Prolog

Carlos Hurtado L.

Depto de Ciencias de la Computación, Universidad de Chile

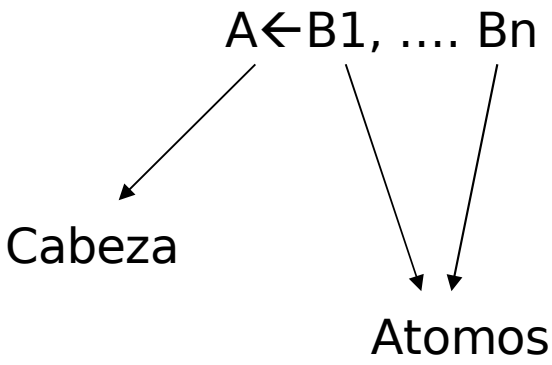
Prolog

- ***Pro****gramation et* ***Log****ique*
- Desarrollado por Colmerauer y Roussel de la Universidad de Aix-Marselle a principios de los 70s.
- Define el paradigma de programación lógica (versus programación procedural).

Programa en Prolog

- Base de Conocimiento (KB):
  - Hechos:
    - Cláusulas con un único literal positivo
    - Ejemplo: “P←” o “P” representa la cláusula (P)
  - Reglas
    - Cláusulas con un literal positivo y más de un literal negativo
    - Ejemplo: “Q←P” representa la cláusula ($\neg P, Q$).
- Objetivo o consulta (α):
    - Conjunción de literales positivos
    - Al negar el objetivo queda un cláusula con literales negativos
    - Ejemplo: “$\leftarrow P,Q$” *o* “*?P,Q*” representa: ($P \wedge Q$) y al negarlo queda como la cláusula ($\neg P, \neg Q$).
- Luego en Prolog ($KB \wedge \neg\alpha$) es un conjunto de cláusulas de Horn.

Reglas en Prolog



Resolución SDL

SDLRes (O: Objetivo)

- Para todo átomo Ai de O
  - Para toda cláusula Cj←B1, …, Bn y UMG σ para Ai y Ciσ de KB
    - Sea O’ el resultado de reemplazar Ai por B1σ,…,Bnσ en O
    - Si O’ es “←” retornar (true)
    - SDLRes(O’)
- retornar(falso)

SLD con Variables en KB pero no en la Consulta

KB:

(1) rich(joan).
(2) mother(linda,joan).
(3) mother(mary,linda).
(4) rich(X) <- mother(X,Y) & rich(Y).

Query:

? rich(linda).

Derivation:

<- rich(linda).
<- mother(linda,Y) & rich(Y).
How: Select rich(linda); resolve with (4) using {X/linda}
<- rich(joan).
How: Select mother(linda,Y) ; resolve with (2) using {Y/joan}
<- .
How: Select rich(joan); resolve with (1) using { }

SLD con Variables en consulta y en KB

KB:

(1) rich(joan).
(2) mother(linda,joan).
(3) mother(mary,linda).
(4) rich(X) <- mother(X,Y) & rich(Y).

Query:

? rich(Z).

Derivation:

yes(Z) <- rich(Z).
yes(Z) <- mother(Z,Y) & rich(Y).
Select rich(Z); resolve with (4) using {X/Z}
yes(Z) <-mother(Z,joan).
Select rich(Y); resolve with (1) using {Y/joan}
yes(linda) <- .
Select mother(Z,joan) ; resolve with (2) using {Z/linda}

SLD con Variables en consulta y en KB

KB:

(1) rich(joan).
(2) mother(linda,joan).
(3) mother(mary,linda).
(4) rich(X) <- mother(X,Y) & rich(Y).

Query:

? rich(Z).

A Different Derivation:

yes(Z) <- rich(Z).
yes(joan) <- .

Select rich(Z); resolve with (1) using {Z/joan}

Different derivations can give different answers;
Exercise: construct derivation that gives the answer "mary".

Resolución SDL con Extracción de Respuestas

SDLRes (O: Objetivo)

// O contiene un átomo answer(T)

- Para todo átomo Ai de O
  - Para toda cláusula Cj←B1, ..., Bn y UMG σ para Ai y Ciσ de KB
    - Sea O' el resultado de reemplazar Ai por B1σ,...,Bnσ en O
    - Si O' es "← answer(T)" agregar T a la respuesta.
    - SDLRes(O')
- retornar(falso)

Prolog

- Selecciona átomos a resolver del cuerpo de una regla en orden izquierda-derecha
- Selecciona reglas a usar en orden arriba-abajo
- Registra elecciones e intenta alternativas ante falla (busqueda primero en profundidad + backtracking)
- Entrega una respuesta única para consultas con variables, pero se puede forzar la busqueda de otras respuestas (usando la opción “;”).

Listas en Prolog

- Prolog provee funciones y predicados para manejar listas:

- Consider the function *cons*, constant *el*:
  - *cons* accepts two args, returns pair containing them
  - e.g, *cons(a,b)*, *cons(a,cons(b,c))*
  - *el* is a constant denoting the empty list
- A proper list is either *el* or a pair whose second element is a proper list
  - *cons(a, cons(b, cons(c, el)))* = [a,b,c]

Notación de Listas en Prolog

- [] is a constant symbol (empty list)
- [|] is a binary function symbol: infix notation (cons)
  - [X|Y] ≡ cons(X,Y)
- [a,b,c] shorthand for [a | [b | [c | []]]] ≡ cons(a,cons(b,cons(c,el))).
  - [c | []] ≡ [c] the list ‘c’—different from ‘c’ by itself.
  - [b | [c]] = [b,c] the list ‘b,c’.
  - [a | [b,c]] = [a,b,c].

Definición de Concatenación (Append)

- El predicado Append(X,Y,Z) se define usando dos reglas:

```
(A1)  append([], Z, Z).

(A2)  append([E1 | R1], Y, [E1 | Rest]) <-
                append(R1, Y, Rest).
```

Ejemplo Concatenación (1)

Query: ? append([a,b], [c,d], [a,b,c,d]).

(A1) append([], Z, Z).

(A2) append([E1 | R1], Y, [E1 | Rest]) <-
append(R1, Y, Rest).

Derivation:

yes <- append([a,b], [c,d], [a,b,c,d]).

yes <- append([b], [c,d], [b,c,d]).

Resolve with (A2) using { E1/a, R1/[b], Y/[c,d], Rest/[b,c,d] }

yes <- append([], [c,d], [c,d]).

Resolve with (A2) using { E1/b, R1/[], Y/[c,d], Rest/[c,d] }

yes <- .

Resolve with (A1) using { Z/[c,d] }

Answer: yes

Ejemplo Concatenacióm (2)

Query: ? append([a,b], [c,d], [g,b,c,d]).

(A1) append([], Z, Z).

(A2) append([E1 | R1], Y, [E1 | Rest]) <-
append(R1, Y, Rest).

Derivation:

yes <- append([a,b], [c,d], [g,b,c,d]).

No append rule can unify with this atom
(convince yourself: look at E1)

Answer: no

Ejemplo Concatenación (3)

Query: ? append(L, M, [a,b,c,d]).

(A1) append([], Z, Z).

(A2) append([E1 | R1], Y, [E1 | Rest]) <-
append(R1, Y, Rest).

Derivation:

yes(L,M) <- append(L, M, [a,b,c,d]).

yes([a|R1], M) <- append(R1, M, [b,c,d]).

Resolve with (A2) using { L/[a|R1], Y/M, E1/a, Rest/[b,c,d] }

yes([a], [b,c,d]) <- .

Resolve with (A1) using { R1/[], M/[b,c,d], Z/[b, c,d] }

Answer: L = [a], M = [b,c,d]

Ejercicio: Concatenación

- Exercise: Give derivations for at least two other answers for the previous query:
- Query: ? append(L, M, [a,b,c,d]).
  - L = [], M = [a,b,c,d]
  - L = [a], M = [b,c,d]
  - L = [a,b], M = [c,d]
  - L = [a,b,c], M = [d]
  - L = [a,b,c,d], M = []
- Note that specifying append "declaratively" (as as set of clauses) means that we can use these clauses in very flexible ways. This would not be possible with a "procedural" representation!