

Tema 4: Aritmética e inducción en PVS

José A. Alonso Jiménez

Jose-Antonio.Alonso@cs.us.es

<http://www.cs.us.es/~jalonso>

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Definición de funciones aritméticas

- Tipos numéricos: real, rat, int y nat con las operaciones +, -, * y / y las relaciones =, <, <=, > y >=
- Definición de funciones aritméticas

```
aritmetica: THEORY
BEGIN
  x, y: VAR int

  f(x,y): int = (x+y)*(x-y)

  ej1: THEOREM f(5,3) = 16

  ej2: THEOREM f(x,y) = x*x - y*y
END aritmetica
```

Demostración aritmética básica

- Demostración del ej1

ej1 :

|-----
{1} f(5, 3) = 16

Rule? (expand "f")

Expanding the definition of f,
this simplifies to:

ej1 :

|-----
{1} TRUE

which is trivially true.

Q.E.D.

Evaluación aritmética básica

- Evaluación aritmética básica con M-x pvs-ground-evaluator

```
<GndEval> "f(5,3)"
```

```
==>
```

```
16
```

```
<GndEval> q
```

```
Do you really want to quit? (Y or N): y
```

```
NIL
```

Demostración aritmética

- Demostración del ej2

ej2 :

```
|-----  
{1}  FORALL (x, y: int): f(x, y) = x * x - y * y
```

Rule? (expand "f")

Expanding the definition of f,
this simplifies to:

ej2 :

```
|-----  
{1}  TRUE
```

which is trivially true.

Q.E.D.

Procedimientos aritmético de decisión

- Ejemplos de teoremas aritméticos demostrables mediante los procedimientos aritméticos de decisión (usando sólo 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, f(y) + 2)

END procedimientos_de_decision
```

Aritmética no lineal

- Aritmética no lineal

```
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
```

- Prueba usando una teoría del preludio

aritm_no_lineal :

|-----
{1} FORALL (x, y: real): x < 0 AND y < 0 IMPLIES x * y > 0

Rule? (grind :theories "real_props")

pos_times_gt rewrites x * y > 0

 to (0 > x AND 0 > y) OR (x > 0 AND y > 0)

Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.

Tipos en definiciones y demostraciones

- Subtipos para definir funciones totales
(ejemplo de la teoría reals del prelude.pvs)

```
nonzero_real: NONEMPTY_TYPE = {r: real | r /= 0} CONTAINING 1
nzreal:      NONEMPTY_TYPE = nonzero_real

/: [real, nzreal -> real]
```

- Condiciones de tipo generadas en las demostraciones

```
tipos: THEORY
BEGIN
  x, y: VAR real
  ej1: LEMMA x /= y IMPLIES (x - y)/(x - y) = 1
END tipos
```

```
ej1_TCC1: OBLIGATION FORALL (x, y: real): x /= y IMPLIES (x - y) /= 0;
```


Demostración usando propiedades del preludio

- Demostración de ej12 usando la propiedad `div_simp` de la teoría `real_props` de `prelude.pvs`

ej1 :

```
|-----  
{1}  FORALL (x, y: real): x /= y IMPLIES (x - y) / (x - y) = 1
```

Rule? (lemma "div_simp")

Applying `div_simp` this simplifies to:

ej1 :

```
{-1}  FORALL (n0x: nonzero_real): n0x / n0x = 1
```

```
|-----  
[1]  FORALL (x, y: real): x /= y IMPLIES (x - y) / (x - y) = 1
```

Rule? (skosimp)

Skolemizing and flattening, this simplifies to:

Demostración usando propiedades del preludio

ej1 :

```
[-1]  FORALL (n0x: nonzero_real): n0x / n0x = 1
      |-----
{1}   x!1 = y!1
{2}   (x!1 - y!1) / (x!1 - y!1) = 1
```

Rule? (reduce)

Repeatedly simplifying with decision procedures, rewriting,
propositional reasoning, quantifier instantiation, skolemization,
if-lifting and equality replacement,
Q.E.D.

Definición recursiva y condiciones generadas

- Definición recursiva de suma y ejemplo de teorema

```
suma: THEORY
BEGIN
  n:VAR nat

  suma(n): RECURSIVE nat =
  IF n=0 THEN 0
    ELSE n+suma(n-1)
  ENDIF
  MEASURE n

END suma
```

- Condiciones generadas (y probadas)

```
% Subtype TCC generated (at line 7, column 22) for n - 1
suma_TCC1: OBLIGATION FORALL (n: nat): NOT n = 0 IMPLIES n - 1 >= 0
% Termination TCC generated (at line 7, column 17) for suma(n - 1)
suma_TCC2: OBLIGATION FORALL (n: nat): NOT n = 0 IMPLIES n - 1 < n
```

Evaluación básica

- Evaluación básica con M-x pvs-ground-evaluator

```
% <GndEval> "suma(5)"  
% ==>  
% 15  
% <GndEval> "suma(3)"  
% ==>  
% 6  
% <GndEval> q
```

Demostración por inducción con induct

- Teorema: $\sum_{i=0}^n i = \frac{n(n+1)}{2}$
- Ampliación de suma.pvs

```
fla_suma: LEMMA
  suma(n) = (n*(n+1))/2
```

- Demostración de fla_suma con induct

fla_suma :

```
|-----
{1}  FORALL (n: nat): suma(n) = (n * (n + 1)) / 2
```

Rule? (induct "n")

Inducting on n on formula 1, this yields 2 subgoals:

Demostración por inducción con induct

fla_suma.1 :

|-----
{1} suma(0) = (0 * (0 + 1)) / 2

Rule? (expand "suma")

Expanding the definition of suma, this simplifies to:

fla_suma.1 :

|-----
{1} 0 = 0 / 2

Rule? (assert)

Simplifying, rewriting, and recording with decision procedures,

This completes the proof of fla_suma.1.

Demostración por inducción con induct

fla_suma.2 :

```
|-----  
{1}  FORALL j:  
      suma(j) = (j * (j + 1)) / 2 IMPLIES  
      suma(j + 1) = ((j + 1) * (j + 1 + 1)) / 2
```

Rule? (skosimp)

Skolemizing and flattening, this simplifies to:

fla_suma.2 :

```
{-1} suma(j!1) = (j!1 * (j!1 + 1)) / 2  
|-----  
{1}  suma(j!1 + 1) = ((j!1 + 1) * (j!1 + 1 + 1)) / 2
```

Demostración por inducción con induct

Rule? (expand "suma" +)

Expanding the definition of suma, this simplifies to:

fla_suma.2 :

$$[-1] \quad \text{suma}(j!1) = (j!1 * (j!1 + 1)) / 2$$

$$\{1\} \quad 1 + \text{suma}(j!1) + j!1 = (2 + j!1 + (j!1 * j!1 + 2 * j!1)) / 2$$

Rule? (assert)

Simplifying, rewriting, and recording with decision procedures,

This completes the proof of fla_suma.2.

Q.E.D.

Demostración por inducción con induct-and-simplify

- Demostración por inducción con induct-and-simplify

fla_suma :

|-----
{1} FORALL (n: nat): suma(n) = (n * (n + 1)) / 2

Rule? (induct-and-simplify "n")

suma rewrites suma(0)

to 0

suma rewrites suma(1 + j!1)

to 1 + suma(j!1) + j!1

By induction on n, and by repeatedly rewriting and simplifying,
Q.E.D.

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: LEMMA
    (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 procedures,
This completes the proof of monedas.2.1.1.

```
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 procedures,
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 :

$$[-1] \quad j!1 + 8 = 3 * a!1 + 5 * b!1$$

|-----

$$\{1\} \quad b!1 = 0$$

$$[2] \quad \text{EXISTS } a, b: j!1 + 1 + 8 = 3 * a + 5 * b$$

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 sub

monedas.2.2.1 :

$$[-1] \quad j!1 + 8 = 3 * a!1 + 5 * b!1$$

|-----

$$[1] \quad b!1 = 0$$

$$\{2\} \quad j!1 + 1 + 8 = 3 * (a!1 + 2) + 5 * (b!1 - 1)$$

Rule? (assert)

Simplifying, rewriting, and recording with decision procedures,

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))))))))))
```

Bibliografía

- M. Hofmann *Razonamiento asistido por computadora (2001–02)*
- N. Shankar *Mechanized verification methodologies*