s2(elegant(X),
                        @very(close(X, metro)))
```

¿El buen hotel?

```
vanguardist(hydropolis) ← #s1
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← #@s2(elegant(X),
                        @very(close(X, metro)))
```

?- good_hotel(X)

¿El buen hotel?

```
vanguardist(hydropolis) ← #s1
    elegant(ritz) ← 0.8
close(hydropolis, taxi) ← 0.7
    good_hotel(X) ← #@s2(elegant(X),
        @very(close(X, metro)))
```

```
?- good_hotel(X)
    <#@s2((0.6 &godel #s1), 0.16), {X/hydropolis}>
    <#@s2(0.8, 0.0), {X/ritz}>
```

Calibrado de programas lógicos difusos (I)

Casos de prueba

0.3 → `good_hotel(hydropolis)`

0.4 → `good_hotel(ritz)`

Casos de prueba

0.3 \rightarrow `good_hotel(hyropolis)`

0.4 \rightarrow `good_hotel(ritz)`

Constantes simbólicas

$v^{s1} \in \{0.3, 0.5, 0.7\}$

$@^{s2} \in \{@_{aver}, @_{geom}\}$

Calibrado de programas lógicos difusos (I)

Casos de prueba

0.3 \rightarrow `good_hotel(hydropolis)`

0.4 \rightarrow `good_hotel(ritz)`

Constantes simbólicas

$v^{s1} \in \{0.3, 0.5, 0.7\}$

$@^{s2} \in \{@_{aver}, @_{geom}\}$

Sustituciones simbólicas

$\Theta_1 = \{v^{s1}/0.3, @^{s2}/@_{aver}\}$ $\Theta_2 = \{v^{s1}/0.3, @^{s2}/@_{geom}\}$

$\Theta_3 = \{v^{s1}/0.5, @^{s2}/@_{aver}\}$ $\Theta_4 = \{v^{s1}/0.5, @^{s2}/@_{geom}\}$

$\Theta_5 = \{v^{s1}/0.7, @^{s2}/@_{aver}\}$ $\Theta_6 = \{v^{s1}/0.7, @^{s2}/@_{geom}\}$

Calibrado de programas lógicos difusos (II)

t_1 : 0.3 \rightarrow good_hotel(hydropolis)

t_2 : 0.4 \rightarrow good_hotel(ritz)

Calibrado de programas lógicos difusos (II)

$t_1 : 0.3 \rightarrow \text{good_hotel}(\text{hydropolis})$

$t_2 : 0.4 \rightarrow \text{good_hotel}(\text{ritz})$

$\Downarrow \rightsquigarrow_*$

$Q'_1 : \#s2((0.6 \ \&godel \ #s1), 0.16)$

$Q'_2 : \#s2(0.8, 0.0)$

Calibrado de programas lógicos difusos (II)

$t_1 : 0.3 \rightarrow \text{good_hotel}(\text{hydropolis})$

$t_2 : 0.4 \rightarrow \text{good_hotel}(\text{ritz})$

$\Downarrow \rightsquigarrow_*$

$Q'_1 : \#s2((0.6 \ \&godel \ s1), 0.16)$

$Q'_2 : \#s2(0.8, 0.0)$

$\Downarrow \theta_1 \cdots \Downarrow \theta_n$

$Q'_1 \theta_1 : \text{@aver}((0.6 \ \&godel \ 0.3), 0.16)$

$Q'_2 \theta_1 : \text{@aver}(0.8, 0.0)$

Calibrado de programas lógicos difusos (II)

$t_1 : 0.3 \rightarrow \text{good_hotel}(\text{hydropolis})$

$t_2 : 0.4 \rightarrow \text{good_hotel}(\text{ritz})$

$\Downarrow \rightsquigarrow_*$

$Q'_1 : \#s2((0.6 \ \&godel \ s1), 0.16)$

$Q'_2 : \#s2(0.8, 0.0)$

$\Downarrow_{\theta_1} \cdots \Downarrow_{\theta_n}$

$Q'_1 \theta_1 : \text{@aver}((0.6 \ \&godel \ 0.3), 0.16)$ $Q_1 \theta_1 : 0.23$

$Q'_2 \theta_1 : \text{@aver}(0.8, 0.0)$ $\Rightarrow_{\rightsquigarrow_*} \text{IS} \quad Q_2 \theta_1 : 0.4$

Calibrado de programas lógicos difusos (II)

$t_1 : 0.3 \rightarrow \text{good_hotel}(\text{hydropolis})$

$t_2 : 0.4 \rightarrow \text{good_hotel}(\text{ritz})$

$\Downarrow_{\rightsquigarrow_*}$

$Q'_1 : \text{\#@s2}((0.6 \ \&\text{godel} \ \#s1), 0.16)$

$Q'_2 : \text{\#@s2}(0.8, 0.0)$

$\Downarrow_{\Theta_1} \cdots \Downarrow_{\Theta_n}$

$Q'_1 \Theta_1 : \text{\@aver}((0.6 \ \&\text{godel} \ 0.3), 0.16)$ $\Rightarrow_{\rightsquigarrow_*}^{\text{IS}}$ $Q_1 \Theta_1 : 0.23$

$Q'_2 \Theta_1 : \text{\@aver}(0.8, 0.0)$ $\Rightarrow_{\rightsquigarrow_*}^{\text{IS}}$ $Q_2 \Theta_1 : 0.4$

\Downarrow_{d_i}

$d_{(1,1)} : d(0.3, 0.23) = 0.07$

$d_{(2,1)} : d(0.4, 0.4) = 0.0$

$d_1 : d_{(1,1)} + d_{(2,1)} = 0.07 + 0.0 = 0.07$

Calibrado de programas lógicos difusos (III)

Tabla: Resumen de pesos al calibrar el programa del buen hotel

v^{s1}	$@^{s2}$	Θ	<i>hydropolis</i>		<i>ritz</i>		<i>z</i>
0.3	$@_{aver}$	Θ_1	0.23	0.07	0.4	0.0	0.07
	$@_{geom}$	Θ_2	0.22	0.08	0.0*	0.4*	0.48*
0.5	$@_{aver}$	Θ_3	0.33	0.03	0.4	0.0	0.03
	$@_{geom}$	Θ_4	0.28	0.02	0.0	0.4	0.42*
0.7	$@_{aver}$	Θ_5	0.38	0.08	0.4*	0.0*	0.48*
	$@_{geom}$	Θ_6	0.31	0.01	0.0	0.4	0.41*

Running

Tuning

Test cases

```
1 0.3 -> good_hotel(hydropolis).
2 0.4 -> good_hotel(ritz).
3
```

FASILL – Thresholded method ▾

Tune

Output

Symbolic substitution

```
1 best symbolic substitution: {#s1/0.5, #@s2/@aver}
2 deviation: 0.0300000000000000027
3 execution time: 3 milliseconds
```

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad**
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro

El problema de satisfacibilidad

- Determinar si una fórmula proposicional puede evaluarse como cierta

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

$$(\bar{0}) \wedge (\bar{1} \vee \bar{0}) \wedge (1 \vee 0) \wedge (1 \vee \bar{0} \vee 0) \equiv$$

$$(1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 1 \vee 0) \equiv$$

$$1 \wedge 1 \wedge 1 \wedge 1 \equiv 1$$

El problema de satisfacibilidad

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

$$(\bar{0}) \wedge (\bar{1} \vee \bar{0}) \wedge (1 \vee 0) \wedge (1 \vee \bar{0} \vee 0) \equiv$$

$$(1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 1 \vee 0) \equiv$$

$$1 \wedge 1 \wedge 1 \wedge 1 \equiv 1$$

- Primer problema en ser clasificado como NP-completo

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

$$(\bar{0}) \wedge (\bar{1} \vee \bar{0}) \wedge (1 \vee 0) \wedge (1 \vee \bar{0} \vee 0) \equiv$$

$$(1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 1 \vee 0) \equiv$$

$$1 \wedge 1 \wedge 1 \wedge 1 \equiv 1$$

- Primer problema en ser clasificado como NP-completo
- Varios dominios de aplicación importantes

El problema de satisfacibilidad

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

$$(\bar{0}) \wedge (\bar{1} \vee \bar{0}) \wedge (1 \vee 0) \wedge (1 \vee \bar{0} \vee 0) \equiv$$

$$(1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 1 \vee 0) \equiv$$

$$1 \wedge 1 \wedge 1 \wedge 1 \equiv 1$$

- Primer problema en ser clasificado como NP-completo
- Varios dominios de aplicación importantes
- Algoritmos eficientes en las últimas décadas: DPLL y CDCL

El problema de satisfacibilidad

- Determinar si una fórmula proposicional puede evaluarse como cierta

$$(\bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$x_1 = 1, x_2 = 0, x_3 = 0$$

$$(\bar{0}) \wedge (\bar{1} \vee \bar{0}) \wedge (1 \vee 0) \wedge (1 \vee \bar{0} \vee 0) \equiv$$

$$(1) \wedge (0 \vee 1) \wedge (1 \vee 0) \wedge (1 \vee 1 \vee 0) \equiv$$

$$1 \wedge 1 \wedge 1 \wedge 1 \equiv 1$$

- Primer problema en ser clasificado como NP-completo
- Varios dominios de aplicación importantes
- Algoritmos eficientes en las últimas décadas: DPLL y CDCL
 - Solucionadores de satisfacibilidad (SAT)

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden)

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

- **Problema:** Los demostradores de teoremas de primer orden de propósito general no son capaces de resolver estas fórmulas

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

- **Problema:** Los demostradores de teoremas de primer orden de propósito general no son capaces de resolver estas fórmulas
 - Algunas teorías no pueden ser capturadas por un conjunto finito de fórmulas de primer orden

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

- **Problema:** Los demostradores de teoremas de primer orden de propósito general no son capaces de resolver estas fórmulas
 - Algunas teorías no pueden ser capturadas por un conjunto finito de fórmulas de primer orden
 - Cuando esto es posible, el rendimiento suele ser inaceptable

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

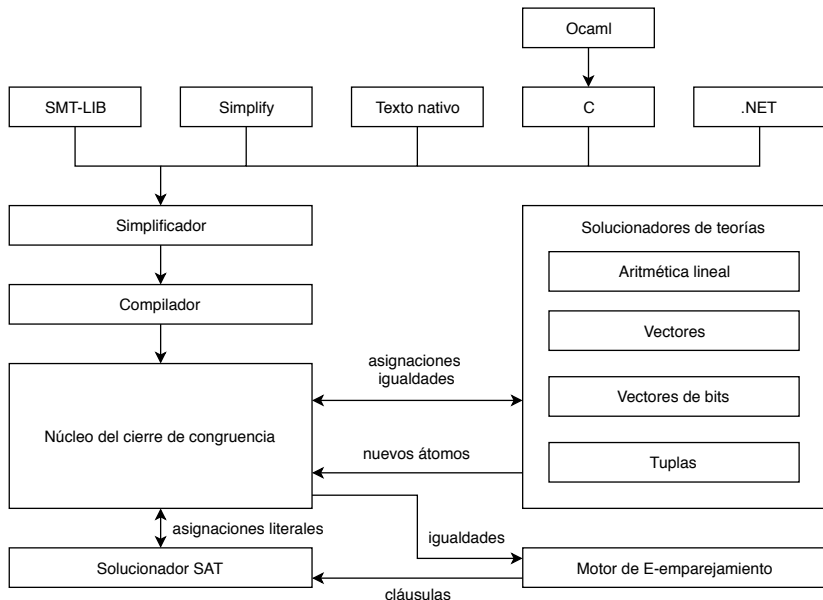
- **Problema:** Los demostradores de teoremas de primer orden de propósito general no son capaces de resolver estas fórmulas
 - Algunas teorías no pueden ser capturadas por un conjunto finito de fórmulas de primer orden
 - Cuando esto es posible, el rendimiento suele ser inaceptable
- **Solución:** Métodos adaptados a la teoría en cuestión

- Necesidad de determinar la satisfacibilidad de fórmulas en lógicas más expresivas (lógica de primer orden), respecto a alguna teoría que fije la interpretación de ciertos símbolos

$$(x < y) \wedge \neg(x < y + 0)$$

- **Problema:** Los demostradores de teoremas de primer orden de propósito general no son capaces de resolver estas fórmulas
 - Algunas teorías no pueden ser capturadas por un conjunto finito de fórmulas de primer orden
 - Cuando esto es posible, el rendimiento suele ser inaceptable
- **Solución:** Métodos adaptados a la teoría en cuestión
 - Solucionadores de satisfacibilidad módulo teorías

El solucionador Z3



- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT**
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro

- El calibrado puede ser visto como un problema de optimización:

- El calibrado puede ser visto como un problema de optimización:
*Encontrar los valores y conectivos del retículo que **minimizan** el error con respecto a un conjunto de casos de prueba*

- El calibrado puede ser visto como un problema de optimización:
*Encontrar los valores y conectivos del retículo que **minimizan** el error con respecto a un conjunto de casos de prueba*
- Una vez obtenida las sfca's, sólo es necesario asignar valores e interpretar las expresiones

- El calibrado puede ser visto como un problema de optimización:
*Encontrar los valores y conectivos del retículo que **minimizan** el error con respecto a un conjunto de casos de prueba*
- Una vez obtenida las sfca's, sólo es necesario asignar valores e interpretar las expresiones
- **Ventaja:** Los solucionadores SMT no necesitan discretizar los elementos del retículo

- El calibrado puede ser visto como un problema de optimización:
*Encontrar los valores y conectivos del retículo que **minimizan** el error con respecto a un conjunto de casos de prueba*
- Una vez obtenida las sfca's, sólo es necesario asignar valores e interpretar las expresiones
- **Ventaja:** Los solucionadores SMT no necesitan discretizar los elementos del retículo
- **Inconveniente:** La aritmética real no lineal es muy costosa, y los solucionadores SMT no son completos para estas fórmulas

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT
 - Z3 puede ejecutar guiones en lenguaje SMT-LIB

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT
 - Z3 puede ejecutar guiones en lenguaje SMT-LIB
 - La sintaxis de SMT-LIB es tipo LISP (fácil de analizar)

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT
 - Z3 puede ejecutar guiones en lenguaje SMT-LIB
 - La sintaxis de SMT-LIB es tipo LISP (fácil de analizar)
- Traducir los retículos de Prolog a SMT-LIB, y expresar las L^S -expresiones de FASILL como expresiones SMT-LIB

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT
 - Z3 puede ejecutar guiones en lenguaje SMT-LIB
 - La sintaxis de SMT-LIB es tipo LISP (fácil de analizar)
- Traducir los retículos de Prolog a SMT-LIB, y expresar las L^S -expresiones de FASILL como expresiones SMT-LIB
- **Problema**: Prolog no cuenta con un paquete para analizar SMT-LIB

- FASILL está implementado sobre Prolog, pero Z3 no cuenta con una interfaz para Prolog
- **SMT-LIB**: iniciativa internacional para facilitar el desarrollo de SMT
 - Z3 puede ejecutar guiones en lenguaje SMT-LIB
 - La sintaxis de SMT-LIB es tipo LISP (fácil de analizar)
- Traducir los retículos de Prolog a SMT-LIB, y expresar las L^S -expresiones de FASILL como expresiones SMT-LIB
- **Problema**: Prolog no cuenta con un paquete para analizar SMT-LIB
- **Solución**: ¡Crearlos!

Analizador de SMT-LIB en Prolog

An SMT-LIB parser in Prolog <http://jarlaza.es/swipl/smtlib>

Edit

Manage topics

41 commits

1 branch

0 releases

1 contributor

BSD-3-Clause

Branch: master










New pull request

Create new file

Upload files

Find file

Clone or download

 jarlazavalverde release v0.0.6	Latest commit 5ba7551 24 days ago
 prolog	release v0.0.6 24 days ago
 releases	release v0.0.6 24 days ago
 sample	release v0.0.6 24 days ago
 .gitattributes	release v0.0.6 24 days ago
 LICENSE	release v0.0.6 24 days ago
 README.md	release v0.0.6 24 days ago
 bibliography.bib	release v0.0.6 24 days ago
 pack.pl	release v0.0.6 24 days ago



Packs (add-ons) for SWI-Prolog

[Home](#)[DOWNLOAD](#)[DOCUMENTATION](#)[TUTORIALS](#)[COMMUNITY](#)[USERS](#)[WIKI](#)

Package "smtlib"

Title: *SMT-LIB parser for SWI-Prolog*

Rating: Not rated. [Create](#) the first rating!

Latest version: 0.0.6

SHA1 sum: 76fe142cdea350a88a300e4f192f1af4026ec505

Author: Jose Antonio Riaza Valverde <riaza.valverde@gmail.com>

Maintainer: Jose Antonio Riaza Valverde <riaza.valverde@gmail.com>

Packager: Jose Antonio Riaza Valverde <riaza.valverde@gmail.com>

Home page: <https://github.com/jariazavalverde/prolog-smtlib>

Download URL: <https://github.com/jariazavalverde/prolog-smtlib>

 Barrett, C., Fontaine, P. & Tinelli, C. *The SMT-LIB Standard: Version 2.6*. Department of Computer Science, The University of Iowa (2017)

SMT-LIB

THE SATISFIABILITY MODULO THEORIES LIBRARY

Utilities and Tools

The page contains a set of utilities and tools to work with the SMT-LIB 2 format.

Note: Most of the utilities tools below have been developed by third parties and are provided here as a convenience to the community. Appearance on this page does not imply an endorsement of their full conformance to the SMT-LIB standard. Please contact each utility's authors directly for questions, bug reports, or other requests.

Editing

- Syntax highlighting for [VIM](#).
- Syntax highlighting mode [TextMate](#) and [Sublime](#).
- [Emacs mode](#) to edit and run SMT-LIB 2 scripts.

Parsing

- An [ANTLR grammar](#) of SMT-LIB 2.6 scripts.
- [jsMTLIB](#), a suite of Java tools for parsing and type-checking SMT-LIB 2 scripts, and translating them to the input languages of some non-SMT-LIB-conforming solvers.
- An open source [generic lexer and parser](#) for SMT-LIB 2.0 scripts implemented in Flex and Bison and C99.
- A [Haskell library](#) for parsing and printing SMT-LIB 2 scripts.
- An SWI-Prolog [parser](#) for SMT-LIB 2.6 scripts.
- An OCaml [parser](#) for SMT-LIB 2.0 scripts (not maintained).

[Home](#)

[About](#)

[News](#)

[Standard](#)

[Language](#)

[Theories](#)

[Logics](#)

[Examples](#)

[Benchmarks](#)

[Software](#)

[Solvers](#)

[Utilities](#)

[Contact](#)

[Related](#)

[Credits](#)

Calibrado de programas lógicos difusos en Z3 (I)

```
0.3 → good_hotel(hydro)  ~>*  #@s2((0.6 &godel #s1), 0.16)
0.4 → good_hotel(ritz)   ~>*  #@s2(0.8, 0.0)
```

Calibrado de programas lógicos difusos en Z3 (I)

```
0.3 → good_hotel(hydro)  ~>*  #@s2((0.6 &godel #s1), 0.16)
0.4 → good_hotel(ritz)   ~>*  #@s2(0.8, 0.0)
```

- 1 Traducir (manualmente) los retículos a SMT-LIB

Calibrado de programas lógicos difusos en Z3 (I)

```
0.3 → good_hotel(hydro) ~>* #@s2((0.6 &godel #s1), 0.16)
0.4 → good_hotel(ritz) ~>* #@s2(0.8, 0.0)
```

- 1 Traducir (manualmente) los retículos a SMT-LIB
- 2 Declarar las constantes simbólicas como variables:
 - `(declare-const deviation! Real)`
 - `vs1 → (declare-const sym!td!0!s1 Real)`
 - `@s2 → (declare-const sym!agr!2!s2 String)`

Calibrado de programas lógicos difusos en Z3 (I)

```
0.3 → good_hotel(hydro)  ⇨*  #@s2((0.6 &godel #s1), 0.16)
0.4 → good_hotel(ritz)   ⇨*  #@s2(0.8, 0.0)
```

- 1 Traducir (manualmente) los retículos a SMT-LIB
- 2 Declarar las constantes simbólicas como variables:
 - (declare-const deviation! Real)
 - $v^{s1} \rightarrow$ (declare-const sym!td!0!s1 Real)
 - $@^{s2} \rightarrow$ (declare-const sym!agr!2!s2 String)
- 3 Calcular las sfca's de los objetivos de los casos de prueba:

```
(assert (= deviation! (+
  (lat!distance 0.3
    (call!sym!agr!2 sym!agr!2!s2
      (lat!and!godel!2 0.6 sym!td!0!s1) 0.16)))
  (lat!distance 0.4
    (call!sym!agr!2 sym!agr!2!s2 0.8 0))))))
```

- 4 Minimizar la desviación y obtener el modelo:
(`minimize deviation!`)
(`check-sat`)
(`get-model`)

- 4 Minimizar la desviación y obtener el modelo:
(`minimize` deviation!)
(`check-sat`)
(`get-model`)
- 5 Ejecutar el guión en Z3 y analizar el resultado para recoger los valores de la mejor sustitución simbólica:

```
sat  
(model  
  (define-fun sym!td!0!s1 () Real 0.43)  
  (define-fun deviation! () Real 0.0)  
  (define-fun sym!agr!2!s2 () String "agr_aver"))
```

Calibrado de programas lógicos difusos en Z3 (III)

Fuzzy Aggregators and Similarities Into a Logic Language <http://dectau.uclm.es/fasill>


Edit

Manage topics

● Prolog 54.1% ● PHP 31.7% ● JavaScript 6.0% ● TeX 5.7% ● SMT 1.6% ● CSS 0.8% ● Shell 0.1%

Branch: master ▾ New pull request

Create new file Upload files Find file Clone or download ▾

 jariazavalverde Update unit.lat.smt2 Latest commit 2d12ac3 29 days ago

doc	fixed decimal answers with interrogation (?)	29 days ago
lattices	Update unit.lat.smt2	29 days ago
logo	fixed decimal answers with interrogation (?)	29 days ago
sample	fixed decimal answers with interrogation (?)	29 days ago
src	fixed decimal answers with interrogation (?)	29 days ago
test	fixed decimal answers with interrogation (?)	29 days ago
www	fixed decimal answers with interrogation (?)	29 days ago
.gitattributes	fixed decimal answers with interrogation (?)	29 days ago
.gitignore	fixed decimal answers with interrogation (?)	29 days ago
LICENSE	fixed decimal answers with interrogation (?)	29 days ago
README.md	fixed decimal answers with interrogation (?)	29 days ago
bibliography.bib	fixed decimal answers with interrogation (?)	29 days ago
fasill	added command line arguments	3 months ago
install.sh	added command :license	3 months ago

Calibrado de programas lógicos difusos en Z3 (IV)

Running

Tuning

Test cases

```
1 0.4 -> good_hotel(hydropolis).  
2 0.6 -> good_hotel(ritz).  
3
```

FASILL – Thresholded method ▾

FASILL – Thresholded method

SMT – Boolean lattice

SMT – Real lattice

SMT – Unit lattice

Tune

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado**
- 6 Conclusiones y trabajo futuro

Equivalencia de circuitos combinacionales (I)

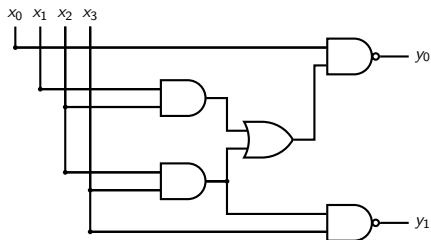
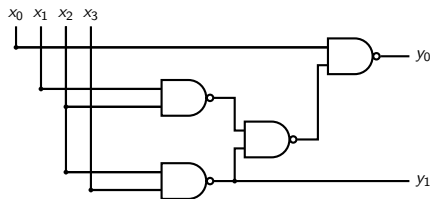
- Comprobar la equivalencia de dos circuitos combinacionales

Equivalencia de circuitos combinacionales (I)

- Comprobar la equivalencia de dos circuitos combinacionales
- Problema de vital importancia en el campo de la verificación de circuitos digitales

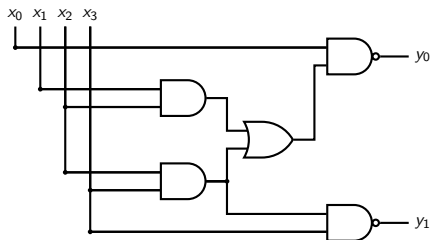
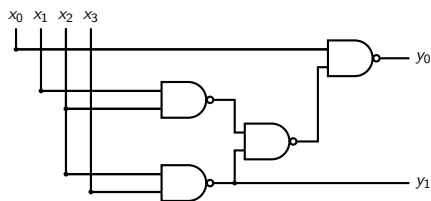
Equivalencia de circuitos combinatoriales (I)

- Comprobar la equivalencia de dos circuitos combinatoriales
- Problema de vital importancia en el campo de la verificación de circuitos digitales



Equivalencia de circuitos combinatoriales (I)

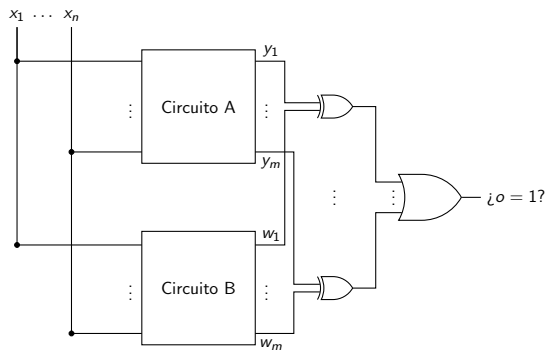
- Comprobar la equivalencia de dos circuitos combinatoriales
- Problema de vital importancia en el campo de la verificación de circuitos digitales



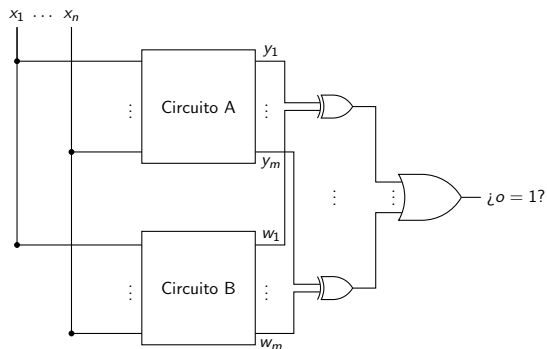
$a([X_0, X_1, X_2, X_3], [Y_0, Y_1]) :-$

```
truth_degree(@not('&'(X0, @not('&'(
    @not('&'(X1, X2)), @not('&'(X2, X3))))), Y0),
truth_degree(@not('&'(X2, X3)), Y1).
```

Equivalencia de circuitos combinatoriales (II)

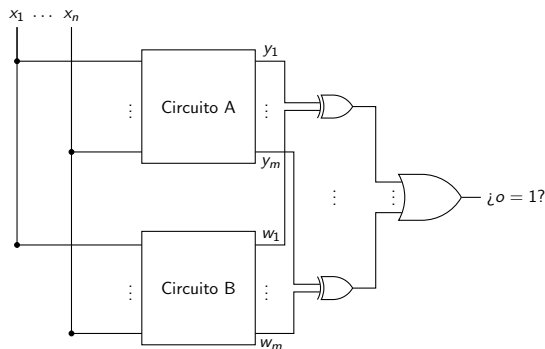


Equivalencia de circuitos combinacionales (II)



```
miter(Ca, Cb, Xs) :-  
    call(Ca, Xs, Ys),  
    call(Cb, Xs, Ws),  
    zip_xor(Ys, Ws, XOR),  
    fold_or(XOR, OR),  
    OR.
```

Equivalencia de circuitos combinacionales (II)



```
miter(Ca, Cb, Xs) :-  
  call(Ca, Xs, Ys),  
  call(Cb, Xs, Ws),  
  zip_xor(Ys, Ws, XOR),  
  fold_or(XOR, OR),  
  OR.
```

```
true -> miter(a, b, [#x0,#x1,#x2,#x3]).
```

Equivalencia de circuitos combinacionales (III)

Tabla: Tiempo de ejecución (en milisegundos) del algoritmo de calibrado en FASILL y Z3 para la equivalencia de circuitos combinacionales en función del número de entradas

# Entradas	FASILL	Z3
4	45	36
5	930	37
6	2260	40
7	5670	41
8	12200	42
9	29340	43
10	65480	45

Regresión lineal (I)

- Modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente Y y las variables independientes X_i :

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

Regresión lineal (I)

- Modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente Y y las variables independientes X_i :

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

- Los grillos hacen chirridos. A veces más rápido, a veces más lento. **Pregunta:** ¿La temperatura afecta a la frecuencia de chirrido de los grillos?



Regresión lineal (I)

- Modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente Y y las variables independientes X_i :

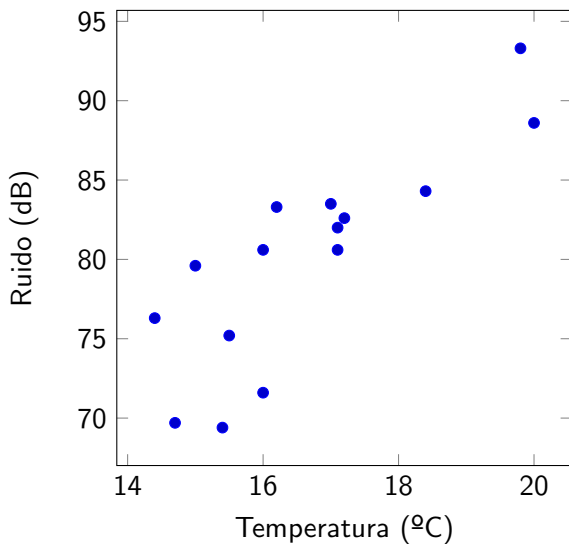
$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$$

- Los grillos hacen chirridos. A veces más rápido, a veces más lento. **Pregunta:** ¿La temperatura afecta a la frecuencia de chirrido de los grillos?
- Un viejo estudio midió la frecuencia de los chirridos (pulsos por segundo) 15 veces, a diferentes temperaturas:

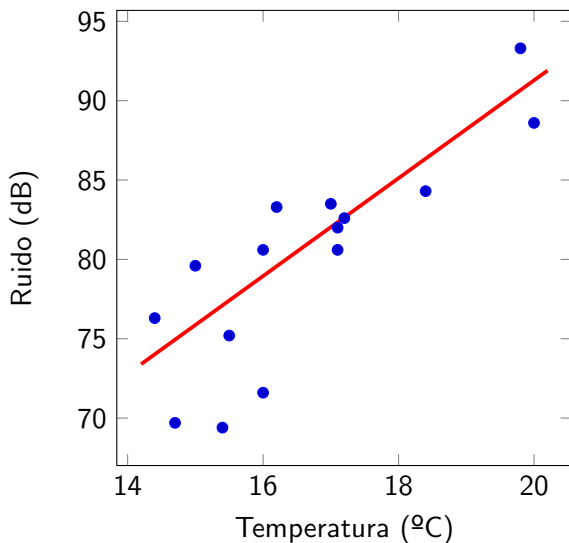


Temp.	20.0	16.0	19.8	18.4	17.1	15.5	14.7	17.1
Chirridos	88.6	71.6	93.3	84.3	80.6	75.2	69.7	82.0
Temp.	15.4	16.2	15.0	17.2	16.0	17.0	14.4	
Chirridos	69.4	83.3	79.6	82.6	80.6	83.5	76.3	

Regresión lineal (II)



Regresión lineal (II)



$$y = 25.23 + 3.29x$$

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$$

- Expresaremos el modelo de regresión como un programa FASILL con una única cláusula:

```
chirps(Temperature) <-  
  #b0 |add (#b1 &prod Temperature).
```

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$$

- Expresaremos el modelo de regresión como un programa FASILL con una única cláusula:

```
chirps(Temperature) <-  
  #b0 |add (#b1 &prod Temperature).
```

- Calibraremos este programa introduciendo un caso de prueba por cada muestra del conjunto de datos:

```
88.6 -> chirps(20.0).
```

```
71.6 -> chirps(16.0).
```

```
93.3 -> chirps(19.8).
```

...

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_n X_n$$

- Expresaremos el modelo de regresión como un programa FASILL con una única cláusula:

```
chirps(Temperature) <-  
  #b0 |add (#b1 &prod Temperature).
```

- Calibraremos este programa introduciendo un caso de prueba por cada muestra del conjunto de datos:

```
88.6 -> chirps(20.0).
```

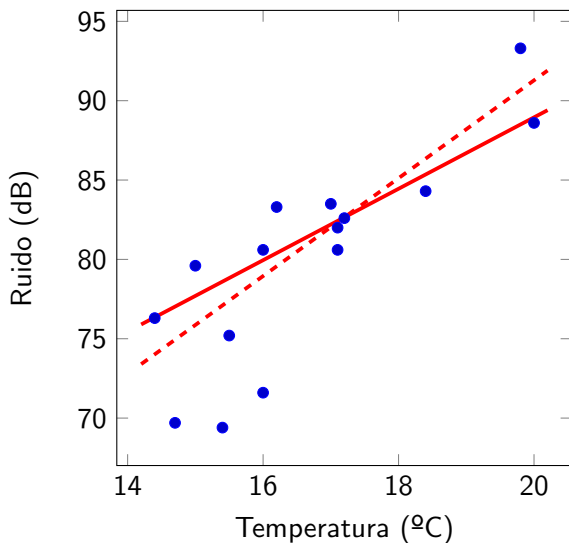
```
71.6 -> chirps(16.0).
```

```
93.3 -> chirps(19.8).
```

...

- $\{\beta_0/42.92, \beta_1/2.28\}$

Regresión lineal (IV)



$$y = 42.92 + 2.28x$$

Regresión lineal (V)

Tabla: Tiempo de ejecución (en segundos) del algoritmo de calibrado en Z3 para regresión lineal en función del número de variables explicativas y casos de prueba

		# Variables explicativas				
		1	2	3	4	5
# Casos de prueba	20	0.35	1.55	6.27	11.38	33.48
	30	2.22	6.45	27.05	49.40	72.34
	40	6.01	25.32	107.68	165.52	378.15
	50	14.04	43.60	184.35	583.45	1547.56
	60	29.80	150.85	619.22	1094.20	3319.37
	70	40.30	213.32	794.55	2452.84	7532.63
	80	51.08	311.07	1058.35	4261.00	17317.25
	90	106.15	625.05	1658.60	7688.81	40313.90
	100	215.10	874.47	3189.72	11204.84	85873.49

- 1 Programación lógica difusa
- 2 Calibrado de programas lógicos difusos
- 3 El problema de satisfacibilidad
- 4 Calibrado de programas lógicos difusos con SMT
- 5 Casos de uso del calibrado
- 6 Conclusiones y trabajo futuro**

Conclusiones

- Formalismos subyacentes de la lógica difusa, de la programación lógica difusa y de SAT/SMT, así como de las técnicas de calibrado simbólico implementadas

- Formalismos subyacentes de la lógica difusa, de la programación lógica difusa y de SAT/SMT, así como de las técnicas de calibrado simbólico implementadas
- Diseño de una nueva técnica de calibrado con SMT:
 - No es necesario discretizar los elementos del retículo
 - Es más eficiente para determinadas teorías

- Formalismos subyacentes de la lógica difusa, de la programación lógica difusa y de SAT/SMT, así como de las técnicas de calibrado simbólico implementadas
- Diseño de una nueva técnica de calibrado con SMT:
 - No es necesario discretizar los elementos del retículo
 - Es más eficiente para determinadas teorías
- Aplicación de las técnicas de calibrado a problemas no triviales:
 - Circuitos digitales
 - Regresión lineal
 - Web semántica

- Formalismos subyacentes de la lógica difusa, de la programación lógica difusa y de SAT/SMT, así como de las técnicas de calibrado simbólico implementadas
- Diseño de una nueva técnica de calibrado con SMT:
 - No es necesario discretizar los elementos del retículo
 - Es más eficiente para determinadas teorías
- Aplicación de las técnicas de calibrado a problemas no triviales:
 - Circuitos digitales
 - Regresión lineal
 - Web semántica
- Ampliación de las capacidades del lenguaje FASILL:
 - Implementación de la nueva técnica de calibrado con SMT
 - Incorporación del calibrado al entorno FLOPER online

- Formalismos subyacentes de la lógica difusa, de la programación lógica difusa y de SAT/SMT, así como de las técnicas de calibrado simbólico implementadas
- Diseño de una nueva técnica de calibrado con SMT:
 - No es necesario discretizar los elementos del retículo
 - Es más eficiente para determinadas teorías
- Aplicación de las técnicas de calibrado a problemas no triviales:
 - Circuitos digitales
 - Regresión lineal
 - Web semántica
- Ampliación de las capacidades del lenguaje FASILL:
 - Implementación de la nueva técnica de calibrado con SMT
 - Incorporación del calibrado al entorno FLOPER online
- Publicaciones presentadas en congresos internacionales:
 - **SUM'17, ESCIM'17, IEEE SSCI'17, IEEE SSCI'18, PROLE'18**
 - **CORE B: LOPSTR'16, RuleML+RR'17**
 - **CORE A: FUZZIEEE'19** (sometido)

- Calibrar programas con constantes simbólicas en las relaciones de similitud

- Calibrar programas con constantes simbólicas en las relaciones de similitud
- Considerar nuevas teorías/retículos para más aplicaciones

- Calibrar programas con constantes simbólicas en las relaciones de similitud
- Considerar nuevas teorías/retículos para más aplicaciones
- Combinar las técnicas de calibrado y desplegado para mejorar las prestaciones

- Calibrar programas con constantes simbólicas en las relaciones de similitud
- Considerar nuevas teorías/retículos para más aplicaciones
- Combinar las técnicas de calibrado y desplegado para mejorar las prestaciones
- Aplicar el calibrado a la gestión de grandes volúmenes de datos de redes sociales mediante consultas FSA-SPARQL

⊖ FASILL based tuning of FSA-SPARQL queries

Example 1 Example 2 Example 3

? FSA-SPARQL Query

```
1 PREFIX movie: <http://www.movies.org#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 PREFIX f: <http://www.fuzzy.org#>
5 PREFIX l: <http://www.lattice.org#>
6 SELECT ?movie ?Rank
7 WHERE {
8   ?Movie f:type (movie:genre movie:Thriller ?c) .
9   ?Movie f:type (movie:quality movie:Good ?r) .
10  BIND(1-WMEAN(0.5,?r,?c) as ?Rank)
11  FILTER (?Rank > '#s2' ) }
```

movies.rdf ▾

↵ Compile

</> FASILL Program

```
1 p(MOVIE,RANK):-
2   rdf(MOVIE,'http://www.fuzzy.org#type',X0),
3   rdf(X0,'http://www.fuzzy.org#onProperty','http://www.movies.org#genre'),
4   rdf(X0,'http://www.w3.org/1999/02/22-rdf-syntax-ns#type','http://www.movies.org#Thriller'),
5   rdf(X0,'http://www.fuzzy.org#truth',C),
6   rdf(MOVIE,'http://www.fuzzy.org#type',X1),
7   rdf(X1,'http://www.fuzzy.org#onProperty','http://www.movies.org#quality'),
8   rdf(X1,'http://www.w3.org/1999/02/22-rdf-syntax-ns#type','http://www.movies.org#Good'),
9   ...
```

🔍 Test cases

```
1 0.85^^_ -> p('http://www.movies.org#The_American', TD) & TD.
2 0.2^^_ -> p('http://www.movies.org#The_descendants', TD) & TD.
```

↵ Tune

Q Symbolic substitution

```
1 best symbolic substitution: {#s2/^^(0.5, 'http://www.w3.org/2001/XMLSchema#decimal')}
2 deviation: 0.3
3 execution time: 1438 milliseconds
```

FIN

?- question(X), answer(X,Y), tell(Y).