4.3. Resolución

- En programación lógica se utiliza una única regla: la **regla de resolución**.
- Utilizando esta regla y la demostración por reducción al absurdo podemos demostrar cualquier teorema dentro del subconjunto de lógica de predicados que tratamos (las cláusulas de Horn).
- Vamos a ver la regla de resolución aplicada primero a lógica proposicional, y después la veremos ya aplicada a lógica de predicados.

4.3.1. Resolución en lógica proposicional

- Sean dos cláusulas A y B, A = $C1 \vee p$ y $B = C2 \vee \sim p$, donde C1 y C2 son cláusulas y p es una variable proposicional. La regla de resolución se define de la siguiente manera:

  $C1 \vee p, C2 \vee \sim p \vdash C1 \vee C2$

  Se puede demostrar que esta regla es correcta utilizando las reglas de inferencia que ya conocemos.

Ejemplo de aplicación de la regla de resolución:

$p \Rightarrow (q \vee r), q \Rightarrow r, r \Rightarrow s \vdash p \Rightarrow s$

4.3.1. Resolución en lógica proposicional

- Si A y B son cláusulas de Horn, la regla de resolución se puede reescribir de la siguiente manera:

  $C1 \vee p \equiv (q1 \wedge ... \wedge qn) \Rightarrow p$

  $C2 \vee \sim p \equiv (p \wedge r1 \wedge ... \wedge rm) \Rightarrow s$

  siendo q1, ..., qn, r1, ..., rm, p y s variables proposicionales.

$(q1 \wedge ... \wedge qn) \Rightarrow p, (p \wedge r1 \wedge ... \wedge rm) \Rightarrow s \vdash$

$(q1 \wedge ... \wedge qn \wedge r1 \wedge ... \wedge rm) \Rightarrow s$

4.3.1. Resolución en lógica proposicional

- La demostración de una fórmula utilizando resolución sobre cláusulas de Horn, se hace siempre por reducción al absurdo.

Ejemplo:

$p \wedge q \Rightarrow r, p, q \vdash r$

4.3.2. Resolución en lógica de predicados

Sean La y Lb literales tales que La y Lb son unificables mediante el umg σ: $[La]\sigma = [Lb]\sigma$

La regla de resolución se escribe de la siguiente forma:

$(q1 \wedge ... \wedge qn) \Rightarrow La, (Lb \wedge r1 \wedge ... \wedge rm) \Rightarrow s \vdash$

$[(q1 \wedge ... \wedge qn \wedge r1 \wedge ... \wedge rm) \Rightarrow s]\sigma$

donde qi, ri y s son literales.

A la conclusión de la regla se la denomina ***resolvente***.

Ejemplo:

humano(pepe), humano(juan), mortal(X) $\Leftarrow$ humano(X) $\vdash$ mortal(pepe)

4.3.3. Estrategias de resolución

- En una demostración, normalmente existen varias cláusulas a las que podemos aplicar la regla de resolución.
- Existen varias estrategias para elegir estas cláusulas. La que se utiliza en Prolog es la **estrategia de *entrada lineal*** o **SLD-Resolución**.
- Estrategias:
  - **de *entrada*:** se utiliza siempre una de las cláusulas del programa.
  - ***lineal*:** se utiliza siempre el resolvente anterior.

4.3.3. Estrategias de resolución

Estrategia de entrada lineal: Sea S = {L1, .. , Ln} el conjunto de los axiomas (es decir, el programa). Sea F la pregunta o cláusula a demostrar. Para deducir que F es cierto, lo que se hace es demostrar que S U {~F} es una contradicción.

1) La primera aplicación de la regla de resolución se hace sobre ~F y una de las cláusulas de S. Llamaremos al resolvente R1.

2) Cada resolvente Ri+1 se obtiene por resolución a partir del resolvente anterior Ri y una de las cláusulas del programa S.

Si llegamos a la **cláusula vacia (⇐)**, significa que hemos llegado a una **contradicción**, por lo que la pregunta es **cierta**.

Si **quedan preguntas** y no podemos aplicar la regla de resolución, se produce un **fallo**. No podemos demostrar la cláusula.

4.3.3. Estrategias de resolución

Ejemplo:

```
s(X):-
  q(Y),
  r(X,Y).
q(X):-
  p(X).
p(b).
r(a,b).
```

¿s(a)?

4.3.3. Estrategias de resolución

- En la estrategia de entrada lineal, los **resolventes son siempre preguntas.** Además siempre está claro sobre que cláusulas se utiliza la regla de resolución.
- Por esta razón, la representación de la demostración se puede simplificar y escribir en forma de árbol:

s(a)
| {a/X}
q(Y) ∧ r(a,Y)
| {Y/X}
p(Y) ∧ r(a,Y)
| {b/Y}
r(a,b)
|
⇐

4.3.3. Estrategias de resolución

- Si tenemos varias preguntas, ¿en que orden se resuelven?
  - En Prolog, siempre se escoge la pregunta más a la izquierda.
- ¿Qué cláusula del programa se escoge de todas las posibles?
  - En Prolog, la cláusula a unificar con la pregunta se busca siempre de arriba a abajo dentro del programa.

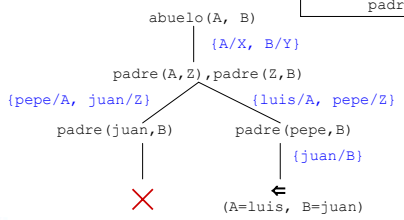
4.3.4. Backtracking

- Cuando se elige una cláusula del programa para resolver la pregunta actual, ¿Qué pasa si esa no es la correcta para realizar la demostración?. **Se deberían probar todas las posibilidades**.
- Para resolver este problema el Prolog posee un mecanismo de **backtracking** o **vuelta atrás**:
  - Si hay varias opciones, se elige la primera posibilidad.
  - Si se produce un fallo, deshacemos lo hecho y elegimos la siguiente posibilidad.
  - Si no quedan posibilidades se produce un fallo.
- Utilizando backtracking, la demostración tiene forma de árbol.

4.3.4. Backtracking

¿abuelo(A,B)?

```
padre(pepe, juan).
padre(luis, pepe).
abuelo(X, Y):-
      padre(X, Z),
      padre(Z, Y).
```



4.3.4. Backtracking

- Los puntos de la ejecución donde se pueden elegir varias posibilidades se denominan **puntos de backtracking**.
- Un **punto de backtracking** guarda el estado del programa (el valor de todas las variables) en una pila antes de realizar la elección, de forma que para volver atrás se restauran los valores de las variables.

4.3.4. Backtracking

- **Ejemplo: ¿m(a)?**

```
m(X):-
        s(X),
        n(b).
m(X):-
        s(X),
        n(X).

s(X):-
        l(X).
s(X):-
        q(Y),
        r(X,Y).
```

```
q(X):-
        p(X).
p(b).
r(a,b).
l(c).
n(a).
```

4.3.5. Negación en programación lógica

- Utilizando cláusulas de Horn junto con resolución no es posible derivar literales negativos, es decir, cláusulas del tipo: **~abuelo(pepe, juan)**.
- Para solucionar esto, en Prolog se supone que únicamente es cierto lo que está escrito en el programa, por tanto, todo lo que no conocemos es falso. Esta suposición se denomina **suposición del mundo cerrado**.
- Todo lo que no se puede demostrar será también falso. Si intentamos demostrar **abuelo(pepe, juan)** y no podemos, se considerará falso.