Razonamiento automático (2005–06)

Tema 9: Lógica de orden superior en PVS

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Funciones de segundo orden

- Funciones de segundo orden

```
suma_general: THEORY
 BEGIN
  m, n: VAR nat
  f:    VAR [nat -> nat]

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

  cuadrado(n): nat = n*n
  suma_de_cuadrados: LEMMA sumag(cuadrado)(n) = (n * (n + 1) * (2*n

  suma_de_cubos: LEMMA sumag((lambda (m): m*m*m))(n) = (n^2*(n+1)^2
 END suma_general
```

Se prueban con `induct-and-simplify`

Cuantificación sobre funciones

- Teorema de segundo orden (con cuantificación sobre funciones):

$$\sum_{m=0}^{m=n}(f(m)+g(m)) = \sum_{m=0}^{m=n} f(m) + \sum_{m=0}^{m=n} g(m)$$

```
suma_general: THEORY
 BEGIN
  asociativa: LEMMA
    FORALL (f, g: [nat -> nat], n: nat):
    sumag(lambda(m): f(m)+g(m))(n) = sumag(f)(n) + sumag(g)(n)
```

- Se prueba con `induct-and-simplify`

Expresión del axioma de inducción de los naturales

- Expresión del axioma de inducción de los naturales en la teoría `naturalnumbers` del `prelude.pvs`

```
naturalnumbers: THEORY
 BEGIN
  ...
  p: VAR pred[nat]

  nat_induction: LEMMA
    (p(0) AND (FORALL j: p(j) IMPLIES p(j+1)))
          IMPLIES (FORALL i: p(i))

  % Strong induction on naturals.

  NAT_induction: LEMMA
    (FORALL j: (FORALL k: k < j IMPLIES p(k)) IMPLIES p(j))
         IMPLIES (FORALL i: p(i))
 END naturalnumbers
```

Propiedades de funciones

- Especificación de propiedades de funciones

```
funciones [D, R: TYPE]: THEORY
 BEGIN
  f: VAR [D -> R]
  x, x1, x2: VAR D
  y: VAR R

  inyectiva(f):    bool = (FORALL x1, x2: (f(x1) = f(x2) => (x1 = x
  suprayectiva(f): bool = (FORALL y: (EXISTS x: f(x) = y))
  biyectiva(f):    bool = inyectiva(f) & suprayectiva(f)
 END funciones
```

Propiedades de funciones

- Demostración de propiedades de funciones

```
propiedades_funciones: THEORY
 BEGIN
  IMPORTING funciones[int, int]

  x: VAR int

  opuesta (x): int = -x

  opuesta_es_inyectiva:     LEMMA inyectiva(opuesta)

  opuesta_es_suprayectiva: LEMMA suprayectiva(opuesta)

  opuesta_es_biyectiva:     LEMMA biyectiva(opuesta)

 END propiedades_funciones
```

Propiedades de funciones

- Demostración de `opuesta_es_inyectiva`: por `grind`
- Demostración de `opuesta_es_suprayectiva`

```
opuesta_es_suprayectiva :

  |-------
{1}   suprayectiva(opuesta)

Rule? (grind :if-match nil)
opuesta rewrites opuesta(x)
  to -x
suprayectiva rewrites suprayectiva(opuesta)
  to FORALL (y: int): EXISTS (x: int): -x = y
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
```

Propiedades de funciones

```
opuesta_es_suprayectiva :
{-1}  integer_pred(y!1)
  |-------
{1}   EXISTS (x: int): -x = y!1

Rule? (inst 1 "-y!1")
Instantiating the top quantifier in 1 with the terms: -y!1,
this simplifies to:
opuesta_es_suprayectiva :
[-1]  integer_pred(y!1)
  |-------
{1}   --y!1 = y!1

Rule? (reduce)
Repeatedly simplifying with decision procedures, rewriting,
  propositional reasoning, quantifier instantiation, skolemization,
 if-lifting and equality replacement,
Q.E.D.
```

Propiedades de relaciones

- Especificaciones de propiedades de relaciones

```
relaciones [T: TYPE]: THEORY
 BEGIN
  R: VAR PRED[[T, T]]
  x, y, z: VAR T

  reflexiva(R): bool =
    FORALL x: R(x, x)
  simetrica(R): bool =
    FORALL x, y: R(x, y) IMPLIES R(y, x)
  transitiva(R): bool =
    FORALL x, y, z: R(x, y) & R(y, z) => R(x, z)
  equivalencia(R): bool =
    reflexiva(R) AND simetrica(R) AND transitiva(R)
 END relaciones
```

Propiedades de relaciones

- Demostración de propiedades de relaciones

```
propiedades_relaciones: THEORY
 BEGIN
  IMPORTING relaciones[int]
  x, y: VAR int

  igual_valor_absoluto(x,y): bool =
    abs(x) = abs(y)

  igual_valor_absoluto_es_equivalencia: LEMMA
    equivalencia(igual_valor_absoluto)

 END propiedades_relaciones
```

- El lema `igual_valor_absoluto_es_equivalencia` se prueba con `grind`

Conjuntos como predicados

```
conjuntos [T: TYPE]: THEORY
 BEGIN
  conjunto: TYPE = setof[T]
  x, y: VAR T
  a, b, c: VAR conjunto

  pertenece(x, a): bool =
    a(x)
  es_vacio(a): bool =
    (FORALL x: NOT pertenece(x, a))
  vacio: conjunto =
    {x | false}
  subconjunto(a, b): bool =
    (FORALL x: pertenece(x, a) => pertenece(x, b))
  union(a, b): conjunto =
    {x | pertenece(x, a) OR pertenece(x, b)}
  interseccion(a, b): conjunto =
    {x | pertenece(x, a) AND pertenece(x, b)}
```

Conjuntos como predicados

```
 ej1: LEMMA subconjunto(vacio,a)
 ej2: LEMMA subconjunto(interseccion(a,b), union(a,b))
 ej3: LEMMA interseccion(a,a) = a
END conjuntos
```

- Los lemas `ej1` y `ej2` se demuestran con `grind`

Conjuntos como predicados

- Demostración del lema `ej3` con la táctica `apply-extensionality`

```
ej3 :

  |-------
{1}   FORALL (a: conjunto): interseccion(a, a) = a

Rule? (skosimp)
Skolemizing and flattening, this simplifies to:
ej3 :

  |-------
{1}   interseccion(a!1, a!1) = a!1

Rule? (apply-extensionality :hide? t)
Applying extensionality, this simplifies to:
```

Conjuntos como predicados

```
ej3 :

  |-------
{1}   interseccion(a!1, a!1)(x!1) = a!1(x!1)

Rule? (grind)
pertenece rewrites pertenece(x!1, a!1)
  to a!1(x!1)
interseccion rewrites interseccion(a!1, a!1)(x!1)
  to a!1(x!1) AND a!1(x!1)
pertenece rewrites pertenece(x!1, a!1)
  to a!1(x!1)
interseccion rewrites interseccion(a!1, a!1)(x!1)
  to a!1(x!1) AND a!1(x!1)
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
```

Teorías de orden superior del preludio

- Funciones (`functions`): `bijective?`, `domain`, ...
- Relaciones (`relations`): `reflexive?`, `symmetric?`, `equivalence?`, ...
- Relaciones de orden (`orders`): `preorder?`, `partial_order?`, `total_order?`, `well_founded?`, `well_ordered?`, `upper_bound?`, `least_upper_bound?`, ...
- Conjuntos (`sets`): `subset?`, `union`, ...
- Propiedades de conjuntos (`sets_lemmas`): `subset_partial_order`, `demorgan1`, ...
- Inversa de una función (`function_inverse`): `inverse(f)(y)`, `bij_inv_is_bij`, ...
- Imágenes de funciones (`function_image`): `image(f)(X)`, `inverse_image(f)(Y)`, `image_union`, ...
- Composición de funciones (`function_props`): `o(f2,f1)`, `composition_injective`, `assoc`, ...

Teorías de orden superior del preludio

- Operadores de relaciones (`relation_defs`): `domain(R)`, `image(R,X)`, `total?(R)`, ...
- Propiedades de relaciones (`relation_props`): `o(R1, R2)`, `total_composition`, ...
- Operaciones (`operator_defs`): `commutative?`, `has_inverses?`, ...
- Conjuntos finitos (`finite_sets_def`): `is_finite(S)`, `finite_subset`, ...
- Sucesiones (`sequences`): `nth(seq, n)`, `delete(n, seq)`, `insert(x, n, seq)`, ...

Bibliografía

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