Presentación de Isabelle

. . . y otros sistemas de razonamiento

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

13 de Enero de 2010

Presentación de Isabelle . . . y otros sistemas de razonamiento

1. Razonamiento automático con OTTER/MACE
2. Razonamiento automático con ACL2
3. Razonamiento asistido por PVS
4. Razonamiento asistido por Isabelle/HOL/Isar
5. Comentarios finales

El sistema OTTER

- OTTER es un sistema de razonamiento automático para lógica de primer orden basado en resolución y paramodulación.
- Desarrollado por William McCune en el ANL (Argonne National Laboratory).
- Sucesor de anteriores demostradores de ANL: AURA e ITP.
- Antecesor de EQP y Prover9.

Ejemplo de razonamiento automático con OTTER

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

Ejemplo de razonamiento automático con OTTER

- Entrada `grupos.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.
b * a != a * b.
end_of_list.
```

Ejemplo de razonamiento automático con OTTER

- Uso de OTTER

```
otter <grupos.in
```

- Resultado:

```
---------------- PROOF ----------------------------------
1 []                                                b*a!=a*b.
3,2 []                                              e*x=x.
5,4 []                                              x*e=x.
10 []                                               (x*y)*z=x*(y*z).
13 []                                               x*x=e.
17 [para_into,10.1.1.1,13.1.1,demod,3,flip.1]       x*(x*y)=y.
23 [para_into,10.1.1,13.1.1,flip.1]                 x*(y*(x*y))=e.
33 [para_from,23.1.1,17.1.1.2,demod,5,flip.1]       x*(y*x)=y.
37 [para_from,33.1.1,17.1.1.2]                      x*y=y*x.
38 [binary,37.1,1.1]                                F.
------------ end of proof ----------------------------------
```

Búsqueda de modelos con MACE

- Problema: Buscar un grupo con conmutativo.
- Representación en MACE (`grupos2.in`):

```
op(400, xfy, *).
op(300, yf, ^).
set(auto2).

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
b * a != a * b.
end_of_list.
```

Búsqueda de modelos con MACE

- Búsqueda con MACE

```
mace2 -N10 -p < grupos2.in

======================= Model #1 at 0.17 seconds:

 * :
      | 0 1 2 3 4 5
    --+------------
    0 | 0 1 2 3 4 5
    1 | 1 0 3 2 5 4
    2 | 2 4 0 5 1 3
    3 | 3 5 1 4 0 2
    4 | 4 2 5 0 3 1
    5 | 5 3 4 1 2 0

 e: 0

 ^ :
        0 1 2 3 4 5
    ---------------
        0 1 2 4 3 5

 b: 1

 a: 2
end_of_model
```

Procesamiento del conocimiento matemático

- 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.

Extracción del conocimiento

- 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.

Representación del conocimiento en OTTER

- Representación en OTTER:

```
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.
```

Representación del conocimiento en OTTER

- 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

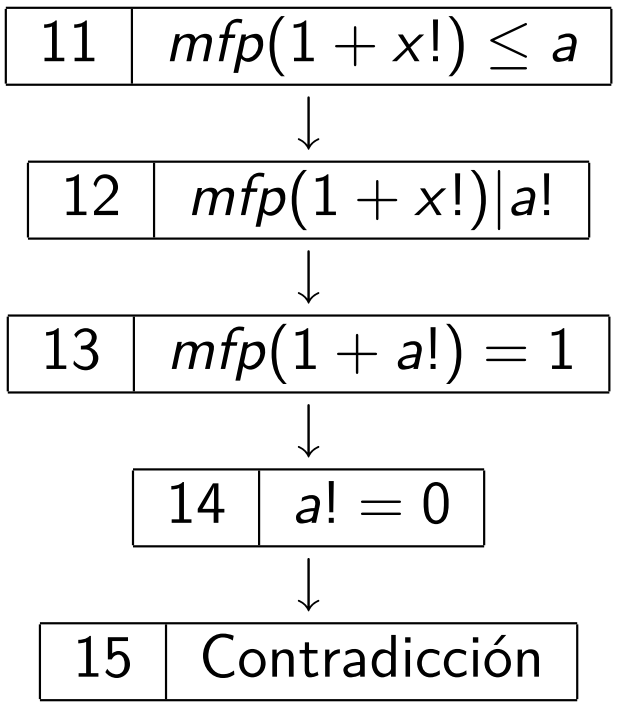
```
set(ur_res).
```

Procesamiento del conocimiento 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]  -> .
```

Correspondencia de la prueba de OTTER y la inicial

- Grafo de la prueba:



Presentación de Isabelle . . . y otros sistemas de razonamiento

1. Razonamiento automático con OTTER/MACE
2. Razonamiento automático con ACL2
3. Razonamiento asistido por PVS
4. Razonamiento asistido por Isabelle/HOL/Isar
5. Comentarios finales

El sistema ACL2

- **A C**omputacional **L**ogic for an **A**pplicative **C**ommon **L**isp.
- Desarrollado por Moore y Kaufmann en la Universidad de Texas.
- Sucesor de NQTHM de Boyer y Moore.
- Tres aspectos de ACL2:
  1. Lenguaje de programación.
  2. Lógica.
  3. Sistema de razonamiento automático.

ACL2 como lenguaje de programación

- Definición:

```
(defun concatena (x y)
  (if (endp x) y
      (cons (car x) (concatena (cdr x) y))))

(defun inversa (x)
  (if (endp x) nil
      (concatena (inversa (cdr x)) (list (car x)))))
```

- Evaluación:

```
ACL2 !> (concatena '(a b) '(c d e))
(A B C D E)
ACL2 !> (inversa '(a b c))
(C B A)
ACL2 !> (inversa (inversa '(a b c)))
(A B C)
```

ACL2 como lógica

- ACL2 es una lógica de primer orden con igualdad sin cuantificadores de la parte aplicativa de Common Lisp.
- Ejemplo de definiciones de la teoría de listas:

```
(defun true-listp (x)
  (if (consp x) (true-listp (cdr x))
                (eq x nil)))
```

- Ejemplo de axiomas de la teoría de lista:

```
(defaxiom car-cdr-elim
  (implies (consp x)
           (equal (cons (car x) (cdr x)) x)))
(defaxiom car-cons (equal (car (cons x y)) x))
(defaxiom cdr-cons (equal (cdr (cons x y)) y))
```

- Mediante `defun` se extiende la lógica de manera conservativa. Se necesitan condiciones de admisibilidad.

ACL2 como lógica: Admisibilidad

- Ejemplo 1:

```
ACL2 !> (defun f (x) y)

ACL2 Error in (DEFUN F ...): The body of F contains
a free occurrence of the variable symbol Y.
```

- Ejemplo 2:

```
ACL2 !> (defun f (x) (+ (f x) 1))

ACL2 Error in (DEFUN F ...): No :measure was
supplied with the definition of F. Our heuristics
for guessing one have not made any suggestions.
No argument of the function is tested along every
branch and occurs as a proper subterm at the same
argument position in every recursive call. You must
specify a :measure.
```

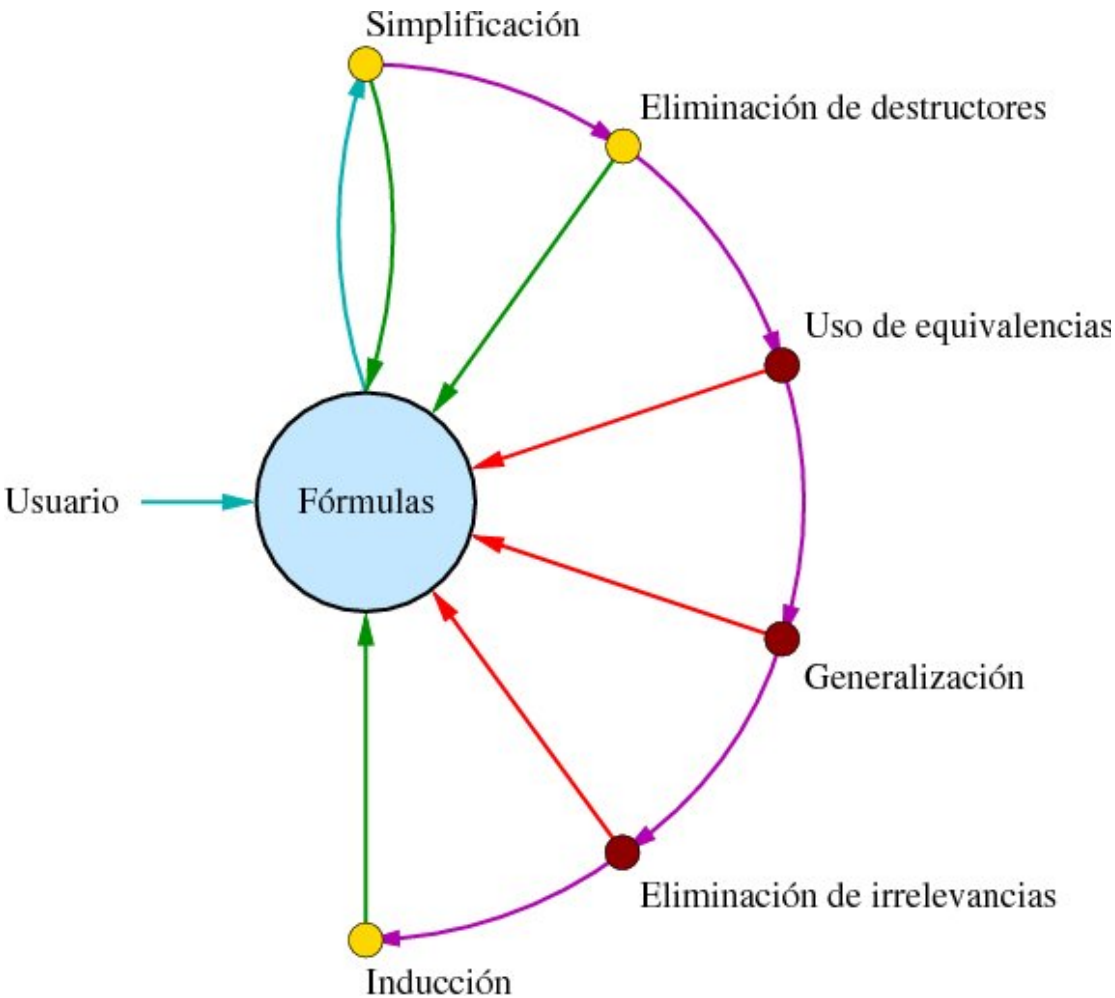
ACL2 como lógica: Admisibilidad

- Ejemplo 3:

```
ACL2 !> (defun inversa (x)
          (if (endp x)
              nil
              (concatena (inversa (cdr x))
                         (list (car x)))))

The admission of INVERSA is trivial, using the relation
E0-ORD-< (which is known to be well-founded on the
domain recognized by E0-ORDINALP) and the measure
(ACL2-COUNT X).  We observe that the type of INVERSA is
described by the theorem
(OR (CONSP (INVERSA X)) (EQUAL (INVERSA X) NIL)).
We used primitive type reasoning and the
:type-prescription rule CONCATENA.
```

ACL2 como SRA (sistema de razonamiento automático)



ACL2 como SRA: Inducción

```
ACL2 !> (defthm inversa-inversa
          (implies (true-listp x)
                   (equal (inversa (inversa x)) x)))

   Name the formula above *1.
   Perhaps we can prove *1 by induction. Two induction schemes
are suggested by this conjecture. These merge into one
derived induction scheme.
   We will induct according to a scheme suggested by (INVERSA
X). If we let (:P X) denote *1 above then the induction
scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
```

El papel del usuario

- Frecuentemente los primeros intentos de demostración de resultados no triviales fallan.
- Ello significa que el demostrador necesita demostrar lemas previos.
- Estos lemas se obtienen:
  - de una demostración preconcebida a mano o
  - del anális del fallo del intento de prueba.
- Por tanto, el papel del usuario es:
  - Formalizar la teoría en la lógica.
  - Implementar una estrategia de la prueba, mediante una sucesión de lemas.
- El resultado es un fichero con definiciones y teoremas
  - *Un libro* en la terminología de ACL2.
  - El libro puede certificarse y usarse en otros libros.

Presentación de Isabelle . . . y otros sistemas de razonamiento

1. Razonamiento automático con OTTER/MACE
2. Razonamiento automático con ACL2
3. Razonamiento asistido por PVS
4. Razonamiento asistido por Isabelle/HOL/Isar
5. Comentarios finales

Historia de PVS

- PVS:
  - Nombre: *P*rototype *V*erification *S*ystem
  - Autores: N. Shankar, S. Owre y J.M. Rushby en el SRI (Stanford Research Institute).
  - Def: "PVS is a verification system: that is, a specification language integrated with support tools and a theorem prover"
  - Historia: HDM (1970), EHDM (1984), PVS (1991), PVS 4.2 (2008).
- Propósitos:
  - The primary purpose of PVS is to provide formal support for conceptualization and debugging in the early stages of the lifecycle of the design of a hardware or software system
  - The primary emphasis in the PVS proof checker is on supporting the construction of readable proofs

El problema de las monedas

- Enunciado: Demostrar que con monedas de 3 y 5 se puede obtener cualquier cantidad que sea mayor o igual a 8.
- Especificación

```
monedas : THEORY
BEGIN
 n, a, b: VAR nat
 monedas: LEMA
   (FORALL n: (EXISTS a, b: n+8 = 3*a + 5*b))
END monedas
```

El problema de las monedas

- Demostración manual: Por inducción en n
  - Base $n = 0$: $0 + 8 = 3 \times 1 + 5 \times 1$. Basta elegir $a = b = 1$
  - Paso $n + 1$: Supongamos que existen a y b tales que $n + 8 = 3 \times a + 5 \times b$. Vamos a distinguir dos casos según que $b = 0$.

    Caso 1 Sea $b = 0$, entonces $n + 8 = 3 \times a$, $a > 3$ y
    $$(n + 1) + 8 = 3 \times (a - 3) + 5 \times 2$$
    Caso 2 Sea $b \neq 0$, entonces
    $$(n + 1) + 8 = 3 \times (a + 2) + 5 \times (b - 1)$$

El problema de las monedas

- Demostración con PVS

```
monedas :

  |-------
{1}   (FORALL n: (EXISTS a, b: n + 8 = 3 * a + 5 * b))

Rule? (induct "n")
Inducting on n on formula 1, this yields  2 subgoals:
monedas.1 :

  |-------
{1}   EXISTS a, b: 0 + 8 = 3 * a + 5 * b

Rule? (inst 1 1 1)
Instantiating the top quantifier in 1 with the terms: 1, 1,
this simplifies to:
```

El problema de las monedas

```
monedas.1 :

  |-------
{1}   0 + 8 = 3 * 1 + 5 * 1

Rule? (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of monedas.1.

monedas.2 :

  |-------
{1}   FORALL j:
        (EXISTS a, b: j + 8 = 3 * a + 5 * b) IMPLIES
         (EXISTS a, b: j + 1 + 8 = 3 * a + 5 * b)

Rule? (skosimp*)
Repeatedly Skolemizing and flattening, this simplifies to:
```

El problema de las monedas

```
monedas.2 :
{-1}  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (case "b!1=0")
Case splitting on b!1 = 0, this yields  2 subgoals:
monedas.2.1 :
{-1}  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
[1]   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (inst 1 "a!1-3" "2")
Instantiating the top quantifier in 1 with the terms:
a!1-3, 2, this yields  2 subgoals:
```

El problema de las monedas

```
monedas.2.1.1 :
[-1]  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   j!1 + 1 + 8 = 3 * (a!1 - 3) + 5 * 2

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure

This completes the proof of monedas.2.1.1.
```

El problema de las monedas

```
monedas.2.1.2 (TCC):
[-1]  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   a!1 - 3 >= 0

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure
This completes the proof of monedas.2.1.2.
This completes the proof of monedas.2.1.
```

El problema de las monedas

```
monedas.2.2 :
[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
   |-------
{1}   b!1 = 0
[2]   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (inst 2 "a!1+2" "b!1-1")
Instantiating the top quantifier in 2 with the terms:
a!1+2, b!1-1, this yields  2 subgoals:
```

El problema de las monedas

```
monedas.2.2.1 :
[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
[1]   b!1 = 0
{2}   j!1 + 1 + 8 = 3 * (a!1 + 2) + 5 * (b!1 - 1)

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure

This completes the proof of monedas.2.2.1.
```

El problema de las monedas

```
monedas.2.2.2 (TCC):

[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   b!1 - 1 >= 0
[2]   b!1 = 0

Rule? (assert)
Simplifying, rewriting, and recording with decision procedures,

This completes the proof of monedas.2.2.2.

This completes the proof of monedas.2.2.

This completes the proof of monedas.2.
Q.E.D.
```

Demostración PVS

- Llamada: `M-x edit-proof`
- Demostración

```
("" 
 (INDUCT "n")
 (("1" (INST 1 1 1) (ASSERT))
  ("2"
   (SKOSIMP*)
   (CASE "b!1=0")
   (("1" (INST 1 "a!1-3" "2") (("1" (ASSERT)) ("2" (ASSERT))))
    ("2" (INST 2 "a!1+2" "b!1-1") (("1" (ASSERT)) ("2" (ASSERT)))
```

Procedimientos de decisión en PVS

Ejemplos de teoremas demostrados con los procedimientos de decisión de la aritmética lineal con la orden (`reduce`)

```
procedimientos_de_decision: THEORY
 BEGIN
  x,y,z: VAR real
  ej1: THEOREM x < 2*y AND y < 3*z IMPLIES 3*x < 18*z

  i,j,k: VAR int
  ej2: THEOREM i > 0 AND 2*i < 6 IMPLIES i = 1 OR i = 2

  f: [real -> real]
  g: [real, real -> real]
  ej3: THEOREM x = f(y) IMPLIES g(f(y + 2 - 2), x + 2) = g(x,

 END procedimientos_de_decision
```

Ejemplo de teorema de la aritmética no lineal

- Ejemplo del teorema

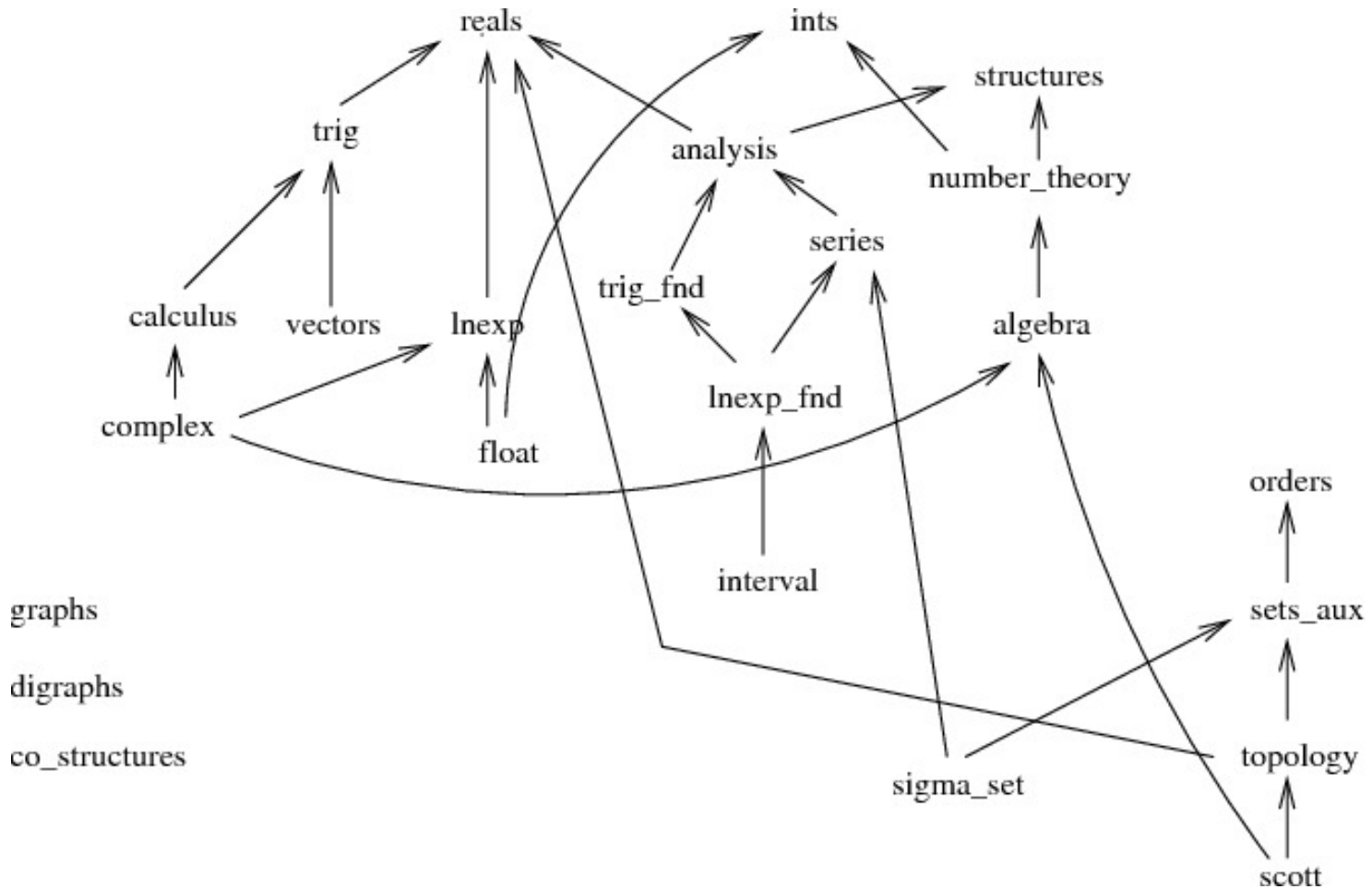
```
aritmetica_no_lineal: THEORY
 BEGIN
  x, y: VAR real
  aritm_no_lineal: THEOREM x<0 AND y<0 IMPLIES x*y>0
 END aritmetica_no_lineal
```

- No se demuestra sólo con los procedimientos de decisión.
- Se demuestra usando la teoría de los números reales de PVS:

```
(grind :theories "real_props")
```

Teorías formalizadas en PVS

The PVS prelude y NASA Langley PVS Libraries (5–Enero–2010)



Presentación de Isabelle . . . y otros sistemas de razonamiento

1. Razonamiento automático con OTTER/MACE
2. Razonamiento automático con ACL2
3. Razonamiento asistido por PVS
4. Razonamiento asistido por Isabelle/HOL/Isar
5. Comentarios finales

El sistema Isabelle/HOL/Isar

- Isabelle es un marco lógico basado en lógica de orden superior.
- Isabelle/HOL es la especialización de Isabelle para HOL (Higher-Order Logic).
- Isar (Intelligible semi–automated reasoning) es un lenguaje de pruebas para Isabelle.
- Isabelle/HOL se ha desarrollado en la Universidad de Cambridge (Larry Paulson) y la Universidad de Munich (Tobias Nipkow).
- Isar se ha desarrollado en Universidad de Munich (Markus Wenzel).

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Demostración a nivel de primero de Matemáticas

Demostración natural

Sea $A = \{x.x \notin f(x)\}$. Vamos a demostrar que $A \notin \mathit{rango}(f)$. Lo haremos por reducción al absurdo. Supongamos que $A \in \mathit{rango}(f)$, entonces existe un b tal que $A = f(b)$. Vamos a demostrar una contradición distinguiendo dos casos:

- Caso 1: Supongamos que $b \in A$. Entonces, $b \notin f(b)$ (por la definición de A) y $b \notin A$ (por la definición de b). Contradicción con $b \in A$.
- Caso 2: Supongamos que $b \notin A$. Entonces, $b \in f(b)$ (por la definición de A) y $b \in A$ (por la definición de b). Contradicción con $b \notin A$.

Demostración a nivel de primero de Matemáticas

Demostración verificada en Isabelle/Isar

```
proof
  let ?A = "{x. x ∉ f x}"
  show "?A ∉ range f"
  proof
    assume "?A ∈ range f"
    then obtain b where "?A = f b" ..
    show False
    proof cases
      assume "b ∈ ?A"
      hence "b ∉ f b" by simp
      hence "b ∉ ?A" using `?A = f b` by simp
      thus False using `b ∈ ?A` by contradiction
    next
      assume "b ∉ ?A"
      hence "b ∈ f b" by simp
      hence "b ∈ ?A" using `?A = f b` by simp
      thus False using `b ∉ ?A` by contradiction
    qed
  qed
qed
```

Demostración a nivel de tercero de Matemáticas

Demostración natural

Consideremos el conjunto $A = \{x.x \notin f(x)\}$. Vamos a demostrar que $A \notin \mathit{rango}(f)$. Lo haremos por reducción al absurdo. Supongamos que $A \in \mathit{rango}(f)$, entonces existe un b tal que $A = f(b)$. Se obtiene una contradición distinguiendo dos casos según que $b \in A$.

Demostración a nivel de tercero de Matemáticas

Demostración verificada en Isabelle/Isar

```
proof
  let ?A = "{x. x ∉ f x}"
  show "?A ∉ range f"
  proof
    assume "?A ∈ range f"
    then obtain a where "?A = f a" ..
    show False
    proof cases
      assume "a ∈ ?A"
      show False using `?A = f a` by blast
    next
     assume "a ∉ ?A"
     show False using `?A = f a` by blast
    qed
```

Demostración a nivel de quinto de Matemáticas

- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Demostración natural:
  Trivial.
- Teorema y demostración verificada en Isabelle/Isar

```
theorem Cantor_3:
  fixes f :: "'α ⇒ 'α set"
  shows "∃A. A ∉ range f"
by best
```

Demostración a nivel de quinto de Matemáticas

- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Demostración natural:
  Trivial.
- Teorema y demostración verificada en Isabelle/Isar

```
theorem Cantor_3:
  fixes f :: "'α ⇒ 'α set"
  shows "∃A. A ∉ range f"
by best
```

Ejemplo de razonamiento inductivo

- Teorema: Para todo número natural n se tiene que

$$1 + 3 + \cdots + (2n - 1) = n \times n$$

- Definición:
  primrec sumaImpares :: "nat ⇒ nat" **where**
  "sumaImpares 0 = 0"
  | "sumaImpares (Suc n) = (2 * (Suc n) - 1) + sumaImpares n"
- Demostración por inducción y simplificación:
  lemma "sumaImpares n = n * n"
  by (induct n) simp_all

Ejemplo de razonamiento inductivo

- Demostración con patrones:

```
lemma "sumaImpares n = n * n" (is "?P n")
proof (induct n)
  show "?P 0" by simp
next
  fix n assume "?P n"
  thus "?P(Suc n)" by simp
qed
```

Ejemplo de razonamiento inductivo

- Demostración con patrones y razonamiento ecuacional:

```
lemma "sumaImpares n = n * n" (is "?P n")
proof (induct n)
  show "?P 0" by simp
next
  fix n assume HI: "?P n"
  have "sumaImpares (Suc n) = (2 * (Suc n) - 1) + sumaImpares n"
   by simp
  also have "... = (2 * (Suc n) - 1) + n * n" using HI by simp
  also have "... = n * n + 2 * n + 1" by simp
  finally show "?P(Suc n)" by simp
qed
```

Teorías formalizadas en Isabelle

- Teorías de Isabelle/HOL:
  - Índice de teorías de Isabelle/HOL
  - Grafo de dependencias de teorías de Isabelle/HOL
  - Documentación de las teorías de Isabelle/HOL
- The Archive of Formal Proofs
  - Perfect Number Theorem
  - Ordinals and Cardinals
  - An Example of a Cofinitary Group in Isabelle/HOL
  - Sums of Two and Four Squares
  - Fermat's Last Theorem for Exponents 3 and 4 and the Parametrisation of Pythagorean Triples
- Isabelle Top 100
- IsarMathLib (A library of formalized mathematics for Isabelle/ZF theorem proving environment).

Presentación de Isabelle . . . y otros sistemas de razonamiento

1. Razonamiento automático con OTTER/MACE
2. Razonamiento automático con ACL2
3. Razonamiento asistido por PVS
4. Razonamiento asistido por Isabelle/HOL/Isar
5. Comentarios finales

Logros actuales en la automatización del razonamiento

- Demostradores automáticos potentes:
  - SAT, procedimientos de decisión, SMT, primer orden
- Extensiones lógicas para modelización:
  - clases de tipos, locales, lógica nominal, reflexión, HOL–Omega.
- Nuevas metodologías para las demostraciones interactivas:
  - demostraciones declarativas, QuickCheck, refutación con SAT.
- Demostración de teoremas importantes:
  - cuatro colores, curva de Jordan, teorema fundamental del cálculo.
- Verificación de aplicaciones industriales:
  - protocolos de seguridad, control de tráfico aéreo.
- Uso de los demostradores como plataformas de programación
  - Programación en los demostradores.
  - Enlaces a herramientas externas.

El futuro del razonamiento automático

- Programación **en** lógica
  - Uso de los demostradores como nuevos IDE.
  - Generación de código certificado.
  - Conexiones con sistemas externos (SAT, SMT, FOL, ...).
- Ampliación de las lógica de los demostradores:
  - HOL2P, HOL–Omega
  - teoría de conjunto
- Ampliación de las aplicacione de los demostradores:
  - Verificación de propiedades profundas de sistemas reales.
  - Uso de demostradores en la enseñanza como "tutores inteligentes".
- Una matemática, muchos sistemas:
  - ampliación de la matemática formalmente verificada (FM, AFP)
  - procesamiento del conocimiento formalizado (MKM)
  - conexiones de sistemas vía una biblioteca universal de teorías.