Programación Lógica

Programación lógica

- ¿Qué es?
- Computación vs Deducción
- Juicios, Pruebas y Sustitución de Respuestas
- Backtracking
- Unificación

¿Qué es la programación Lógica?

Es un paradigma de programación basado en deducción y gira en torno al concepto de *predicado*.

1. $flies(X) \leftarrow bird(X), not\ ab(r1, X)$
2. $bird(X) \leftarrow penguin(X)$
3. $ab(r1, X) \leftarrow penguin(X)$
4. $make\_top(X) \leftarrow flies(X)$

f1. $bird(tweety) \leftarrow$

f2. $penguin(sam) \leftarrow$

Idea general

- Dame reglas y hechos, te doy respuestas
  - e.g. Problema de las N-reinas
- Especificación relacional de la forma de la solución + algoritmo de búsqueda general

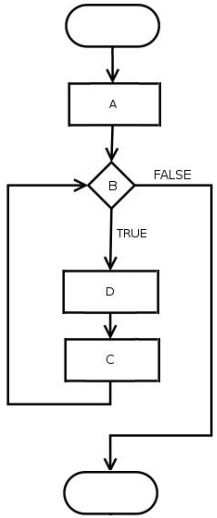
```
#const n = 8.

% n-Queens encoding

{ q(I,1..n) } == 1 :- I = 1..n.
{ q(1..n,J) } == 1 :- J = 1..n.
:- { q(D-J,J) } >= 2, D =   2..2*n.
:- { q(D+J,J) } >= 2, D = 1-n..n-1.
```

Computación vs Deducción

Para entender la programación lógica hay que entender la diferencia entre computación y deducción y la relación entre ambas.

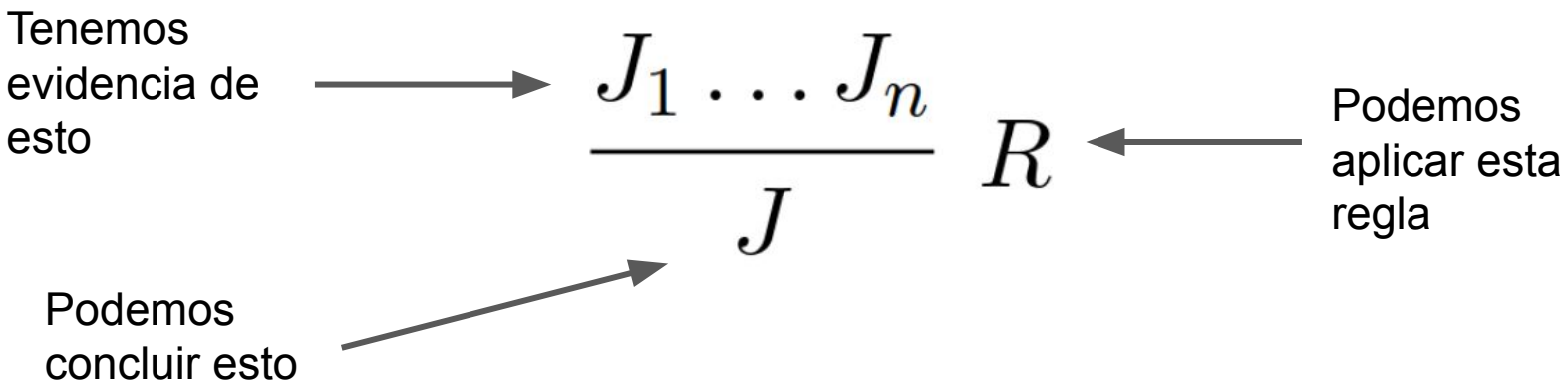


Computación

$$\frac{\dfrac{\varphi \wedge \psi}{\psi}\wedge E \qquad \dfrac{\varphi \wedge \psi}{\varphi}\wedge E}{\psi \wedge \varphi}\wedge I$$

Deducción

Juicios y Pruebas



¿Qué puedo hacer con un programa lógico?

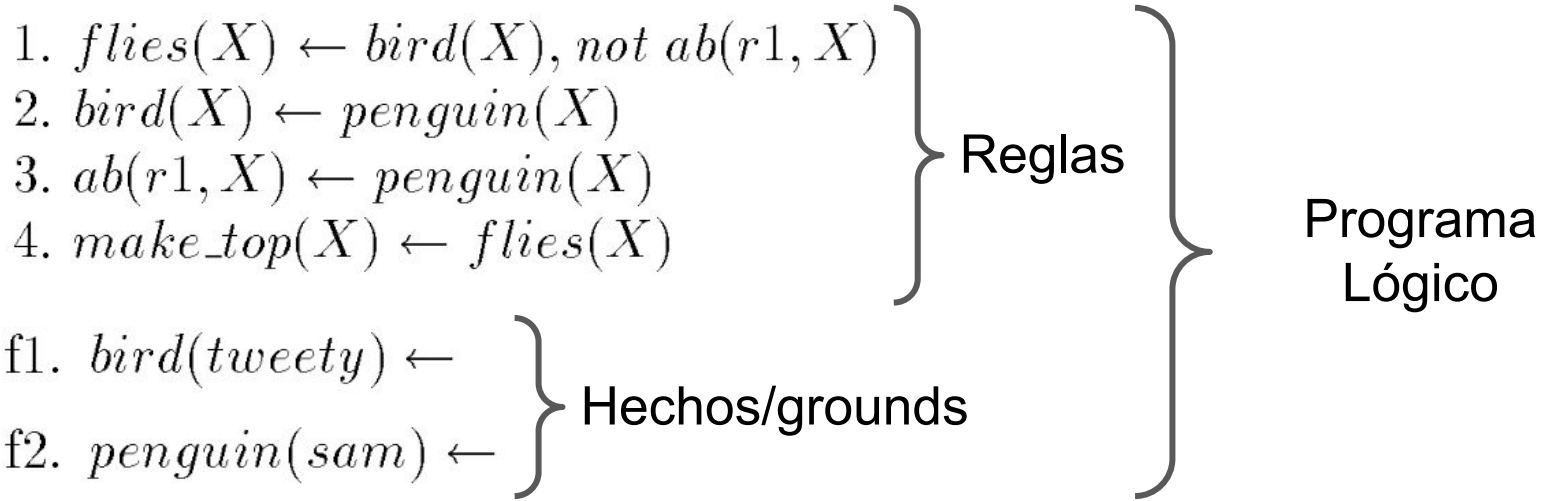
- Preguntas
  - Verificar si una afirmación es consecuencia

    ```
    recomendar(ariel, taladro).
    ```

  - Encontrar un objeto que cumpla con una afirmación, si lo hay

    ```
    recomendar(X, taladro).
    recomendar(ariel, X).
    recomendar(X, Y).
    ```

¿Qué es la programación Lógica?



Cláusulas de Horn

Implementación procedimental de la inferencia

- Regla de inferencia completa por refutación (resolución) + algoritmo de búsqueda completo (e.g. DFS o BFS)

- Comportamientos no declarativos (el orden de evaluación importa, y el orden de escritura de las reglas, el uso de ‘cortes’ o formas de negación no clásicas afecta a las inferencias dadas por el programa - la *negation as failure* y el supuesto de mundo cerrado)

- Toda inferencia es correcta, pero no toda corrida da todas las inferencias (aunque siempre existe una - completitud no determinista)

Búsqueda de prueba como computación

Imaginemos que tenemos las siguientes reglas:

$$\frac{}{\mathsf{even}(\mathsf{z})}\ \mathsf{evz} \qquad\qquad \frac{\mathsf{even}(N)}{\mathsf{even}(\mathsf{s}(\mathsf{s}(N)))}\ \mathsf{evs}$$

podemos deducir:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{even}(\mathsf{z})}\ \mathsf{evz}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))}\ \mathsf{evs}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))))}\ \mathsf{evs}$$

Búsqueda de prueba (ahora en Prolog)

Imaginemos que tenemos las siguientes reglas:

```
even(z).
even(s(s(P))) :- even(P).
```

podemos deducir:

```
? even(s(s(s(s(z)))))
true
```

Prolog to Logic

Cuando escribimos algo de la forma:

$$\textbf{even}(\ s\ (\ s(\ P\ )\ )\ )\ :-\ even(\ P\ ).$$

lo que estamos escribiendo es:

$$even(\ P\ ) \Rightarrow \textbf{even}(\ s\ (\ s\ (\ P\ )\ )$$

La programación lógica puede verse como sucesivas aplicaciones de **modus ponens** más otras técnicas que se explicarán más adelante.

Sustitución de Respuestas

Imaginemos que queremos la función suma:

$$\begin{aligned} (m+1)+n &= (m+n)+1 \\ 0+n &= n \end{aligned}$$

podemos expresarla de la siguiente forma:

$$\frac{\mathsf{plus}(M,N,P)}{\mathsf{plus}(\mathsf{s}(M),N,\mathsf{s}(P))}\ \mathsf{ps} \qquad \frac{}{\mathsf{plus}(\mathsf{z},N,N)}\ \mathsf{pz}.$$

o en Prolog:

```
plus_s(z, N, N).
plus_s(s(M), N, s(P)) :- plus_s(M,N,P)
```

Sustitución de Respuestas

Podemos hacer una query del estilo:

? plus(s (z) , s (z), P).

P es lo que se denomina una *variable lógica* (también encontrado como *variable esquemática*). La idea de estas variables es conocer un valor que hace verdadero al predicado. Para este caso, Prolog responde:

P = s (s (z))

Sustitución de Respuestas

No solo eso, podemos pedir:

? plus(M , s (z), s (s (z))).

a lo que Prolog responde con

M = s (z)

Backtracking

Retomemos la siguiente query:

$$\text{plus}(\ M\ ,\ \text{s}\ (\ \text{z}\ ),\ \text{s}\ (\ \text{s}\ (\ \text{z}\ )\ )\ )$$

y recordemos las reglas:

$$\frac{\text{plus}(M, N, P)}{\text{plus}(\text{s}(M), N, \text{s}(P))}\ \text{ps} \qquad \frac{}{\text{plus}(\text{z}, N, N)}\ \text{pz}.$$

Backtracking

$$\frac{\mathsf{plus}(M, N, P)}{\mathsf{plus}(\mathsf{s}(M), N, \mathsf{s}(P))}\ \mathsf{ps} \qquad \frac{}{\mathsf{plus}(\mathsf{z}, N, N)}\ \mathsf{pz}.$$

$$\frac{\begin{array}{c}\vdots \\ \mathsf{plus}(M_1, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{z}))\end{array}}{\mathsf{plus}(M, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})))}\ \mathsf{ps} \quad \text{with } M = \mathsf{s}(M_1)$$

Backtracking

$$\frac{\mathsf{plus}(M, N, P)}{\mathsf{plus}(\mathsf{s}(M), N, \mathsf{s}(P))}\ \mathsf{ps} \qquad \frac{}{\mathsf{plus}(\mathsf{z}, N, N)}\ \mathsf{pz}.$$

$$\vdots$$

$$\frac{\dfrac{\mathsf{plus}(M_2, \mathsf{s}(\mathsf{z}), \mathsf{z})}{\mathsf{plus}(M_1, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{z}))}\ \mathsf{ps} \quad \text{with } M_1 = \mathsf{s}(M_2)}{\mathsf{plus}(M, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})))}\ \mathsf{ps} \quad \text{with } M = \mathsf{s}(M_1)$$

Backtracking

$$\frac{\mathsf{plus}(M, N, P)}{\mathsf{plus}(\mathsf{s}(M), N, \mathsf{s}(P))}\ \mathsf{ps} \qquad \frac{}{\mathsf{plus}(\mathsf{z}, N, N)}\ \mathsf{pz}.$$

$$\frac{\dfrac{}{\mathsf{plus}(M_1, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{z}))}\ \mathsf{pz} \quad \text{with } M_1 = \mathsf{z}}{\mathsf{plus}(M, \mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})))}\ \mathsf{ps} \quad \text{with } M = \mathsf{s}(M_1)$$

Backtracking

Consideremos ahora la siguiente función:

$$\frac{}{\mathsf{times}(\mathsf{z}, N, \mathsf{z})}\ \mathsf{tz} \qquad \frac{\mathsf{times}(M, N, P) \quad \mathsf{plus}(P, N, Q)}{\mathsf{times}(\mathsf{s}(M), N, Q)}\ \mathsf{ts}$$

con su versión en Prolog:

```
times(z, N, z).
times(s(M), N, Q) :- times(M,N,Q), plus_s(P,N,Q).
```

Backtracking

Si queremos calcular 1x2 = Q, tenemos que deducir:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \mathsf{times}(\mathsf{z}, \mathsf{s}(\mathsf{s}(\mathsf{z})), P) & \mathsf{plus}(P, \mathsf{s}(\mathsf{s}(\mathsf{z})), Q)\end{array}}{\mathsf{times}(\mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})), Q)}\ \mathsf{ts}$$

por lo que necesitamos deducciones para ambas premisas. Podemos pensarlo como:

$$\mathsf{times}(0,2,\ P)\ \wedge\ \mathsf{plus}(P,2,\ Q) \Rightarrow \mathsf{times}(1,\ 2,\ Q)$$

Backtracking

$$\frac{\mathsf{plus}(M, N, P)}{\mathsf{plus}(\mathsf{s}(M), N, \mathsf{s}(P))}\ \mathsf{ps} \qquad \frac{}{\mathsf{plus}(\mathsf{z}, N, N)}\ \mathsf{pz}.$$

$$\frac{}{\mathsf{times}(\mathsf{z}, N, \mathsf{z})}\ \mathsf{tz} \qquad \frac{\mathsf{times}(M, N, P) \quad \mathsf{plus}(P, N, Q)}{\mathsf{times}(\mathsf{s}(M), N, Q)}\ \mathsf{ts}$$

$$\dfrac{\dfrac{}{\mathsf{times}(\mathsf{z}, \mathsf{s}(\mathsf{s}(\mathsf{z})), P)}\ \mathsf{ts}\ (P = \mathsf{z}) \qquad \begin{array}{c}\vdots\\ \mathsf{plus}(P, \mathsf{s}(\mathsf{s}(\mathsf{z})), Q)\end{array}}{\mathsf{times}(\mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})), Q)}\ \mathsf{ts}$$

Backtracking

$$\dfrac{\dfrac{\mathsf{plus}(M,N,P)}{\mathsf{plus}(\mathsf{s}(M),N,\mathsf{s}(P))}\,\mathsf{ps}}{\dfrac{}{\mathsf{times}(\mathsf{z},N,\mathsf{z})}\,\mathsf{tz}} \qquad \dfrac{\dfrac{}{\mathsf{plus}(\mathsf{z},N,N)}\,\mathsf{pz}.}{\dfrac{\mathsf{times}(M,N,P) \quad \mathsf{plus}(P,N,Q)}{\mathsf{times}(\mathsf{s}(M),N,Q)}\,\mathsf{ts}}$$

$$\dfrac{\dfrac{}{\mathsf{times}(\mathsf{z},\mathsf{s}(\mathsf{s}(\mathsf{z})),P)}\,\mathsf{ts}\ (P=\mathsf{z}) \qquad \dfrac{}{\mathsf{plus}(P,\mathsf{s}(\mathsf{s}(\mathsf{z})),Q)}\,\mathsf{pz}\ (Q=\mathsf{s}(\mathsf{s}(\mathsf{z})))}{\mathsf{times}(\mathsf{s}(\mathsf{z}),\mathsf{s}(\mathsf{s}(\mathsf{z})),Q)}\,\mathsf{ts}.$$

Eligiendo el subobjetivo incorrecto

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{plus}(\mathsf{z},N,N)}\,\mathsf{pz}.}{\mathsf{plus}(\mathsf{s}(M),N,\mathsf{s}(P))}\,\mathsf{ps}}{\mathsf{plus}(M,N,P)}}{\mathsf{times}(\mathsf{z},N,\mathsf{z})}\,\mathsf{tz} \quad \dfrac{\mathsf{times}(M,N,P) \quad \mathsf{plus}(P,N,Q)}{\mathsf{times}(\mathsf{s}(M),N,Q)}\,\mathsf{ts}}{\vdots}}{\mathsf{plus}(P_2,\mathsf{s}(\mathsf{s}(\mathsf{z})),Q_2)}\,\mathsf{ps}\ (P_1=\mathsf{s}(P_2),Q_1=\mathsf{s}(Q_2))}{\mathsf{plus}(P_1,\mathsf{s}(\mathsf{s}(\mathsf{z})),Q_1)}\,\mathsf{ps}\ (P=\mathsf{s}(P_1),Q=\mathsf{s}(Q_1))}{\mathsf{plus}(P,\mathsf{s}(\mathsf{s}(\mathsf{z})),Q)}}{\mathsf{times}(\mathsf{z},\mathsf{s}(\mathsf{s}(\mathsf{z})),P) \quad \vdots}}{\mathsf{times}(\mathsf{s}(\mathsf{z}),\mathsf{s}(\mathsf{s}(\mathsf{z})),Q)}\,\mathsf{ts}$$

Unificación y resolución

Dejo mi parte hecha en un bemaer, es muy difícil escribir símbolos matemáticos por acá.

Limitaciones del paradigma

- Rígido y computacionalmente intratable

  (deducción no probabilista – “Sherlock Holmes sin sentido común”)

- No es útil para computación general
- Es muy útil en dominios especializados (planificación, optimización de plantas industriales, resolución de dependencias, y hallazgo de configuraciones consistentes, o de causas de inconsistencias en especificaciones)

Glosario

- **Herbrand Base:** The set of all ground atoms that can be formed from predicate symbols from a clause in Skolemized form and terms from the Herbrand universe of H of F.
- **Negation as failure (NAF):** is a non-monotonic inference rule in logic programming, used to derive *not* p (i.e. that p is assumed not to hold) from failure to derive p. Note that *not* p can be different from the statement ¬p of the logical negation of p, depending on the completeness of the inference algorithm and thus also on the formal logic system.

Preliminares

- Tenemos *objects constants*, *function constants* y *predicate constants*.
- Átomos: $p(t_1, t_2, \ldots, t_n)$ con t_i términos y p un predicado de aridad n.
- Regla: $A_0 \leftarrow A_1, \ldots, A_m$, *not* $A_{m+1}, \ldots,$ *not* A_n donde A_i son átomos y *not* es el conectivo NAF. La izquierda de ←es llamada *conclusión*, la derecha *premisa*.
- Una colección de reglas es un *programa lógico general*.
- Un programa lógico general que no contiene *not* es llamado *programa definitivo.*
- Fórmulas y términos sin variables se denominan *ground.*
- El conjunto de todos los átomos en el lenguage de un programa Π es denotado como HB(Π).

Preliminares 2

- Un modelo estable *S* de un programa definitivo Π es el menor subconjunto de HB(Π) tal que es autocontenido ($A_0 \leftarrow A_1, \ldots, A_n$ en Π y A1, …, An ϵ S entonces A_0 ϵ S).
- Un programa con un solo modelo estable se llama “*categórico”.*
- Un programa sin modelos estables se llama “*incoherente”.*
- Un programa con varios modelos estables se llama “*coherente”.*
-

PROLOG

Prolog es el lenguaje de programación lógica más popular.
Un programa en Prolog se compone de una serie de hechos, relaciones concretas entre objetos de datos y un conjunto de reglas, es decir, un patrón de relaciones entre los objetos de la base de datos .Estos hechos y reglas se introducen en la base de datos a través de una operación de consulta. Un programa se ejecuta cuando el usuario introduce una pregunta un conjunto de términos que deben ser todos ciertos. Los hechos y las reglas de la base de datos se usan para determinar cuáles sustituciones de variables de la pregunta (llamadas unificación) son congruentes con la información de la base de datos. Como intérprete, Prolog solicita entradas al usuario. El usuario digita una pregunta o un nombre de función. La verdad ‘Yes’ o falsedad ‘No’ de esa pregunta se imprime, así como una asignación a las variables de la pregunta que hacen cierta la pregunta, es decir que unifican la pregunta. Si se introduce un ‘;’, entonces se imprime el próximo conjunto de valores que unifican la pregunta, hasta que no son posibles más sustituciones, momento en el que Prolog imprime ‘No’ y aguarda una nueva pregunta. Un cambio de renglón se interpreta como terminación de la búsqueda de soluciones adicionales. La ejecución de Prolog, aunque se basa en la especificación de predicados, opera en forma muy parecida a un lenguaje aplicativo como LISP o ML. El desarrollo de las reglas en Prolog requiere el mismo "pensamiento recursivo" que se necesita para desarrollar programas en esos otros lenguajes aplicativos. Prolog tiene una sintaxis y semántica simples. Puesto que busca relaciones entre una serie de objetos, la variable y la lista son las estructuras de datos básicas que se usan. Una regla se comporta en forma muy parecida a un procedimiento, excepto que el concepto de unificación es más complejo que el proceso relativamente sencillo de sustitución de parámetros por expresiones.