Programación en Lógica

Jorge A. Baier

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile

Santiago, Chile

Paradigmas de Lenguajes de Programación

- Programación Orientada a Objetos (Smalltalk)
- Programación Imperativa (Algol, Pascal, C)
- Combinación de las dos anteriores.
- Programación **declarativa**.
  - Programación Funcional.
  - Programación en Lógica.

Programación en Lógica

- Los programas son fórmulas lógicas.
- Computación = Deducción automática = Demostración Automática de Teoremas

$h(x)$: "x es un hombre"
$m(x)$: "x es mortal"

¿Qué significa $h(Denis) \to m(Denis)$?

La Base: Lógica Matemática

$h(x)$: "x es un hombre"
$m(x)$: "x es mortal"

¿Qué significa $h(Denis) \rightarrow m(Denis)$?

¿Qué significa $\forall x\,(h(x) \rightarrow m(x))$?

La Base: Lógica Matemática

$h(x)$: "x es un hombre"
$m(x)$: "x es mortal"

¿Qué significa $h(Denis) \rightarrow m(Denis)$?

¿Qué significa $\forall x\,(h(x) \rightarrow m(x))$?

¿Y esta: $\neg\exists x\, m(x) \rightarrow \forall x\, \neg h(x)$?

Si estamos dispuestos a aceptar $\forall x\,(h(x) \rightarrow m(x))$ como cierto, ¿qué podemos concluir si $\neg m(Denis)$?

La Base: Lógica Matemática

$h(x)$: "x es un hombre"
$m(x)$: "x es mortal"

¿Qué significa $h(Denis) \to m(Denis)$?

¿Qué significa $\forall x\,(h(x) \to m(x))$?

¿Y esta: $\neg\exists x\, m(x) \to \forall x\, \neg h(x)$?

Si estamos dispuestos a aceptar $\forall x\,(h(x) \to m(x))$ como cierto, ¿qué podemos concluir si $\neg m(Denis)$?

$$\{\forall x\,(h(x) \to m(x)), \neg m(Denis)\} \models \neg h(Denis)$$

La Consecuencia Lógica es Computable!

En 1965, John A. Robinson inventó el principio de **resolución**.

La regla de resolución toma dos *cláusulas* C_1 y C_2 y genera una tercera cláusula que es **consecuencia lógica** de la anterior.

[Ejemplo en pizarra]

El razonamiento lógico proposicional **se puede automatizar**.

Programación en Lógica y Resolución

- No todos los lenguajes de programación en lógica usan resolución.
- Prolog, desarrollado en los '70, es un ejemplo.
- Usaremos el intérprete de acá: `http://www.swi-prolog.org/`

Un Ejemplo

```
padre(juan, amanda).
madre(ximena, amanda).
madre(laura, juan).
padre(andres, juan).
padre(patricio, bonifacio).
padre(juan, patricio).
padre(juan, ana).
madre(ximena, ana).

progenitor(X,Y) :- madre(X,Y).
progenitor(X,Y) :- padre(X,Y).

ancestro(X,Y) :- progenitor(X,Y).
ancestro(X,Y) :- progenitor(X,Z),ancestro(Z,Y).
```

Definiendo los Naturales

Una definición inductiva del conjunto $\mathbb{N}$.

- $0 \in \mathbb{N}$.
- Si $x \in \mathbb{N}$, entonces $succ(x) \in \mathbb{N}$.

Definiendo los Naturales

Una definición inductiva del conjunto $\mathbb{N}$.

- $0 \in \mathbb{N}$.
- Si $x \in \mathbb{N}$, entonces $succ(x) \in \mathbb{N}$.

Ahora podemos definir la suma entre dos números de esta forma

- $0 + x = x$, para todo $x \in \mathbb{N}$.

Definiendo los Naturales

Una definición inductiva del conjunto $\mathbb{N}$.

- $0 \in \mathbb{N}$.
- Si $x \in \mathbb{N}$, entonces $succ(x) \in \mathbb{N}$.

Ahora podemos definir la suma entre dos números de esta forma

- $0 + x = x$, para todo $x \in \mathbb{N}$.
- $succ(x) + y = succ(x + y)$

Definiendo los Naturales

Una definición inductiva del conjunto $\mathbb{N}$.

- $0 \in \mathbb{N}$.
- Si $x \in \mathbb{N}$, entonces $succ(x) \in \mathbb{N}$.

Ahora podemos definir la suma entre dos números de esta forma

- $0 + x = x$, para todo $x \in \mathbb{N}$.
- $succ(x) + y = succ(x + y)$

y la multiplicación de esta forma:

- $0 \cdot x = 0$, para todo $x \in \mathbb{N}$.

Definiendo los Naturales

Una definición inductiva del conjunto $\mathbb{N}$.

- $0 \in \mathbb{N}$.
- Si $x \in \mathbb{N}$, entonces $succ(x) \in \mathbb{N}$.

Ahora podemos definir la suma entre dos números de esta forma

- $0 + x = x$, para todo $x \in \mathbb{N}$.
- $succ(x) + y = succ(x + y)$

y la multiplicación de esta forma:

- $0 \cdot x = 0$, para todo $x \in \mathbb{N}$.
- $succ(x) \cdot y = x + x \cdot y$

La Suma, Usando Predicados

- $natural(x)$: es verdadero ssi x es un natural
- $suma(x, y, z)$: es verdadero ssi $z = x + y$
- $mult(x, y, z)$: es verdadero ssi $z = x \cdot y$

Ahora, en Prolog

[Programamos todo esto en Prolog y vemos como es posible usar estos predicados en ambas direcciones. El “desafío mayor” es construir un enumerador de cuadrados perfectos.]

Un programa que aprende

[Ejecutamos un programa “conversador” que es capaz de aprender una taxonomía]

Listas en Prolog

- Las *listas* son un tipo de dato predefinido

Listas en Prolog

- Las *listas* son un tipo de dato predefinido
- `[1,2,3]` es una lista

Listas en Prolog

- Las *listas* son un tipo de dato predefinido
- `[1,2,3]` es una lista
- Para descomponer una lista se usa unificación.

```
?- [1,2,3]=[X|L].
[1,2,3]=[X|L].
X = 1,
L = [2, 3].

?- [1,2,3]=[X,Y|L].
[1,2,3]=[X,Y|L].
X = 1,
Y = 2,
L = [3].
```

Listas en Prolog

- Las *listas* son un tipo de dato predefinido
- `[1,2,3]` es una lista
- Para descomponer una lista se usa unificación.

```
?- [1,2,3]=[X|L].
[1,2,3]=[X|L].
X = 1,
L = [2, 3].

?- [1,2,3]=[X,Y|L].
[1,2,3]=[X,Y|L].
X = 1,
Y = 2,
L = [3].
```

- **Ejercicio:** construir un predicado que compute la suma de todos los elementos en la lista suponiendo la representación de enteros que definimos anteriormente.

Listas, segunda parte

- El predicado `append(L1,L2,L3)` se satisface si `L3` es la concatenación de `L1` con `L2`.
- Ejemplos:

```
?- append(X,Y,[1,2]).
X = [],
Y = [1, 2] ;
;
X = [1],
Y = [2] ;
;
X = [1, 2],
Y = [] ;
false
```

Programas que se modifican a sí mismos

- Los predicados metalógicos `assert` y `retract` modifican el programa.
- **Ejemplos:**
  - `assert(persona(pedro)),`
    `assert((persona(X) :- hombre(X)))`
  - `retract(persona(X))`
  - `retractall(persona(X))`

Construyendo Términos