Deducción automática

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Prehistoria de la deducción automática

- *1666: Leibniz: Combinatoria*
- *1847: Boole: cálculo proposicional*
- *1869: Jevons: máquina lógica*
- *1879: Frege: cálculo de predicados*
- 1894: Peano: notación lógica
- *1910: Whitehead y Russell: logicismo*
- 1928: Hilbert y Ackermann: libro de lógica
- 1931: Gödel: incompletitud
- 1936: Tarski: concepto de verdad
- 1936: Church: indecidibilidad
- 1936: Turing: máquina universal
- 1943: Post: Reglas de producción

Historia de la deducción automática

- **1954: M. Davis: Demostrador aritmético**
- ***1956: Newell, Shaw y Simon: lógico teórico***
- **1956: Darmouth: Nacimiento de la IA.**
- **1958: McCarthy: lenguaje LISP**
- **1959: Gelernter: máquina geométrica**
- **1960: Wang: demostrador Gentzen**
- ***1960: Gilmore: demostrador Skolem***
- **1960: Davis y Putnam: procedimiento DP**
- ***1964: Robinson: resolución y unificación***
- **1964: Wos y otros: preferencia unidad**
- **1964: Wos: resolución unidad**
- **1964: Wos y otros: estrategia del soporte**
- **1965: Robinson: hiper–resolución**
- **1965: Robinson: subsunción**
- ***1967: Wos y otros: demodulación***

Historia de la deducción automática

- **1968: Loveland: resolución lineal**
- **1969: Guard y otros: SAM (lema)**
- **1969: Green: QA3, obtención de respuestas**
- **1970: Chang: resolución por entradas**
- ***1970: Wos y Robinson: paramodulación***
- ***1970: Knuth y Bendix: reescritura***
- **1970: Bruijin: Automath**
- **1971: Fikes y Nilsson: STRIPS: planificación**
- **1972: AURA (Automated reasoning assistant)**
- **1972: Kowalski: programación lógica**
- ***1973: Colmerauer: Prolog***
- ***1973: Boyer y Moore: lógica computacional***
- **1974: Comienza CADE**
- **1976: Wos y otros: resolución UR**
- **1976: Wos y otros: estrategia de pesos**

Historia de la deducción automática

- **1979: Boyer y Moore: factorización prima**
- **1982: ANL: ITP (Interactive Theorem Prover)**
- **1983: Rusinoff: teorema de Wilson en BM**
- **1984: AMS "ATP (after 25 years)"**
- **1984: Boyer y Moore: completitud de Lisp**
- **1986: Shankar: teorema de incompletitud**
- ***1988: McCune: OTTER***
- **1992: Quaife: teorías con OTTER**
- **1992: Kunen–OTTER: ax. simples de grupos**
- ***1992: Boyer y Moore: Nqthm–1992***
- **1994: McCune: MACE**
- **1995: Kunen–OTTER: grupos de exponente 4**
- ***1996: McCune–OTTER: Pb. de Robbins***

Métodos algebraicos en D.A.

- **Lenguaje de lógicas polivalentes**
  - n, s son enteros positivos
  - $\{X_1, \ldots, X_n\}$ *variables proposicionales*
  - $\{f_1, \ldots, f_s\}$ *conectivas*
  - $\delta : \{1, \ldots, s\} \to \mathbb{N}$ *aridad*
- **Fórmulas proposicionales:** $\mathbb{P}(X_1, \ldots, X_n)$
  - $X_i \in \mathbb{P}(X_1, \ldots, X_n)$
  - Si $j \in \{1, \ldots, s\}$ y $A_1, \ldots, A_{\delta(j)} \in \mathbb{P}(X_1, \ldots, X_n)$, entonces $f_j(A_1, \ldots, A_{\delta(j)}) \in \mathbb{P}(X_1, \ldots, X_n)$
- **Valores de verdad**
  - p es un número primo
  - $\mathbb{Z}_p$ *valores de verdad*
  - 0 *falso*
  - 1 *verdadero*
- **Funciones de verdad**
  - $H_j : \mathbb{Z}_p^{\delta(j)} \to \mathbb{Z}_p$

Métodos algebraicos en D.A.

- **Semántica de lógicas polivalentes**
  - $v : \{X_1, \ldots, X_n\} \to \mathbb{Z}_p$ **valoración**
  - $\hat{v} : \mathbb{P}(X_1, \ldots, X_n) \to \mathbb{Z}_p$
    $$\hat{v}(A) = \begin{cases} v(A), & \text{si } A \in \{X_1, \ldots, X_n\}; \\ H_j(\hat{v}(A_1), \ldots, \hat{v}(A_{\delta(j)})), & \text{si } A \text{ es } f_j(A_1, \ldots, A_{\delta(j)}). \end{cases}$$
  - $\{B_1, \ldots, B_m\} \models A$ **si para toda valoración** v,
    $\hat{v}(B_1) = \cdots = \hat{v}(B_m) = 1 \implies \hat{v}(A) = 1$

Métodos algebraicos en D.A.

- **Fórmulas y polinomios**
  - $\theta : \mathbb{P}(X_1, \ldots, X_n) \to \mathbb{Z}_p[X_1, \ldots, X_n]$
    $\theta(A) = \begin{cases} A, & \text{si } A \in \{X_1, \ldots, X_n\}; \\ \theta_j(\theta(A_1), \ldots, \theta(A_{\delta(j)})), & \text{si } A = f_j(A_1, \ldots, A_{\delta(j)}) \end{cases}$
  - $\theta_j : \mathbb{Z}_p[X_1, \ldots, X_n]^{\delta(j)} \to \mathbb{Z}_p[X_1, \ldots, X_n]$
    $\theta_j(q_1, \ldots, q_{\delta(j)}) = \sum_{i_1, \ldots, i_{\delta(j)}} H_j(i_1, \ldots, i_{\delta(j)}) L_{i_1}(q_1) \ldots L_{i_{\delta(j)}}(q_{\delta(j)})$
    $L_0(q) = 1 - q^{p-1}$
    $L_i(q) = L_0(q - i)$ para $i \in \{1, \ldots, p-1\}$

Métodos algebraicos en D.A.

- **Teorema: Sean $A_0, A_1, \ldots, A_m \in \mathbb{P}(X_1, \ldots, X_n)$. Son equivalentes:**
  - **A_0 es consecuencia de $\{A_1, \ldots, A_m\}$**
  - **$\theta(A_0) - 1$ pertenece al ideal generado por $\{\theta(A_1) - 1, \ldots, \theta(A_m) - 1, X_1^p - X_1, \ldots, X_n^p - X_n\}$**
- **Algoritmo de deducción**
  - ***Entrada*: Una fórmula A_0 y un conjunto finito $\Gamma = \{A_1, \ldots, A_m\}$ de fórmulas**
  - ***Salida*: `SÍ`, si $\Gamma \models A_0$; `NO`, en caso contrario**
  - ***Procedimiento*:**
    $F := \{\theta(A_1) - 1, \ldots, \theta(A_m) - 1, X_1^p - X_1, \ldots, X_n^p - X_n\}$
    $G := BG(F - \{0\})$
    $q := FN(\theta(A_0) - 1, G)$
    si $q = 0$, devolver `SÍ`
    en otro caso devolver `NO`
- **Implementación: Lisp, Reduce, Maple, CoCoA**
- **Aplicaciones:**
  - **Compilación de bases de conocimiento**
  - **Verificación de bases de conocimiento**

Deducción automática en álgebra

- **Problema elemental de grupos**
  - **Teorema:** ***Sea G un grupo y e su elemento neutro. 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**
      $(\forall x)(\forall y)[x.y = y.x]$

Deducción automática en álgebra

- **Entrada** `grupos-1a.in`

```
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.

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

Deducción automática en álgebra

- **Uso de OTTER**

```
otter <grupos-1a.in
```

- **Prueba**

```
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.
```

Deducción automática en álgebra

- **Mejora con demoduladores**
  - **Entrada** `grupos-1b.in`

```
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
end_of_list.

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

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.

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

Deducción automática en álgebra

- **Prueba**

```
5 [] (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,7.1.1,5.1.2,demod,13,13,13] x*y*x*y=e.
19 [para_from,7.1.1,5.1.1.1,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.
```

Deducción automática en aritmética

- Teorema: ***Existen infinitos números primos***
  - Demostración usual:

    Hemos de probar que para cada número natural, x, existe un número primo y que es mayor que x. Lo haremos por reducción al absurdo.

    Supongamos que existe un número natural a tal que cualquier número primo es menor o igual que a. Sea x un número arbitrario. Puesto que el menor factor primo de $1 + x!$ es primo, debe ser menor o igual que a y, por tanto, divide al factorial de a. En particular, el menor factor primo de $1 + a!$ divide al factorial de a y, puesto que también divide a $1 + a!$, tiene que dividir a la diferencia y, por tanto, debe ser igual a 1. De donde se sigue que $a! = 0$. Lo que es una contradicción.

Deducción automática en aritmética

- Símbolos no lógicos:

`fact (x)`	factorial de x
`mfp (x)`	menor factor primo de x, si $x > 1$ y x en caso contrario
`PR (x)`	x es un número primo
`x < y`	x es menor que y
`DIV (x,y)`	x divide a y

- Hipótesis necesarias:

H1 Si un número divide a una suma y a uno de los sumandos, entonces divide al otro.

H2 El menor factor primo de un número divide a dicho número.

H3 Si el menor factor primo de un número es 1, entonces dicho número es 1.

H4 El menor factor primo de $1 + x$ es distinto de 0.

H5 El menor factor primo de $1 + x!$ es primo.

H6 Si $y \neq 0$ e $y \leq x$, entonces y divide a $x!$.

H7 El factorial de cualquier número es distinto de 0.

Deducción automática en aritmética

- **Cláusulas de las hipótesis**

```
list(usable).
DIV(x, y + z), DIV(x, z) -> DIV(x, y).   % H1
-> DIV(mfp(x), x).                       % H2
mfp(x) = 1 -> x = 1.                     % H3
mfp(1 + x) = 0 ->.                       % H4
-> PR(mfp(1 + fact(x))).                 % H5
-> y = 0, x < y, DIV(y, fact(x)).        % H6
fact(x) = 0 ->.                          % H7
end_of_list.
```

- **Tesis**

  Tesis: $\forall x \exists y (x < y \wedge PR(y))$

  Negación: $\exists x \forall y \neg (x < y \wedge PR(y))$

- **Cláusula de la tesis**

```
list(sos).
a < y , PR(y) ->.
end_of_list.
```

- **Demoduladores**

```
list(demodulators).
-> (x + y = x) = (y = 0).
-> DIV(x,1) = (x = 1).
end_of_list.
```

- **Regla de inferencia**

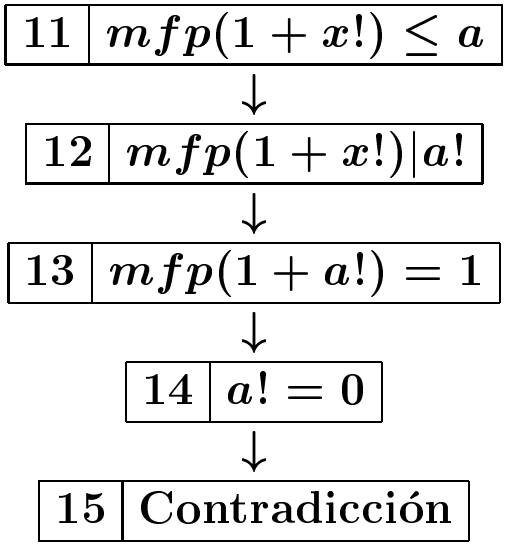
```
set(ur_res).
```

Deducción automática en aritmética

- **Prueba con OTTER**

```
1 [] DIV(x,y+z), DIV(x,z) -> DIV(x,y).
2 []  -> DIV(mfp(x),x).
3 [] mfp(x)=1 -> x=1.
4 [] mfp(1+x)=0 -> .
5 []  -> PR(mfp(1+fact(x))).
6 []  -> y=0, x<y, DIV(y,fact(x)).
7 [] fact(x)=0 -> .
8 []  -> (x+y=x)= (y=0).
9 []  -> DIV(x,1)= (x=1).
10 [] a<y, PR(y) -> .
11 [ur,10,5] a<mfp(1+fact(x)) -> .
12 [ur,11,6,4]  -> DIV(mfp(1+fact(x)),fact(a)).
13 [ur,12,1,2,demod,9]  -> mfp(1+fact(a))=1.
14 [ur,13,3,demod,8]  -> fact(a)=0.
15 [binary,14.1,7.1]  -> .
```

- **Grafo de la prueba:**



Problema de Robbins

- **Axiomas de Huntington (1933):**
  - **(A)** $\mathbf{(x + y) + z = x + (y + z)}$
  - **(C)** $\mathbf{x + y = y + x}$
  - **(H)** $\mathbf{n(n(x) + y) + n(n(x) + n(y)) = x}$
- **Axioma de Robbins (1933):**
  - **(R)** $\mathbf{n(n(n(y) + x) + n(x + y)) = x}$
- **Teorema: (A)+(C)+(H) $\Longrightarrow$ (R)**
- **Problema de Robbins: (A)+(C)+(R) $\Longrightarrow$ (H)**
- **Lemas (Winkler, 1990):**
  - **(A)+(C)+(R)+**$\mathbf{(\exists c)(\exists d)[c + d = c] \Longrightarrow}$ **(H)**
  - **(A)+(C)+(R)+**$\mathbf{(\exists c)(\exists d)[n(c + d) = n(c)] \Longrightarrow}$ **(H)**
- **Teorema (McCune, 1996):**
  - **(A)+(C)+(R)** $\Longrightarrow \mathbf{(\exists c)(\exists d)[c + d = c]}$
- **Entrada a EQP**

```
n(n(n(y)+x)+n(x+y))=x.
x+y != x.
n(x+y) != n(x).
```

Problema de Robbins

```
2      []              -(n(x+y)=n(x)).
3      []              n(n(n(x)+y)+n(x+y))=y.
5      [3,3]           n(n(n(x+y)+n(x)+y)+y)=n(x+y).
6      [3,3]           n(n(n(n(x)+y)+x+y)+y)=n(n(x)+y).
24     [6,3]           n(n(n(n(x)+y)+x+2y)+n(n(x)+y))=y.
47     [24,3]          n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+z)+n(y+z))=z.
48     [24,3]          n(n(n(n(x)+y)+n(n(x)+y)+x+2y)+y)=n(n(x)+y).
146    [48,3]          n(n(n(n(x)+y)+n(n(x)+y)+x+3y)+n(n(x)+y))=y.
250    [47,3]          n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z)=n(y+z).
996    [250,3]         n(n(n(n(n(n(x)+y)+x+2y)+n(n(x)+y)+n(y+z)+z)+z+u)+n(n(y+z)+u))=u.
16379  [5,996,3]       n(n(n(n(x)+x)+3x)+x)=n(n(x)+x).
16387  [16379,3]       n(n(n(n(n(x)+x)+3x)+x+y)+n(n(n(x)+x)+y))=y.
16388  [16379,3]       n(n(n(n(x)+x)+4x)+n(n(x)+x))=x.
16393  [16388,3]       n(n(n(n(x)+x)+n(n(x)+x)+4x)+x)=n(n(x)+x).
16426  [16393,3]       n(n(n(n(n(x)+x)+n(n(x)+x)+4x)+x+y)+n(n(n(x)+x)+y))=y.
17547  [146,16387]     n(n(n(n(n(x)+x)+n(n(x)+x)+4x)+n(n(n(x)+x)+3x)+x)+x)
                       =n(n(n(x)+x)+n(n(x)+x)+4x).
17666  [24,16426,17547] n(n(n(x)+x)+n(n(x)+x)+4x)=n(n(n(x)+x)+3x).
```

n(c+d) = n(c), c = n(n(x)+x)+3x, d = n(n(x)+x)+x

D.A. en la lógica de B.M.

- **Lógica de Boyer y Moore**
  - **Sintaxis: Lisp**
  - **Lógica de primer orden sin cuantificadores, con igualdad**
  - **Definición (terminación)**
  - **Demostradores: Thm, Nqthm, ACL2**
  - **Tipos predefinidos:**
    *** booleanos,**
    *** átomos,**
    *** naturales,**
    *** listas**
  - **Procesos:**
    *** simplificación**
    *** eliminación de destructores**
    *** fertilización cruzada**
    *** generalización**
    *** eliminación de irrelevancias**
    *** inducción**

D.A. en la lógica de B.M.

- **Ejemplo con Nqthm**
  - **Definición del mcd**

```
(defn mcd (x y)
  (if (zerop x) (if (numberp y) y 0)
      (if (zerop y) x
          (if (lessp x y) (mcd x (difference y x))
                          (mcd (difference x y) y))))
  ((lessp (plus x y))))
```

  - **Aceptación de la definición**

```
    Linear arithmetic, the lemmas CORRECTNESS-OF-CANCEL-LESSP-PLUS,
PLUS-DIFFERENCE-ARG2, CORRECTNESS-OF-CANCEL-DIFFERENCE-PLUS, and
PLUS-DIFFERENCE-ARG1, and the definitions of NOT, FIX, ZEROP, PLUS, and EQUAL
establish that the measure (PLUS X Y) decreases according to the well-founded
relation LESSP in each recursive call.  Hence, MCD is accepted under the
principle of definition.  Observe that (NUMBERP (MCD X Y)) is a theorem.
```

D.A. en la lógica de B.M.

- **Lema 1 de corrección**

```
(prove-lemma mcd-es-divisor-comun (rewrite)
  (and (equal (remainder x (mcd x y)) 0)
       (equal (remainder y (mcd x y)) 0)))
```

- **Prueba del lema 1**

```
We will induct according to the following scheme:
       (AND (IMPLIES (AND (ZEROP X) (NUMBERP Y))
                     (p Y X))
            (IMPLIES (AND (ZEROP X) (NOT (NUMBERP Y)))
                     (p Y X))
            (IMPLIES (AND (NOT (ZEROP X)) (ZEROP Y))
                     (p Y X))
            (IMPLIES (AND (NOT (ZEROP X))
                          (NOT (ZEROP Y))
                          (LESSP X Y)
                          (p (DIFFERENCE Y X) X))
                     (p Y X))
            (IMPLIES (AND (NOT (ZEROP X))
                          (NOT (ZEROP Y))
                          (NOT (LESSP X Y))
                          (p Y (DIFFERENCE X Y)))
                     (p Y X))).

The above induction scheme generates the following
ten new goals:

Case 10.(IMPLIES (AND (ZEROP X) (NUMBERP Y))
                 (EQUAL (REMAINDER X (MCD X Y)) 0)).
  This simplifies, rewriting with REMAINDER-OF-NON-NUMBER
  and expanding ZEROP, EQUAL, MCD, LESSP, NUMBERP,
  and REMAINDER, to: T.
```

D.A. en la lógica de B.M.

- **Lema 2 de corrección**

```
(prove-lemma divisor-comun-divide-mcd (rewrite)
   (implies (and (equal (remainder x z) 0)
                 (equal (remainder y z) 0))
            (equal (remainder (mcd x y) z) 0)))
```

- **Lema necesario para probar el Lema 2**

```
(prove-lemma distributividad-producto-mcd (rewrite)
   (equal (mcd (times x z) (times y z))
          (times z (mcd x y))))
```

Cálculos de D.A. en la lógica de B.M.

- **Formalización de cálculos proposicionales**
  - **Formalización de la sintaxis y semántica**
  - **Cálculos proposicionales: tableros, secuentes, resolución**
- **Formalización del retículo de términos**
  - **Retículo de términos**
  - **Problema de la unificación y antiunificación**
- **Formalización de cálculos ecuacionales**
  - **Reducciones entre términos**
  - **Confluencia**
  - **Sistemas de reescritura de términos**
  - **Lema pares críticos de Knuth–Bendix**
  - **Decidibilidad de sistemas de reescritura de términos**
- **Formalización de programación lógica**
  - **Formalización de resolución SLD**
- **Formalización de bases de Gröbner**
  - **Formalización de álgebra de polinomios**
  - **Cálculo de bases de Gröbner**