Meta intérpretes (PROLOG)

Ingeniería de Conocimiento

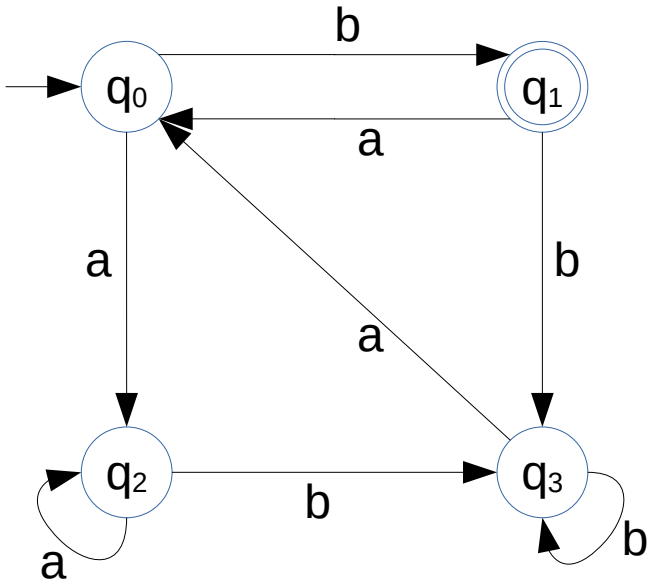
3º Grado de Ingeniería Informática

Intérpretes

- Un metaprograma:
  - Programa que utiliza otro programa como entrada.
  - El hecho de PROLOG ser un intérprete le confiere una ventaja para esta funcionalidad
  - Ejemplos hay muchos. A nivel teórico destaca:
    - Máquina **Universal** de Turing
    - Como entrada recibe la codificación de otra máquina de **Turing**.
    - Esto fue el germen del modelo **Von Neumann**,
    - Lo que llevó a la máquina de **propósito general**

Intérprete de máquinas de estados

- Ejemplo:
  - Máquina de Moore
  - Construir la tabla de transición.
  - Base de conocimiento de un programa Prolog
  - Calcular la función de transición generalizada en Prolog:



$$F(q_i, abbab) = q_f$$

Autómata de Pila

- Reconocer cadenas:

$$L=\{a^n b^n ; n \geq 1\}$$

$$\delta(q_0, a, z)=(q_0, az)$$
$$\delta(q_0, a, a)=(q_0, aa)$$
$$\delta(q_0, b, a)=(q_1, \lambda)$$
$$\delta(q_1, b, a)=(q_1, \lambda)$$
$$\delta(q_1, \lambda, z)=(q_f, z)$$

$$f(q_0, aabb, z)=(q_f, \lambda, z)$$

z - fondo de la pila

λ – palabra vacía

- **Ejercicio:**
  - Construir la base de conocimiento en Prolog
  - Calcular la función de transición f.
  - Plantear un predicado “acepta” de una cadena del lenguaje L, como aquella partiendo del estado inicial, tras procesar dicha cadena, acaba en un estado final.

```
mueve(q0, a, [z], q0, [a|z]).
mueve(q0, a, [a|H], q0, [a|[a|H]]).
mueve(q0, b, [a|H], q1, H).
mueve(q1, b, [a|H], q1, H).
mueve(q1,[],[z],qf,[z]).


transita(q1,[],z,qf,[z]):-!.
transita(Qi, [X|Y], R, Qf, T):- X \=[],
           mueve(Qi, X, R, P, S), transita(P, Y, S, Qf, T).

acepta(X,Resultado) :- transita(q0,X,[z],Q,_), Q=qf,
                       Resultado is 1, !.
```

Ejercicio

- Simular un autómata de pila que acepte los siguientes palíndromos:

$$L = \{ w e w^I / \quad w \in (a/b)^+ \}$$

- w^I es la cadena inversa o reflejada de w.

Máquinas de Turing

- Sería factible su implementación en Prolog:
  - La transición obedecería a:
    - Estado actual
    - Carácter al que apunta el cabezal
  - Provocaría:
    - Estado siguiente
    - Escritura en la posición del cabezal
    - Movimiento de éste: L (izda.) o R (dcha.)
  - Implicaría construir predicados para manejar una **lista** como una **cinta**. Hay alternativas.

Meta-intérprete

- **Intérprete** de un lenguaje escrito en el propio lenguaje.
- Esta idea podría llevar a plantear la creación de **lenguajes de programación**, incluso, su propio entorno integrado.
- En Prolog esto es relativamente fácil, puesto que se pueden formular los problemas bajo un enfoque de **programación lógica**.

Meta intérprete más sencillo

```
solve(A):-A.
```

- No tiene interés, puesto que no aporta nada.
- Con los meta intérpretes, se trata de poder modificar el **cómputo** o la regla de **búsqueda**

Meta intérprete *vanilla*

- Disponible en:

https://artint.info/html/ArtInt_324.html

- En síntesis es:

```
solve(true).
solve((A,B)):-solve(A), solve(B).
solve(A):- clause(A,B), solve(B).
```

vanilla – Lectura Declatariva

```
solve(true).
solve((A,B)):-solve(A), solve(B).
solve(A):- clause(A,B), solve(B).
```

- La meta vacía es cierta.
- La meta conjuntiva (A, B) es cierta, si A es cierta y B es cierta.
- La meta A es cierta, si existe una clausula A:-B y B es cierta

vanilla – Lectura Operacional

```
solve(true).
solve((A,B)):-solve(A), solve(B).
solve(A):-clause(A,B), solve(B).
```

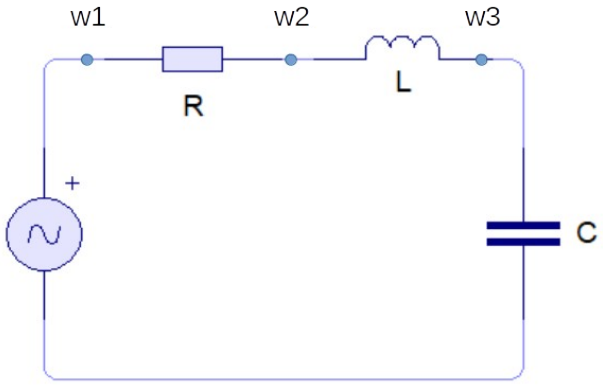
- La meta vacía está resuelta.
- Para resolver la meta (A,B), primero resolver la meta A y después la B. (Regla de cómputo).
- Para resolver la meta A, buscar una claúsula cuya cabeza unifique A y resolver el cuerpo usando la regla de búsqueda de Prolog.

vanilla – Versión mejorada

```
solve(true):- !.
solve((A,B)):- !, solve(A), solve(B).
solve(A):- !, clause(A,B), solve(B).
```

- Estaría limitándose a PROLOG “puro” = programación lógica:
  - Sin modificación de la reevalución: corte, fail, repeat.
  - Sin negación por fallos.
  - Sin asociación de procedimiento: predicados predefinidos

Ejemplo: Propagación de señal




```
valor(w1, 1).
conectado(w2, w1).
conectado(w3, w2).
valor(W,X):-conectado(W,V), valor(V,X)
```

- Ejecutar paso a paso la consulta:
  - `valor(W,X).`
- Añadir el metaintérprete vanilla y ejecutar paso a paso:
  - `solve(valor(W,X)).`