ST0244 Lenguajes de Programacion
7. Programación lógica

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2024-1

Preliminares

Convenciones

- La numeración (capítulos, teoremas, figuras, páginas, etc) en estas diapositivas corresponde a la numeración del texto guía [Lee 2017].
- Los ejemplos que incluyen código fuente están en el repositorio del curso.

Introducción

De los lenguajes prescriptibles a los lenguajes descriptivos (pág. 278)

- «*Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.*»

Introducción

De los lenguajes prescriptibles a los lenguajes descriptivos (pág. 278)

- «*Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.*»
- «*C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.*»

Introducción

De los lenguajes prescriptibles a los lenguajes descriptivos (pág. 278)

- «*Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.*»
- «*C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.*»
- «*[Haskell and] Standard ML are high-level languages too, but allows the programmer to think in a mathematical way about a problem. These languages get away from the traditional von Neumann model in some ways.*»

Introducción

De los lenguajes prescriptibles a los lenguajes descriptivos (pág. 278)

- «*Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.*»
- «*C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.*»
- «*[Haskell and] Standard ML are high-level languages too, but allows the programmer to think in a mathematical way about a problem. These languages get away from the traditional von Neumann model in some ways.*»
- «*Prolog takes the descriptive component of languages further and lets programmers write programs based solely on describing relationships.*»

Introducción a la programación lógica

Características de los lenguajes de programación lógica (págs. 277-8)

- «***Descriptive languages:*** *Programs are expressed as known facts and logical relationships about a problem. Programmers assert the existence of the desired result and a logic interpreter then uses the computer to find the desired result by making inferences to prove its existence.*»

Introducción a la programación lógica

Características de los lenguajes de programación lógica (págs. 277-8)

- «***Descriptive languages:*** *Programs are expressed as known facts and logical relationships about a problem. Programmers assert the existence of the desired result and a logic interpreter then uses the computer to find the desired result by making inferences to prove its existence.*»
- «***Non-procedural languages:*** *The programmer states only what is to be accomplished and leaves it to the interpreter to determine how it is to be accomplished.*»

Introducción a la programación lógica

Características de los lenguajes de programación lógica (págs. 277-8)

- «***Descriptive languages:*** *Programs are expressed as known facts and logical relationships about a problem. Programmers assert the existence of the desired result and a logic interpreter then uses the computer to find the desired result by making inferences to prove its existence.*»
- «***Non-procedural languages:*** *The programmer states only what is to be accomplished and leaves it to the interpreter to determine how it is to be accomplished.*»
- «***Relational languages:*** *Desired results are expressed as relations or predicates instead of as functions. Rather than define a function for calculating a square root, the programmer defines a relation, say* $\mathrm{sqrt}(x, y)$*, that is true exactly when* $y^2 = x$*.*»

Introducción a Prolog

Algunas características

- Prolog es un lenguaje de programación lógica.
- Prolog fue creado en 1972 por Alain Colmerauer y Phillipe Roussel.
- Prolog está basado en lógica de predicados de primer orden y unificación (variables unifican a términos). Éste 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*).
- Hay varias versiones de Prolog disponibles incluyendo SWI-Prolog y GNU Prolog.
- Algunas referencias bibliográficas: [Clocksin y Mellish 2003], [Ulf y Maluszyński 2000] y [Apt 1996].

Introducción a Prolog

Ejemplo

Mirar 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

Definimos el predicado binario (relación) father.

X es father de Y si (`:-`) X es un parent de Y y (`,`) X es male:

```
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.

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

Fundamentos

Ejemplo

Usando el `lp/family.pl` tenemos la siguiente solución:

```
? - 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:

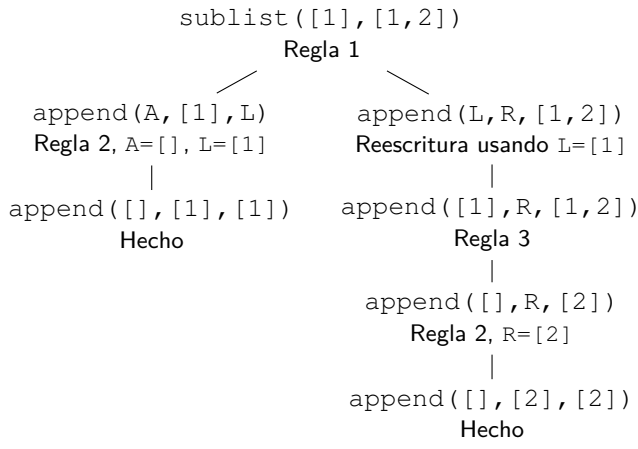
```
sublist(X,Y) :-
  append(_,X,L),
  append(L,_,Y).
```

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

- Apt, Krzystof R. (1996). From Logic Programming to Prolog. Series in Computer Sciences. Prentice-Hall (vid. pág. 10).
- 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. págs. 10, 13).
- Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2.ª ed. Undergraduate Topics in Computer Science. Springer (vid. pág. 2).
- Ulf, Nilsson y Maluszyński, Jan [1990] (2000). Logic, Programming and Prolog. 2.ª ed. John Wiley & Sons (vid. pág. 10).