ST0244 Lenguajes de Programacion
Programación lógica

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2024-2

Preliminares

Convenciones

- La numeración (capítulos, teoremas, figuras, páginas, etc) en estas diapositivas corresponde a la numeración del texto guía [Louden y Lambert 2011].
- Los ejemplos que incluyen código fuente están en el repositorio del curso.

Introducción a la programación lógica

Método de deducción

Argumento deductivo

$$
\begin{array}{l}
P_1 \\
\vdots \\
P_n \\
\therefore\ C
\end{array}
$$

Prueba formal de validez

$$
\begin{array}{rll}
1 & P & \\
 & \vdots & \\
n & P_n & / \therefore\ C \\
n+1 & S_1 & \\
 & \vdots & \\
n+m & S_m &
\end{array}
$$

1

donde:

(i) cada proposición S_i se sigue de las proposiciones anteriores por una regla de inferencia válida y

(ii) la última proposición S_m es la conclusión C.

Introducción a la programación lógica

Definición

El texto guía define un lenguaje de programación lógica por (pág. 108):

> *«A **logic programming language** is a notational system for writing logical statements together with specified algorithms for implementing inference rules.»*

Introducción a la programación lógica

Definición

El texto guía define un lenguaje de programación lógica por (pág. 108):

> *«A **logic programming language** is a notational system for writing logical statements together with specified algorithms for implementing inference rules.»*

Y define un programa lógico por (pág. 108):

> *«The set of logical statements that are taken to be axioms can be viewed as the **logic program**, and the statement or statements that are to be derived can be viewed as the input that initiates the computation. Such inputs are also provided by the programmer and are called **queries** or **goals**.»*

Introducción a la programación lógica

Programación imperativa y programación lógica

- Programación imperativa, Niklaus Wirth (1976)

$$\text{algorithms} + \text{data structures} = \text{programs}.$$

- Programación lógica, Robert Kowalski (1979)

$$\text{algorithm} = \text{logic} + \text{control}.$$

Introducción a Prolog

Características importantes

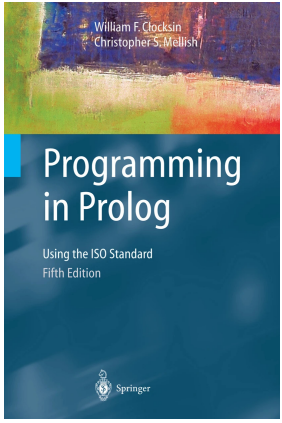
- Prolog es un lenguaje de programación declarativo.
- Prolog es un lenguaje de programación lógica.
- Prolog está basado en lógica de predicados de primer orden y unificación (variables unifican a términos). Prolog no es basado en la arquitectura de von Neumann.
- La unificación es realizada usando una búsqueda primero en profundidad (*depth-first search*) y el rastreo hacia atrás (*backtracking*).
- En Prolog los resultados son obtenidos como relaciones (predicados) en lugar de como funciones. En lugar de definir una función unaria $\text{cuadro}(x)$ para calcular el cuadro de un número, se define una relación binaria $\text{cuadro}(x, y)$ que es verdadera solamente cuando $y = x^2$.

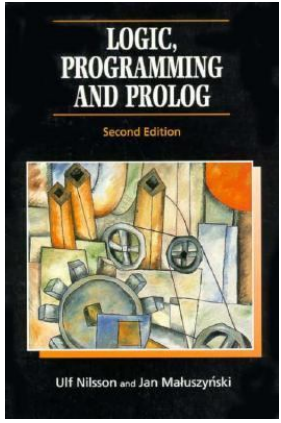
Introducción a Prolog

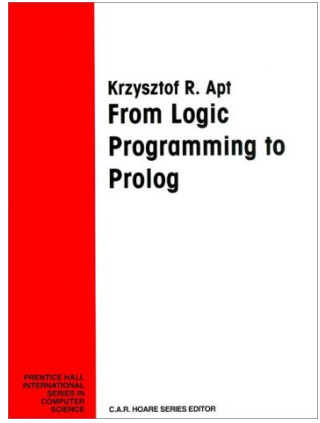
Orígenes

Prolog fue creado en 1972 por Alain Colmerauer y Phillipe Roussel.

Algunos libros







Introducción a Prolog

Usando Prolog

(i) La implementación de Prolog usada en el curso es SWI-Prolog la cual puede ser instalada con la siguiente instrucción:

```
$ sudo apt install swi-prolog
```

Introducción a Prolog

Usando Prolog

(i) La implementación de Prolog usada en el curso es SWI-Prolog la cual puede ser instalada con la siguiente instrucción:

```
$ sudo apt install swi-prolog
```

(ii) Para ejecutar el interpretador de Prolog use la instrucción:

```
$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.6)
...
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Introducción a Prolog

Ejemplo

Mirar el programa en el archivo `lp/family.pl`:

```
$ cd lp
$ swipl family.pl
```

Fundamentos

Terminología

- Variables
- Átomos (*atoms*) (constantes textuales)
- Números (constantes númericas)
- Términos (variables o constantes)
- Predicados (propiedades o relaciones)
- Hechos (*facts*) (predicatos instanciados)

Fundamentos

Definición

Un **programa** en Prolog es un conjunto de hechos y predicados [Clocksin y Mellish 2003].

Fundamentos

Ejemplo

En el archivo `lp/family.pl` tenemos por ejemplo:

- Átomos

  `bruce` y `esther`.

- Predicados

  `female(X)` y `parent(X,Y)`.

- Hechos

  `female(michelle)`, `male(john)` y `parent(gary,kent)`.

Fundamentos

Ejemplo

A partir de la relación $\text{parent}(X, Y)$ y la propiedad $\text{male}(X)$:

$\text{parent}(X, Y)$: X es uno de los padres de Y,
$\text{male}(X)$: X es masculino,

definimos la nueva relación

$\text{father}(X, Y)$: X es el padre de Y

$\text{father}(X, Y)$ si $\text{parent}(X, Y)$ y $\text{male}(X)$,

Fundamentos

Ejemplo

A partir de la relación $\mathrm{parent}(X, Y)$ y la propiedad $\mathrm{male}(X)$:

$\mathrm{parent}(X, Y)$: X es uno de los padres de Y,
$\mathrm{male}(X)$: X es masculino,

definimos la nueva relación

$\mathrm{father}(X, Y)$: X es el padre de Y

$\mathrm{father}(X, Y)$ si $\mathrm{parent}(X, Y)$ y $\mathrm{male}(X)$,

o empleando la sintaxis de Prolog

```
father(X,Y) :- parent(X,Y), male(X).
```

Fundamentos

Descripción

Unificación (*unification*) es el proceso de resolver un conjunto de ecuaciones entre expresiones simbólicas: Determinar si existe una lista de substituciones de variables por términos que satisfagan las ecuaciones.

Fundamentos

Descripción

Unificación (*unification*) es el proceso de resolver un conjunto de ecuaciones entre expresiones simbólicas: Determinar si existe una lista de substituciones de variables por términos que satisfagan las ecuaciones.

Descripción

Prolog realiza la unificación, empleando búsqueda primero en profundidad (*depth-first search*) y el rastreo hacia atrás (*backtracking*), para encontrar un solución.

Ejemplo

En el programa `lp/family.pl` tenemos el siguiente ejemplo de unificación:

```
? - father(gary,X).
X = kent;
X = stephen;
X = anne.
```

Listas

Soporte de Prolog para listas

- La lista vacía es escrita `[]`.
- La lista con cabeza `H` y cola `T` es escrita `[H|T]`.
- Azúcar sintáctico: `[1,2,3]` denota la lista `[1|[2|[3|[]]]]`.
- Azúcar sintáctico:: `[1]` denota la lista `[1|[]]`.
- Puesto que Prolog usa búsqueda primero en profundidad (*depth-first search*) y el rastreo hacia atrás (*backtracking*) podemos usar `[H|T]` para ajuste de patrones (*pattern matching*).

Listas

Ejemplo

Mirar el archivo `lp/lists.pl`:

```
$ cd lp
$ swipl lists.pl
```

Listas

Ejemplo

Regla 1:

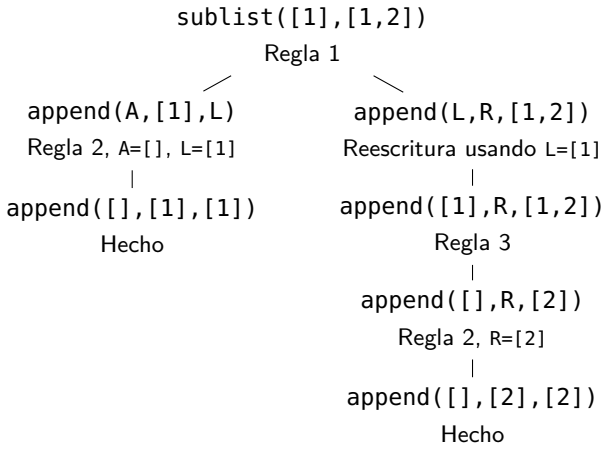
```
sublist(X,Y) :-
  append(_,X,L),
  append(L,_,Y).
```

Regla 2:

```
append([],Y,Y).
```

Regla 3:

```
append([H|T1],L2,[H|T3]) :-
  append(T1,L2,T3).
```



Patrón de acumulación

Ejemplo

Mirar el archivo `lp/reverse.pl`:

```
$ cd lp
$ swipl reverse.pl
```

Predicados *built-in*

- `X = Y` es verdadero si `X` y `Y` unifican.
- `X \= Y` es verdadero si `X` y `Y` no unifican.
- Operadores relacionales numéricos en forma infija (`<`, `>`, `=<`, `>=`, `=:=`, `==`).
- The predicado `not/1` chequea que el argumento (otro predicado) no sea verdadero.
- El predicado `atom/1` chequea que al argumento sea un átomo.
- El predicado `number/1` chequea que el argumento sea un número.

Unificación y aritmética

Operadores de unificación y asignación

En Prolog el operador `=` es para unificación y el operador **`is`** es para asignación.

Ejemplo

```
X = 5 * 3. % unificación
X is 5 * 3. % asignación
```

Unificación y aritmética

Ejemplo

Mirar el archivo `lp/length.pl`:

```
$ cd lp
$ swipl length.pl
```

Referencias


Clocksin, William F. y Mellish, Christopher S. [1981] (2003). Programming in Prolog. 5.ª ed. Springer. DOI: 10.1007/978-3-642-55481-0 (vid. pág. 14).


Louden, Kenneth C. y Lambert, Kenneth A. [1993] (2011). Programming Languages. Principles and Practice. 3.ª ed. Cengage Learning (vid. pág. 2).