Prolog (1)

Carlos Hurtado L.

Depto de Ciencias de la Computación, Universidad de Chile

Clase Pasada

- Refutación mediante resolución para lógica de Predicados
- Extraccón de respuestas de una base de conocimiento

Clase Pasada: Ejemplo: Regla Generalizada de Resolución

Dadas las cláusulas:

(P(x),Q(g(x))) y (¬P(a),R(a),Q(z))

En este caso L=P(X), M=P(a) y su UMG es {x/a}. Luego obtenemos el resolvente:

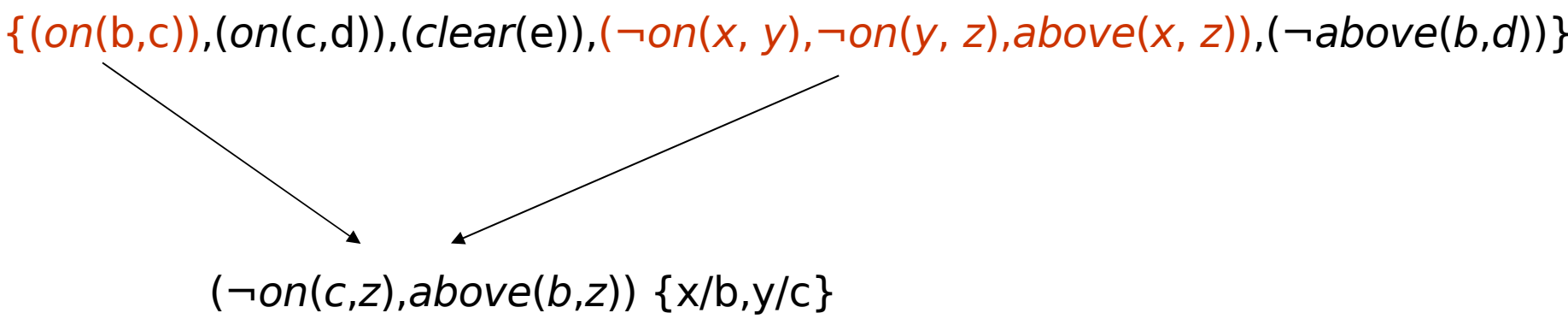
(Q(g(a)),R(a),Q(z))

Clase Pasada: Ejemplo: Mundo de los Bloques

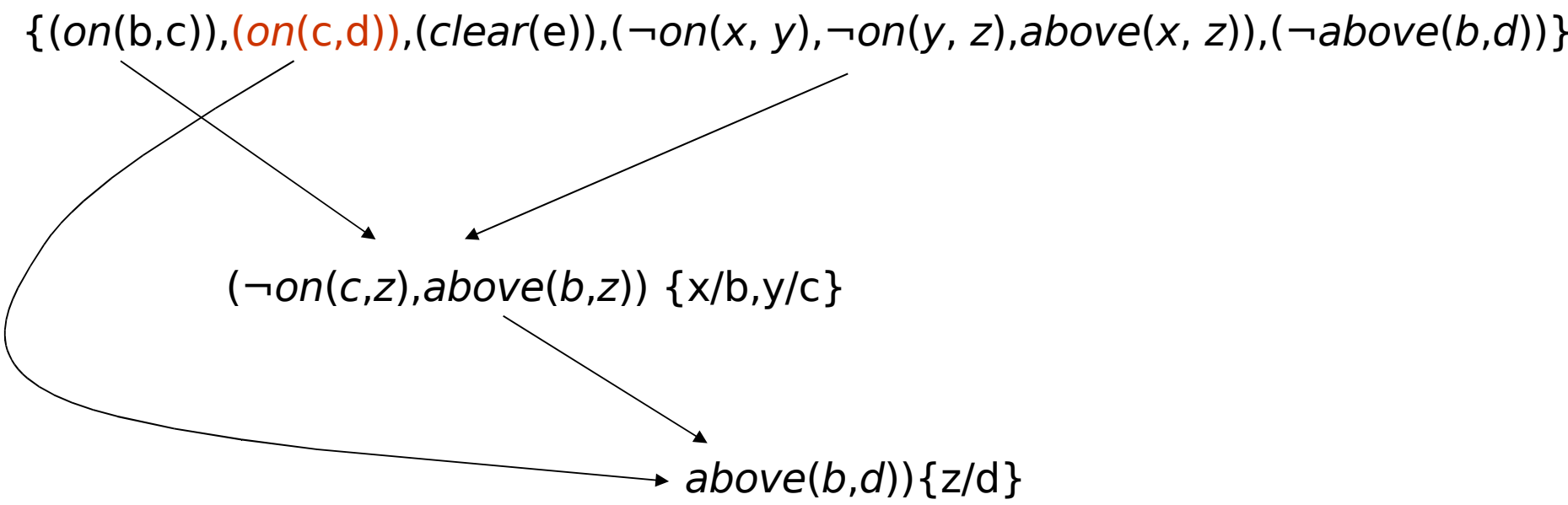
- Sea KB={*on*(b,c),*on*(c,d),*clear*(e), $\forall$ x,y,z.(*on*(*x*, *y*)$\wedge$*on*(*y*, *z*) $\Rightarrow$*above*(*x*, *z*))}.
- ¿KB $\models$ *above*(*b*,*d*)?
- Transformamos KB $\wedge$ $\neg$*above*(*b*,*d*) a conjunto de cláusulas:

{(*on*(b,c)),(*on*(c,d)),(*clear*(e)),($\neg$*on*(*x*, *y*), $\neg$*on*(*y*, *z*),*above*(*x*, *z*)),$\neg$*above*(*b*,*d*)}.

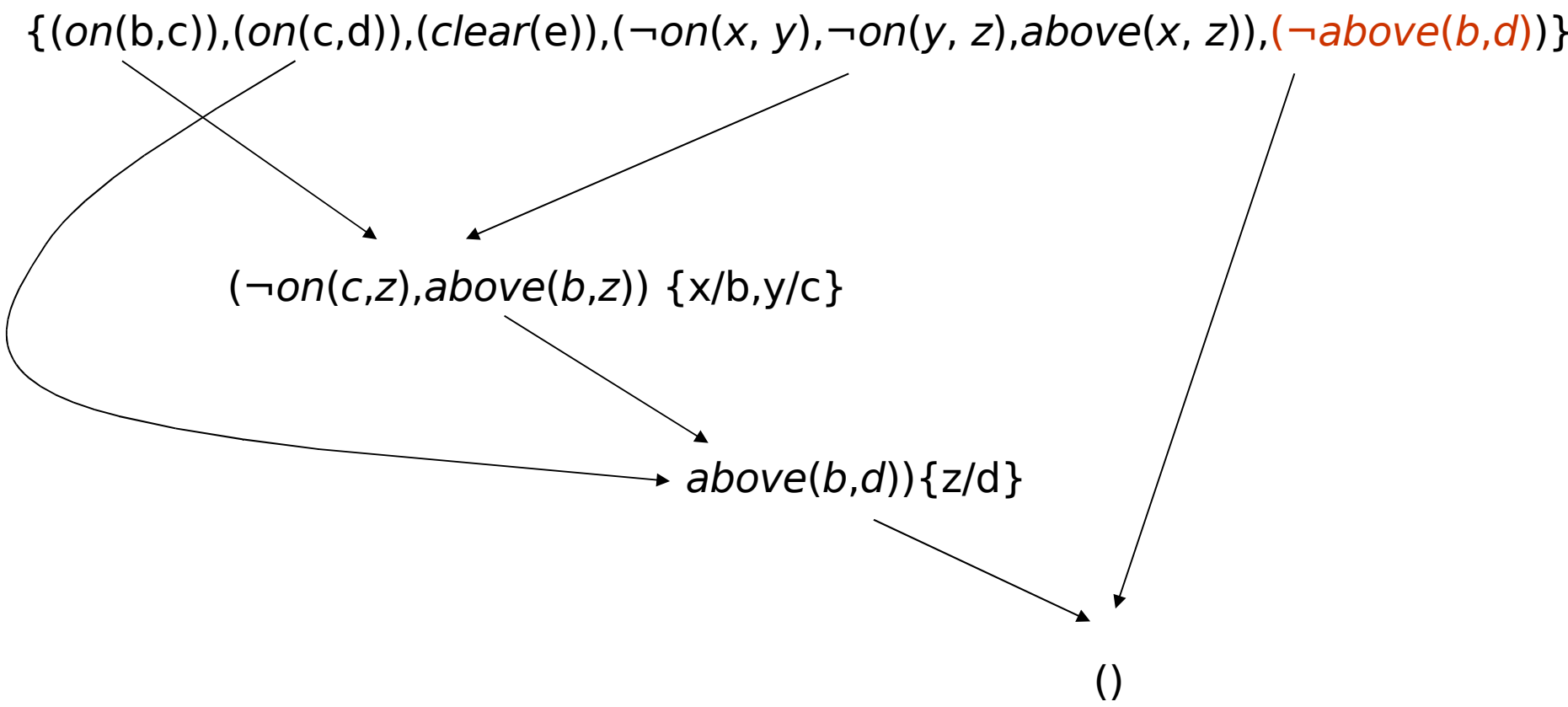
Clase Pasada: Ejemplo: Mundo de los Bloques



Clase Pasada: Ejemplo: Mundo de los Bloques



Clase Pasada: Ejemplo: Mundo de los Bloques



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).
- Resolución sobre cláusulas de Horn

Compiladores e IDEs para Prolog

- GNU Prolog (GPL)
- SWI Prolog (LGPL)
- Kernel Prolog (GPL)
- B-Prolog (comercial)
- Strawberry Prolog (comercial)
- ... etc
- www.prolog.info lista 34 compiladores e IDEs

Cláusulas de Horn

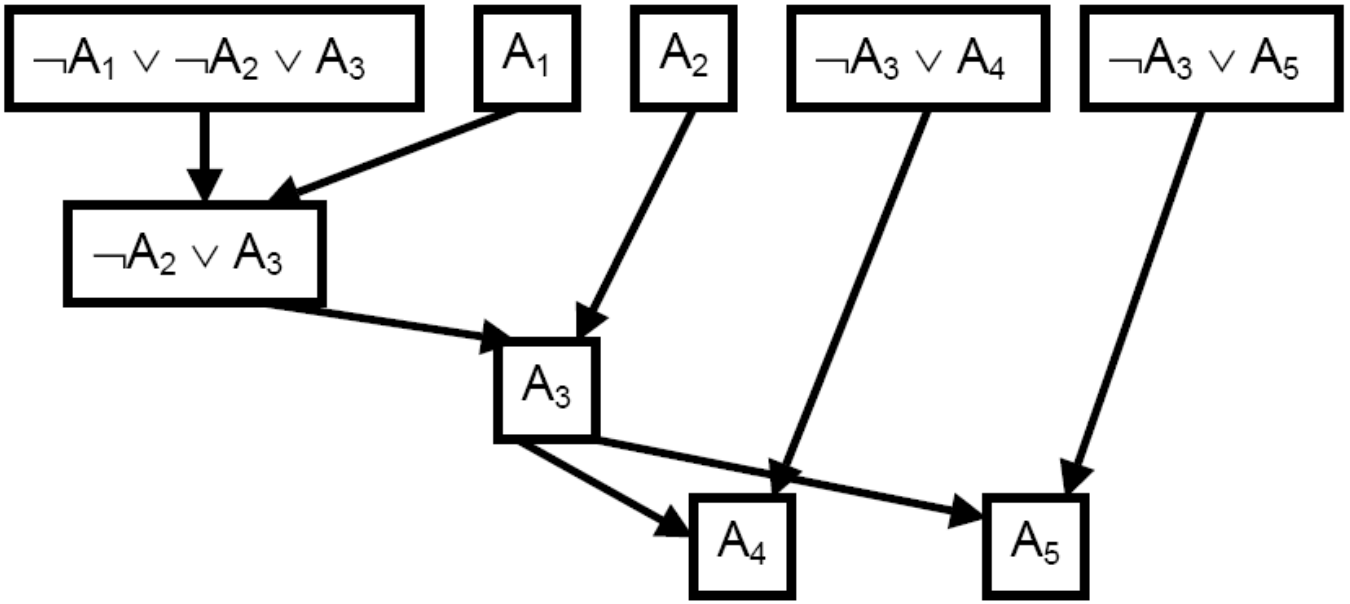
- Cláusulas con no más de un literal positivo
- Ejemplos.
- Lógica Proposicional:

$$(P),\ (\neg P \vee Q\ ),\ (\neg P \vee \neg Q\ )$$

- Lógica de Predicados:

$$(P(x,a)),\ (\neg P(y,a) \vee Q\ (y,b))$$

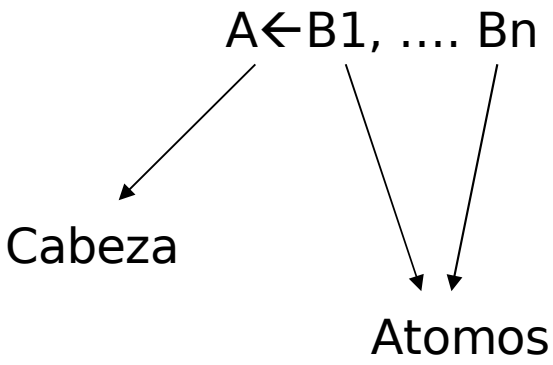
Cláusulas Proposicionales de Horn y Resolución Unitaria



Teorema: si un conjunto de cláusulas de Horn no es satisfacible entonces existe una refutación unitaria positiva.
Corolario: También existe una refutación lineal.

Cláusulas de Horn en Prolog

(A,-B1,...,-Bn)



Casos particulares:

(A) A←

(-B1,...,-Bn) ←B1, ... Bn

Programa en Prolog

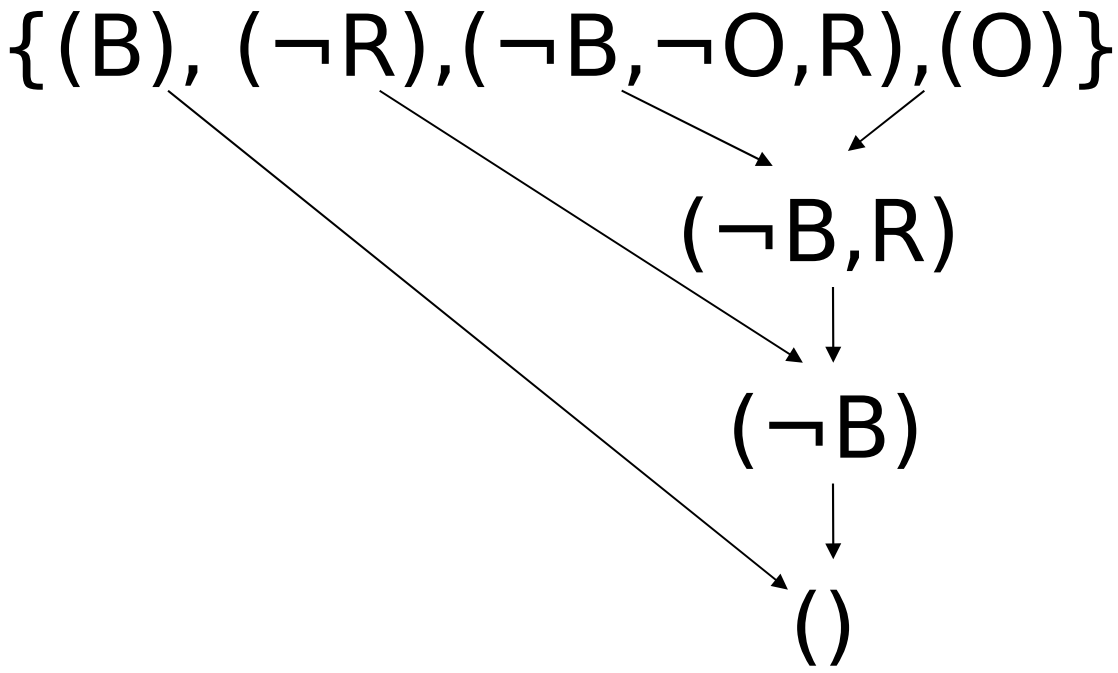
- Base de Conocimiento (KB):
  - Hechos:
    - Cláusulas con un único literal positivo
    - Se denotan “P←” o “P.” representa la cláusula (P)
  - Reglas
    - Cláusulas con un literal positivo y más de un literal negativo
    - Ejemplo: “Q←P” representa la cláusula ($\neg P, Q$).

Programa en Prolog

- Objetivo o consulta (α):
  - Conjunción de literales positivos
  - Al negar el objetivo queda un cláusula con literales negativos
  - *Se denota "?P,Q" y* representa: ($P \wedge Q$) y al negarlo queda como la cláusula ($\neg P, \neg Q$).
  - Es por esto que el objetivo se escribe a veces como: "$\leftarrow P,Q$"
- Es decir, en Prolog ($KB \wedge \neg\alpha$) es un conjunto de cláusulas de Horn.

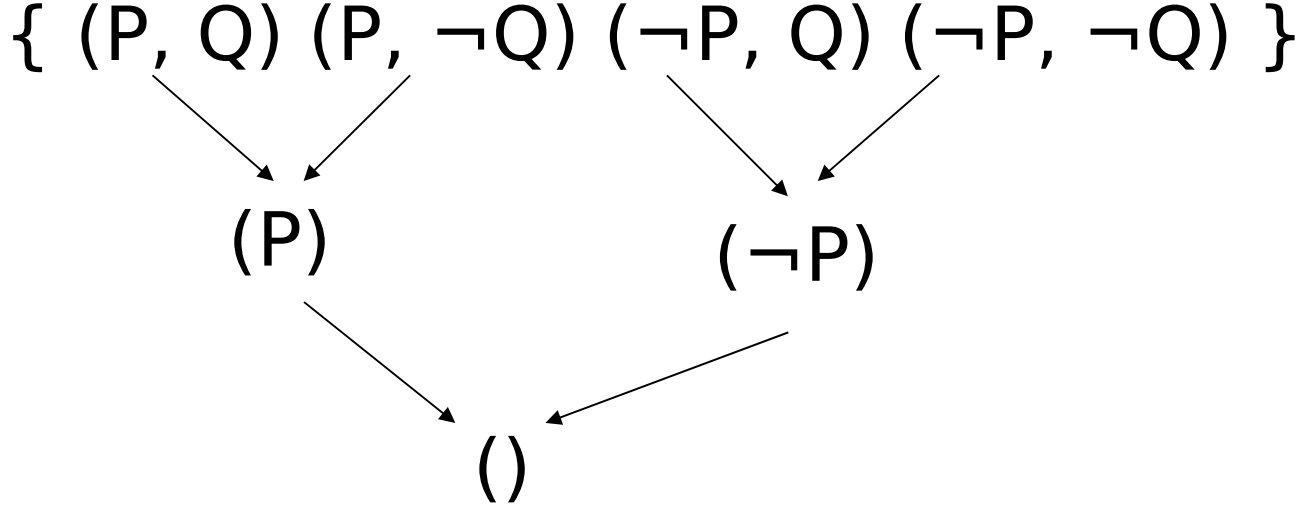
Recordar: Resolución Lineal

- Sólo permite resoluciones que usan al menos una cláusula en ($KB \wedge \neg\alpha$)
- Ejemplo anterior es resolución lineal:



Recordar: Resolución Lineal

- Problema: refutación basada en resolución lineal no es completa



Prolog proposicional vs Prolog

- Prolog Proposicional: Prolog sin variables.
- Prolog: predicados, variables, funciones, constantes.

Prolog y Resolución Lineal

- Teorema de Nerode & Shore:
  - Refutación mediante resolución lineal es completa para programas Prolog.
- Prolog implementa una variante de resolución lineal llamada resolución SLD.

Resolución SLD en Prolog Proposicional

- Dada un objetivo ←*Q1,Q2 (1)*
- Y una regla Q1←A,B,C (2) en el programa.
- Reemplazamos en (1) Q1 por el cuerpo de (2) obteniendo ←A,B,C,Q2 (3)
- Esto es equivalente a resolver (1) con (2)
- Se repite esta operación hasta obtener la clausula vacía “←”
- Prolog implemante SLD + estrategia primero en profundidad

Ejemplo

KB: (1) a <- b & c.
(2) b <- d & e.
(2') b <- c.
(3) b <- g & e.
(4) c <- e.
(5) d.
(6) e.
(7) f <- a & g.

Query: ?a

Derivation Attempt #1

<- a.	
<- b & c.	Select a; choose (1)
<- g & e & c.	Select b; choose (3)
<- g & c.	Select e; choose (6)
<- g & e.	Select c; choose (4)
<- g.	Select e; choose (6)

Select g: FAIL! no matching clause

Ejemplo (cont.)

KB: (1) a <- b & c.
(2) b <- d & e.
(2') b <- c.
(3) b <- g & e.
(4) c <- e.
(5) d.
(6) e.
(7) f <- a & g.

Query: ?a

Derivation Attempt #2

<- a.	
<- b & c.	Select a; choose (1)
<- d & e & c.	Select b; choose (2)
<- e & c.	Select d; choose (5)
<- c.	Select e; choose (6)
<- e.	Select c; choose (4)
<- .	Select e; choose (6)

QUERY IS TRUE: obtained answer

Resolución SLD

- **S**election rule:
  - Siempre se resuelve el primer literal del objetivo parcial
  - La resolución es guiada por el objetivo.
- **L**inear Resolution:
  - Produce resoluciones lineales
- **D**efinite Clauses:
  - Cláusulas de Horn
- Problema:
  - No necesariamente termina (Prolog es incompleto)
  - Esto debido a potenciales loops en el programa

Prolog

- Resolucion SLD
- 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)

Resolución SLD (Proposicional)

SDLRes (O: Objetivo)

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

Resolución SLD (Proposcional)

- ¿Importa la elección del átomo usado para resolver?
  - No, todos los átomos deben ser “resueltos”
- ¿Importa la elección de la regla para resolver?
  - Sí: malas elecciones pueden llevar a fracasos y backtracking.

Resolución SLD

SDLRes (O: Objetivo)

- Para todo átomo Ai de O
  - Para toda cláusula Cj←B1, …, Bn de KB y UMG σ para Ai y Ci:
    - 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 (cont.)

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".

Extracción de Respuestas en Prolog

- Entrega una respuesta única para consultas con variables, pero se puede forzar la busqueda de otras respuestas (usando la opción “;”).

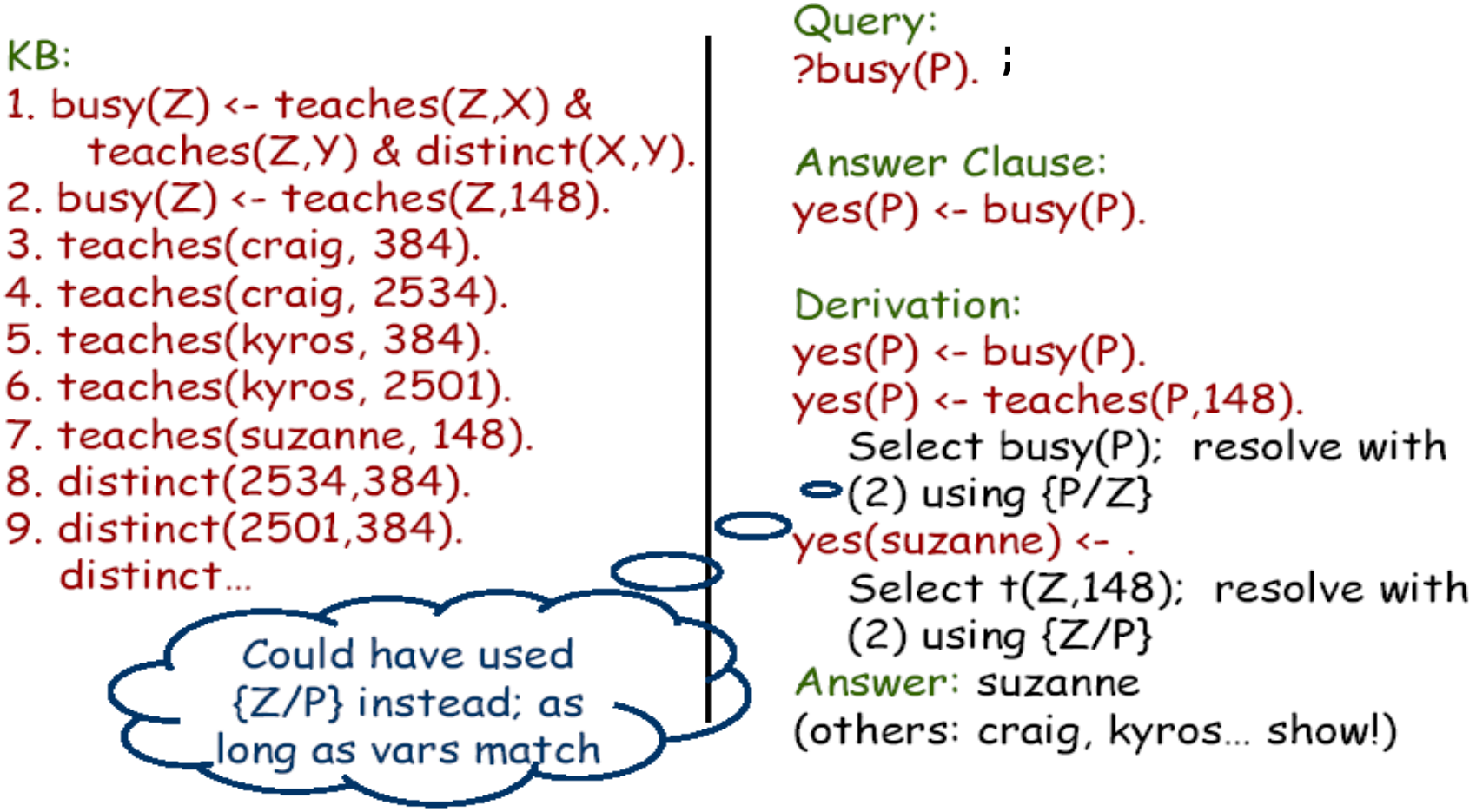
Resolución SDL con Extracción de Respuestas

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 “← resp(T)” agregar T a la respuesta.
    - SDLRes(O’)
- retornar(falso)

SLD con Extracción de Respuestas



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 Concatenaciónm (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!