Programación lógica

Prolog

Prolog se enmarca en el paradigma de los

lenguajes **lógicos y declarativos**.

Sintaxis de Prolog

Los programas en Prolog se componen de **cláusulas de Horn**.

La sintaxis de una cláusula de Horn tiene el siguiente aspecto:

hija(A,B) :- mujer(A), madre(B,A).

Se lee así: "A es hija de B si A es mujer y B es madre de A".

En términos lógicos representa la siguiente implicación:

$(\text{mujer}(A) \land \text{madre}(B,A)) \rightarrow \text{hija}(A,B)$

Por definición de implicación se obtiene la siguiente equivalencia lógica:

$\neg (\text{mujer}(A) \land \text{madre}(B,A)) \lor \text{hija}(A,B)$

$\neg \text{mujer}(A) \lor \neg \text{madre}(B,A) \lor \text{hija}(A,B)$

Def. una fórmula lógica es una **cláusula de Horn** si es una cláusula (disyunción de literales) con, como máximo, un literal positivo (1 o 0)

Sintaxis de Prolog

En PROLOG, el símbolo **:-** separa la conclusión de las condiciones.

Las variables se escriben comenzando por una **letra mayúscula**.

Las constantes con letra minúscula.

Todas las condiciones deben cumplirse para que la conclusión sea válida; por tanto, la coma que separa las distintas condiciones representa a la conjunción .

En cambio la disyunción normalmente no se representa mediante símbolos especiales (aunque puede hacerse con el símbolo *;*), sino añadiendo reglas nuevas al programa. En este caso:

```
hija(A,B) :- mujer(A), padre(B,A).
hija(A,B) :- mujer(A), madre(B,A).
```

Se lee así: "A es hija de B si A es mujer y B es padre de A

- o A es hija de B si A es mujer y B es madre de A".

Razonamiento en Prolog

HECHOS:

```
mujer(alex).
madre(claudia,alex).
```

REGLAS:

```
hija(A,B) :- mujer(A), madre(B,A).
```

RAZONAMIENTO DEDUCTIVO:

```
hija(alex, claudia).
```

Es como aplicar la Regla Modus Ponens.

Hechos y reglas

posee_skill(Persona, S)

se satisface cuando la Persona tiene la habilidad S.

prefiere_rol(Persona,Rol)

se satisface cuando la Persona prefiere asumir el Rol.

es_para(S,Rol)

se satisface cuando la habilidad S es útil para asumir el Rol.

se_puede_trasladar(Persona,L)

se satisface cuando la Persona puede viajar a la ciudad L.

Hechos y reglas

HECHOS:

"Gabriela maneja java y uml" se representa

posee_skill(gabriela, java).
posee_skill(gabriela, uml).

"Gabriela prefiere trabajar como programadora" se representa

prefiere_rol(gabriela, programador).

"Saber java es útil para trabajar como programador".

es_para(java, programador).

"Gabriela se puede trasladar a Córdoba".

se_puede_trasladar(gabriela, cordoba).

Hechos y reglas

REGLAS:

R1: es_apto(Persona, Rol) :-
prefiere_rol(Persona, Rol),
posee_skill(Persona, Skill),
es_para(Skill, Rol).

R2: empleado_disponible(Persona, Rol, Ubicacion):-
se_puede_trasladar(Persona, Ubicacion),
es_apto(Persona, Rol).

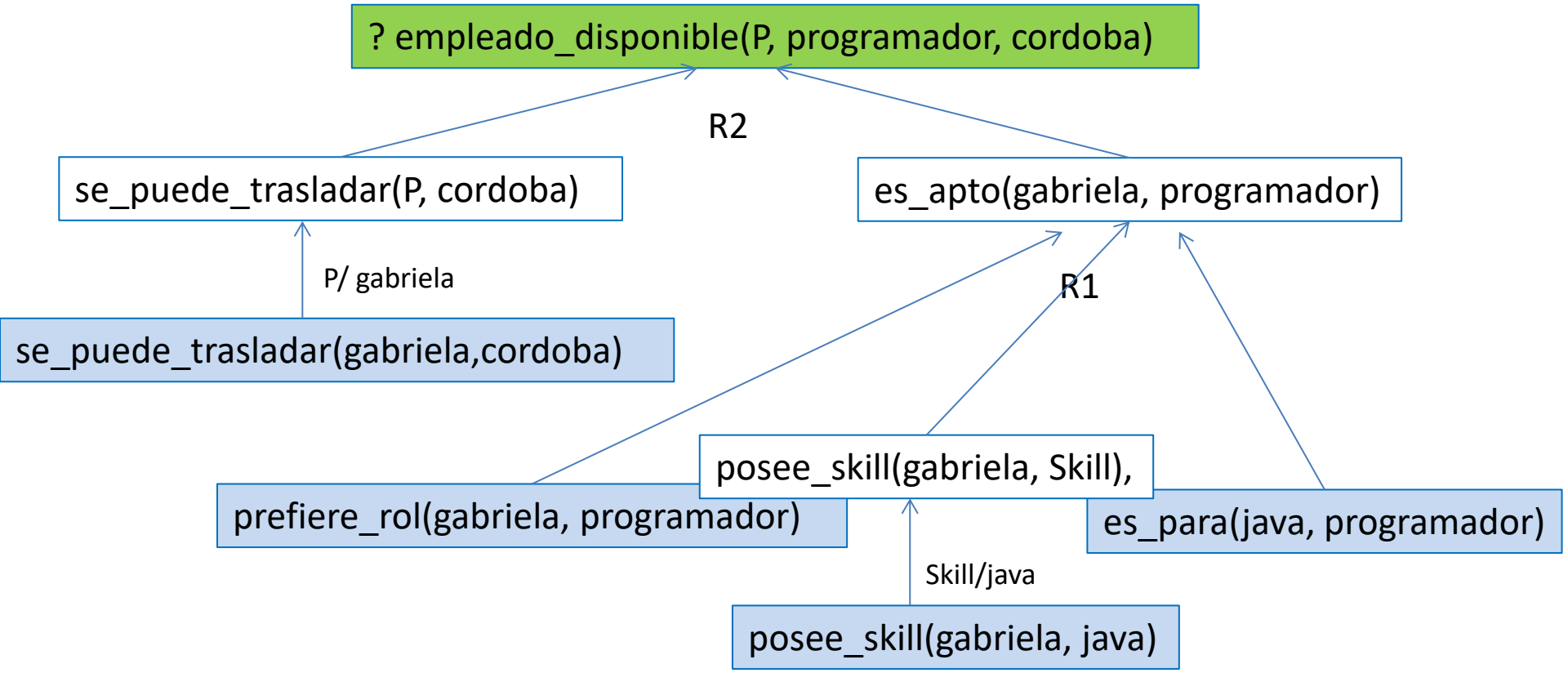
Inferencia hacia atrás y Unificación

La ejecución de un programa Prolog se basa en dos conceptos:

- **la unificación**
- **y el backtracking**

Inferencia hacia atrás. Unificación

El árbol de demostración generado por el encadenamiento hacia atrás.



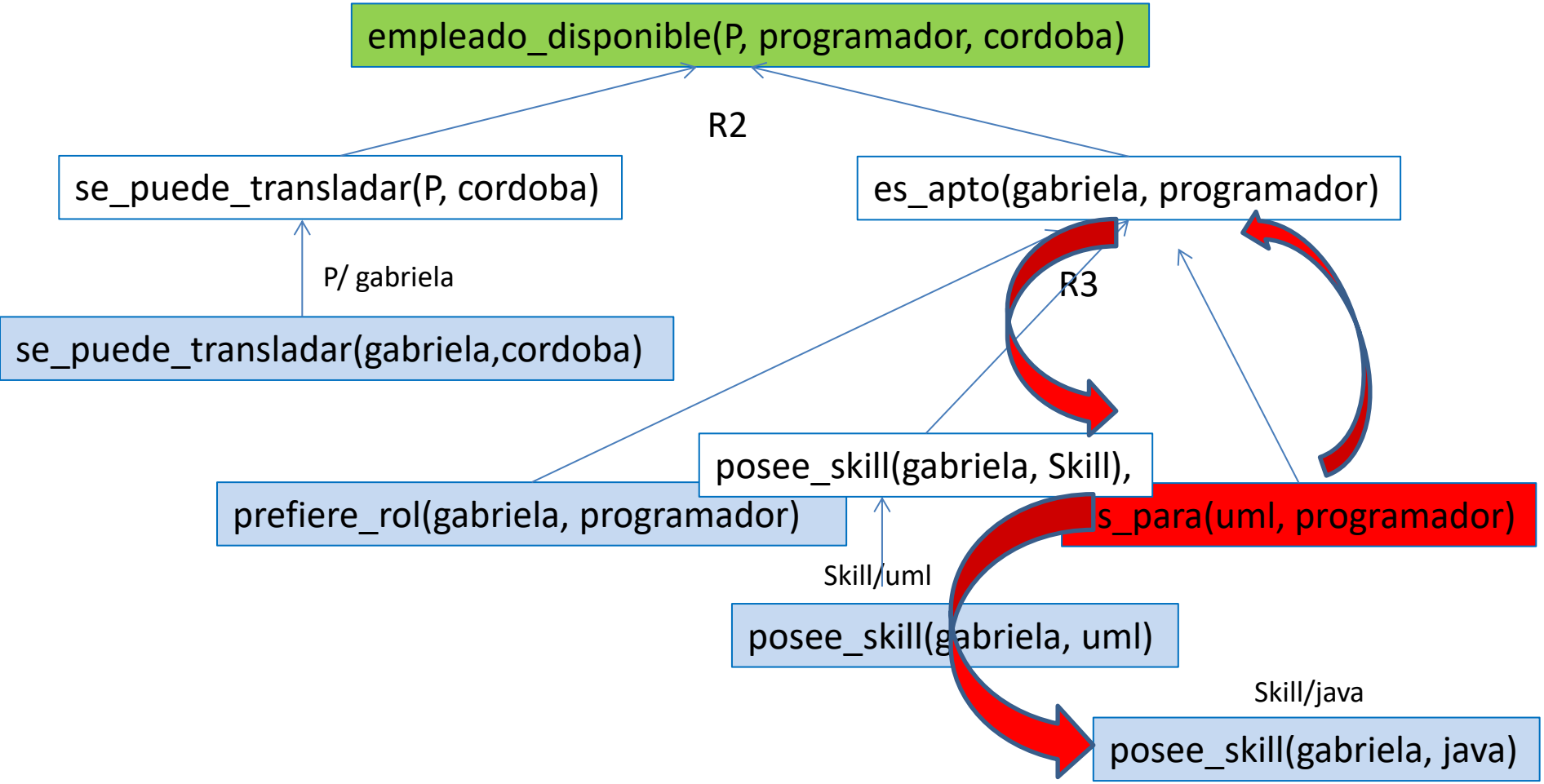
Inferencia hacia atrás. Unificación

Gracias a la ***unificación***, cada *objetivo* determina un subconjunto de cláusulas susceptibles de ser ejecutadas. Cada una de ellas se denomina ***punto de elección***. Prolog selecciona el primer punto de elección y sigue ejecutando el programa hasta determinar si el objetivo es verdadero o falso.

El árbol debe leerse en profundidad de izquierda a derecha.

Para probar empleado_disponible(P, programador, cordoba) hay que probar las dos conjunciones debajo de él. La primera está en la base de conocimiento, es un **hecho**, mientras que otra requiere mas encadenamiento hacia atrás.

Inferencia hacia atrás: backtracking



En caso de ser falso aplica el ***backtracking***, consiste en deshacer todo lo ejecutado situando el programa en el mismo estado en el que estaba justo antes de llegar al punto de elección. Entonces se toma el siguiente punto de elección y se repite el proceso.

Inferencia hacia atrás

función PREGUNTA-EHA-LPO(*BC*, *objetivos*, θ) **devuelve** un conjunto de sustituciones
  entradas: *BC*, una base de conocimiento
    objetivos, una lista de conjuntores que forman la petición (θ ya aplicada)
    θ, la sustitución actual, inicialmente la sustitución vacía { }
  variables locales: *respuestas*, un conjunto de sustituciones, inicialmente vacío

  si *objetivos* está vacío **entonces devolver** $\{\theta\}$
  $q' \leftarrow$ SUST(θ, PRIMERO(*objetivos*))
  para cada sentencia r **en** *BC* **hacer** donde ESTANDARIZAR-VAR(r) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    y $\theta' \leftarrow$ UNIFICA(q, q') tiene éxito
    nuevos_objetivos $\leftarrow [p_1, \ldots, p_n \mid$ RESTO(*objetivos*)$]$
    respuestas $\leftarrow$ PREGUNTA-EHA-LPO(*BC*, *nuevos_objetivos*, COMPON(θ', θ)) $\cup$ *respuestas*
  devolver *respuestas*

Prolog

https://www.swi-prolog.org/

```
% HECHOS: ------------------------------
mujer(alex).
madre(clau,alex).
% REGLAS: % ------------------------------
hija(A,B) :-
   mujer(A),
   madre(B,A).
```

```
% HECHOS:
% -------------------------------
%   “Gabriela maneja java y uml”
posee_skill(gabriela, java).
posee_skill(gabriela, uml).
% “Gabriela prefiere trabajar como programadora”
prefiere_rol(gabriela, programador).
% “Saber java es útil para trabajar como programador”.
es_para(java, programador).
% “Gabriela se puede trasladar a Córdoba”.
se_puede_trasladar(gabriela, cordoba).
% -------------------------------
% REGLAS:
% -------------------------------
%  R1:
es_apto(Persona, Rol) :-
       prefiere_rol(Persona, Rol),
       posee_skill(Persona, Skill),   es_para(Skill, Rol).
% R2:
empleado_disponible(Persona, Rol, Ubicacion):-
      se_puede_trasladar(Persona, Ubicacion),
       es_apto(Persona, Rol).
```