# Programas definitivos y PROLOG

Pablo Barceló

# Programas definitivos

Una cláusula es definitiva si es de la forma

$$\{P(\bar{x}), \neg Q_1(\bar{x}_1), \ldots, \neg Q_n(\bar{x}_n)\} \qquad (n \geq 0)$$

La representamos por $P(\bar{x}) \leftarrow Q_1(\bar{x}_1), \ldots, Q_n(\bar{x}_n)$.

Un programa definitivo es un conjunto de cláusulas definitivas.

Como veremos luego, los programas definitivos son interesantes porque para analizar su semántica nos podemos restringir a un modelo de Herbrand particular...

# El modelo mínimo

Sea $P$ un programa definitivo. Defina $M_P$ como la intersección de todos los modelos de Herbrand de $P$.

Ejercicio: Demuestre que $M_P$ es un modelo de $P$ (lo llamamos el modelo mínimo).

Ejercicio: Demuestre que $M_P = \{A \in B_P \mid P \models A\}$, donde $B_P$ es la base de Herbrand de $P$.

Demuestre que esto no es cierto para programas arbitrarios (no definitivos).

En conclusión, podemos ver a $M_P$ como la interpretación natural de un programa definitivo $P$.

# Semántica procedural

Nos gustaría poder computar el valor de $M_P$, i.e. capturar $M_P$ proceduralmente.

Defina una función $f_P : 2^{B_P} \rightarrow 2^{B_P}$ como

$$f_P(I) = \{A \mid \text{para alguna instanciación } A \leftarrow A_1, \ldots, A_n \text{ de una clausula en } P \text{ se tiene que } \{A_1, \ldots, A_n\} \subseteq I\}$$

Ejercicio: Demuestre que $f_P$ es monótono.

Defina $f_P^0(\emptyset) = \emptyset$ y $f_P^{i+1}(\emptyset) = f_P(f_P^i(\emptyset))$.

Ejercicio: Demuestre que para cada $i \geq 0$, $f_P^i(\emptyset) \subseteq f_P^{i+1}(\emptyset)$.

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

Calcule $f_p(\emptyset), f_P(f_p(\emptyset)), \ldots$:

- $f_P(\emptyset) = \{P(f(f(a))\}$;
- $f_P^2(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a))))\}$;
- $f_P^3(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a)))), P(f(f(f(f(a))))\}$;
- .....

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

Calcule $f_p(\emptyset), f_P(f_p(\emptyset)), \ldots$:

- $f_P(\emptyset) = \{P(f(f(a))\}$;
- $f_P^2(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a))))\}$;
- $f_P^3(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a)))), P(f(f(f(f(a))))\}$;
- .....

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

Calcule $f_p(\emptyset), f_P(f_p(\emptyset)), \ldots$:

- $f_P(\emptyset) = \{P(f(f(a))\}$;
- $f_P^2(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a))))\}$;
- $f_P^3(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a)))), P(f(f(f(f(a))))\}$;
- .....

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

Calcule $f_p(\emptyset), f_P(f_p(\emptyset)), \ldots$:

- $f_P(\emptyset) = \{P(f(f(a))\}$;
- $f_P^2(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a))))\}$;
- $f_P^3(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a)))), P(f(f(f(f(a))))\}$;
- .....

# Ejemplo de semántica procedural

Considere el programa $P$ dado por:

$$\begin{aligned} P(f(x)) &\leftarrow P(x) \\ Q(a) &\leftarrow P(x) \\ P(f(f(a))) &\leftarrow \end{aligned}$$

Calcule $f_p(\emptyset), f_P(f_p(\emptyset)), \ldots$:

- $f_P(\emptyset) = \{P(f(f(a))\}$;
- $f_P^2(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a))))\}$;
- $f_P^3(\emptyset) = \{P(f(f(a)), Q(a), P(f(f(f(a)))), P(f(f(f(f(a))))\}$;
- .....

# Propiedades del operador $f_P$

Ejercicio: Sea $I$ una estructura de Herbrand para programa definitivo $P$. Demuestre que $I \models P$ ssi $f_P(I) \subseteq I$.

Ejercicio: Demuestre que $M_P = \bigcup_{i \geq 0} f_P^i(\emptyset)$.

De ahora en adelante denotamos a $\bigcup_{i \geq 0} f_P^i(\emptyset)$ por $\mathrm{lfp}(f_P)$.

# Ejemplo de semántica procedural vs declarativa

Sea grafo $G = (V = \{v_1, \ldots, v_n\}, E)$. Considere el siguiente programa $P$:

$$
\begin{array}{rcll}
E'(v_i, v_j) & \leftarrow & & (v_i, v_j) \in E \\
T(x, y) & \leftarrow & E'(x, y) & \\
T(x, z) & \leftarrow & E'(x, y), T(y, z) &
\end{array}
$$

Pregunta: ¿Cuál es el valor de $\mathrm{lfp}(f_P)$?

Pregunta: ¿Cuál es el valor de $M_P$?

Pregunta: ¿Cuál es la relación entre la interpretación de $E$ en $G$ y la interpretación de $T$ en $M_P$?

# Programas definitivos y resolución

En cierto sentido, el operador $f_P$ realiza forward chaining, y su búsqueda no es del todo guiada por el objetivo.

Además, no es propiamente un algoritmo puesto que el número de iteraciones puede ser infinito.

Si quisiéramos lograr un proceso backward chaining y guiado por el objetivo podríamos utilizar resolución.

Veremos que para programas definitivos no necesitamos todo el poder de resolución, sino una versión más restrictiva llamada Selection, Linear, Definite (SLD).

# Objetivos

Sea $P$ un programa definitivo, y $\phi$ una oración de la forma $\exists \bar{x} \alpha$, donde $\alpha$ es una conjunción de literales con variables en $\bar{x}$.

- Queremos verificar si $P \models \phi$, i.e. si $P \cup \{\neg\phi\}$ es insatisfacible.

Note que $\neg\phi$ es una cláusula de la forma

$$\leftarrow Q_1(\bar{x}_1), \ldots, Q_n(\bar{x}_n),$$

que llamamos objetivo.

Tal objetivo guiará la búsqueda de $\square$ a partir de $P \cup \{\neg\phi\}$.

# Resolución SLD

Sea $C$ una cláusula. Decimos que $C'$ es una variante de $C$ si $C'$ se puede obtener desde $C$ mediante un renombre de variables.

Sea $P$ programa definitivo y $G$ un objetivo de la forma $\leftarrow A_1, \ldots, A_n$.

Sea $C$ una variante de una cláusula en $P$ de la forma $A \leftarrow B_1, \ldots, B_k$, tal que $A$ y $A_i$ unifican mediante umg $\theta$, para $1 \leq i \leq n$.

Entonces el objetivo $G'$ de la forma $\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_m)\theta$ es un resolvente SLD de $G$ y $C$.

# Derivaciones SLD

Una derivación SLD desde $P \cup \{G\}$ es:

- Una secuencia $G = G_0, G_1, \ldots,$ de objetivos;
- una secuencia $C_0, C_1, \ldots$ de variantes de cláusulas en $P$; y
- una secuencia de umgs $\theta_0, \theta_1, \ldots,$

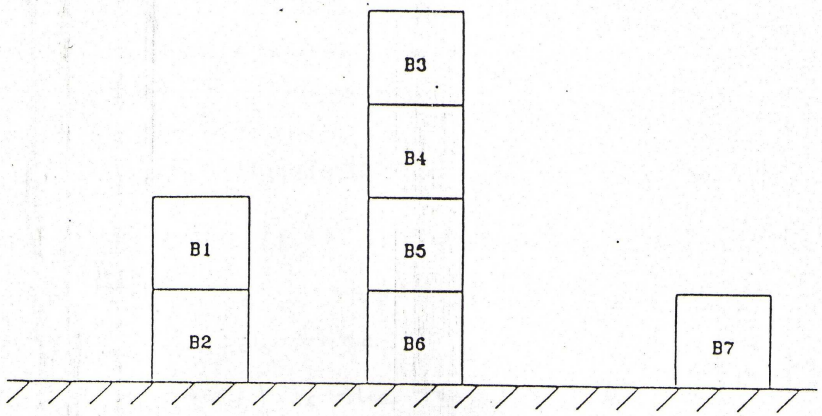
tal que para cada $i \geq 0$, $G_{i+1}$ es un resolvente SLD de $C_i$ y $G_i$ mediante umg $\theta_i$.

Si la secuencia $G_0, G_1, \ldots$ es finita, y su último elemento es $G'$, escibimos $P \cup \{G\} \vdash_{\mathrm{SLD}} G'$, y estamos particularmente interesados en el caso $P \cup \{G\} \vdash_{\mathrm{SLD}} \square$.

A continuación veremos un ejemplo.

# Ejemplo de resolución SLD

Example (top-down proofs: a different view of the same method)




The spatial relationships of interest will be:

ON($x$,$y$) - Block $x$ is on (touching) block $y$.

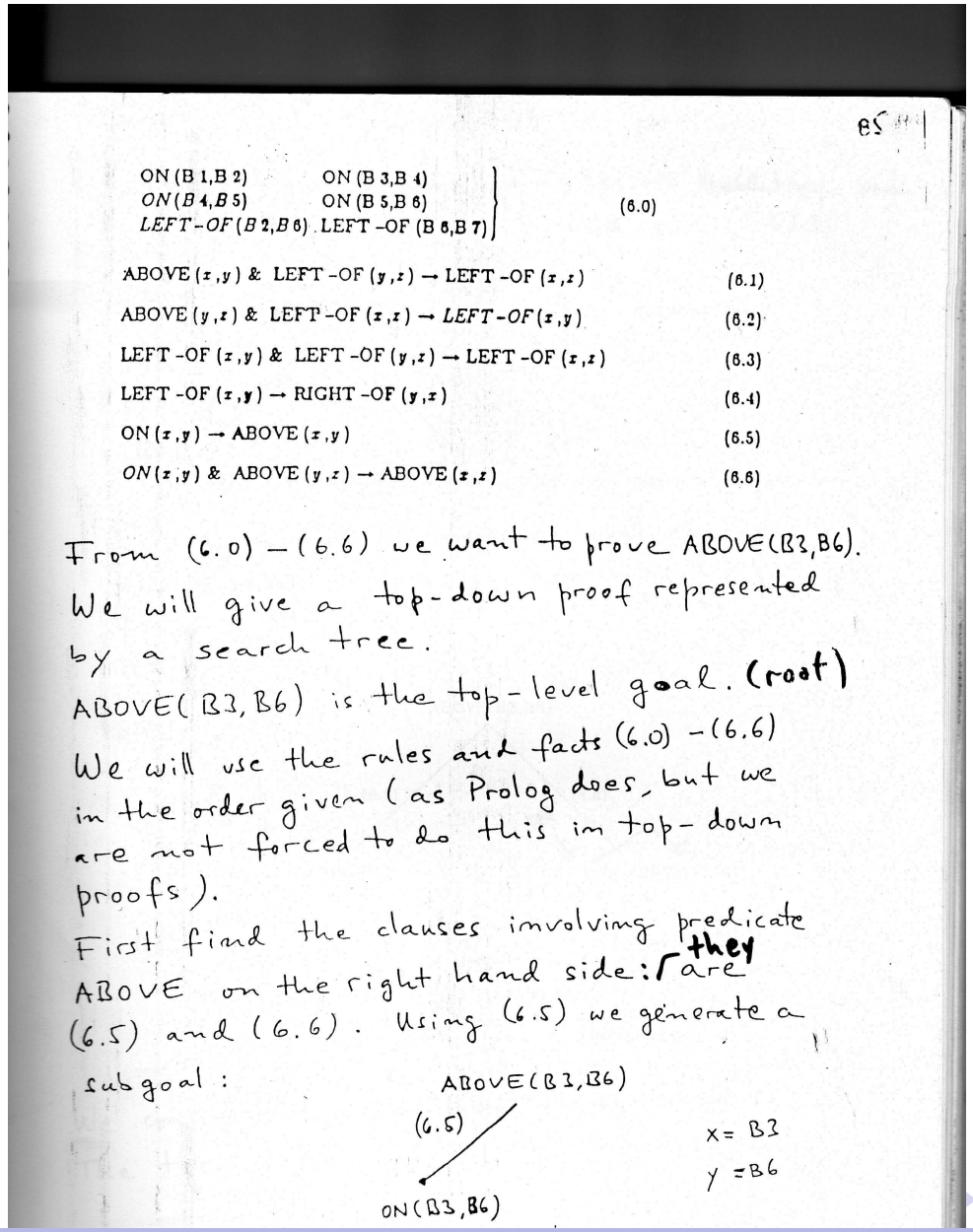
ABOVE($x$,$y$) - In the pile of blocks containing block $y$, block $x$ is above (not necessarily touching) $y$.

LEFT-OF($x$,$y$) - Block $x$ is to the left of block $y$.

RIGHT-OF($x$,$y$) - Block $x$ is to the right of block $y$.

We will use these predicates to describe our world. Let us consider the following rules:

# Ejemplo de resolución SLD



85

ON (B 1,B 2) ON (B 3,B 4)
*ON* (*B* 4,*B* 5) ON (B 5,B 6)
*LEFT-OF* (*B* 2,*B* 6) LEFT-OF (B 6,B 7) (6.0)

ABOVE (*x*,*y*) & LEFT-OF (*y*,*z*) → LEFT-OF (*x*,*z*) (6.1)

ABOVE (*y*,*z*) & LEFT-OF (*x*,*z*) → *LEFT-OF* (*x*,*y*) (6.2)

LEFT-OF (*x*,*y*) & LEFT-OF (*y*,*z*) → LEFT-OF (*x*,*z*) (6.3)

LEFT-OF (*x*,*y*) → RIGHT-OF (*y*,*x*) (6.4)

ON (*x*,*y*) → ABOVE (*x*,*y*) (6.5)

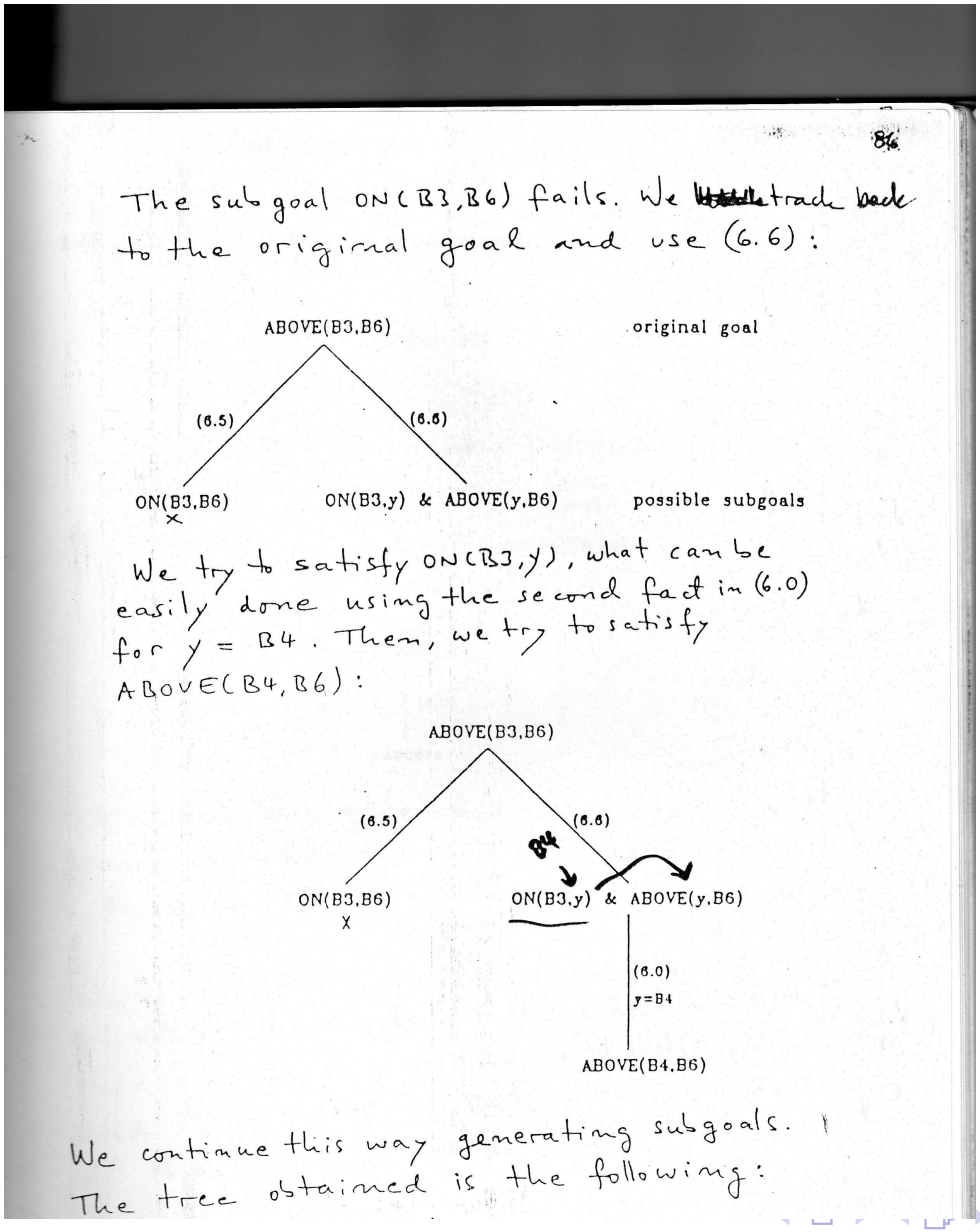
*ON* (*x*,*y*) & ABOVE (*y*,*z*) → ABOVE (*x*,*z*) (6.6)

From (6.0) – (6.6) we want to prove ABOVE(B3,B6).
We will give a top-down proof represented by a search tree.
ABOVE(B3, B6) is the top-level goal. (root)
We will use the rules and facts (6.0) – (6.6) in the order given (as Prolog does, but we are not forced to do this in top-down proofs).
First find the clauses involving predicate ABOVE on the right hand side: they are (6.5) and (6.6). Using (6.5) we generate a subgoal:

ABOVE(B3,B6)
(6.5)
ON(B3,B6)

x = B3
y = B6

# Ejemplo de resolución SLD



86

The subgoal ON(B3,B6) fails. We ~~[illegible]~~ track back to the original goal and use (6.6):

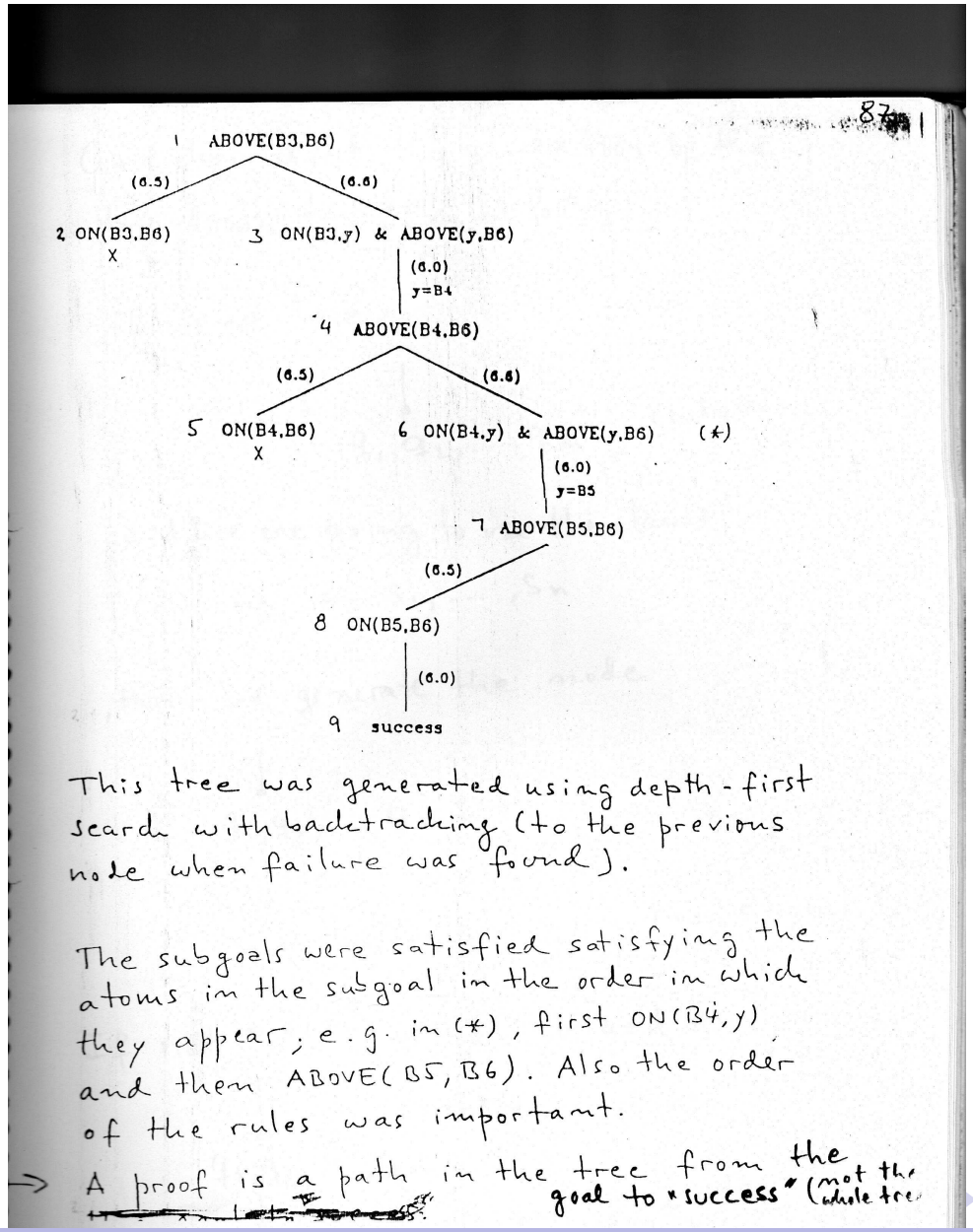



We try to satisfy ON(B3,y), what can be easily done using the second fact in (6.0) for y = B4. Then, we try to satisfy ABOVE(B4,B6):

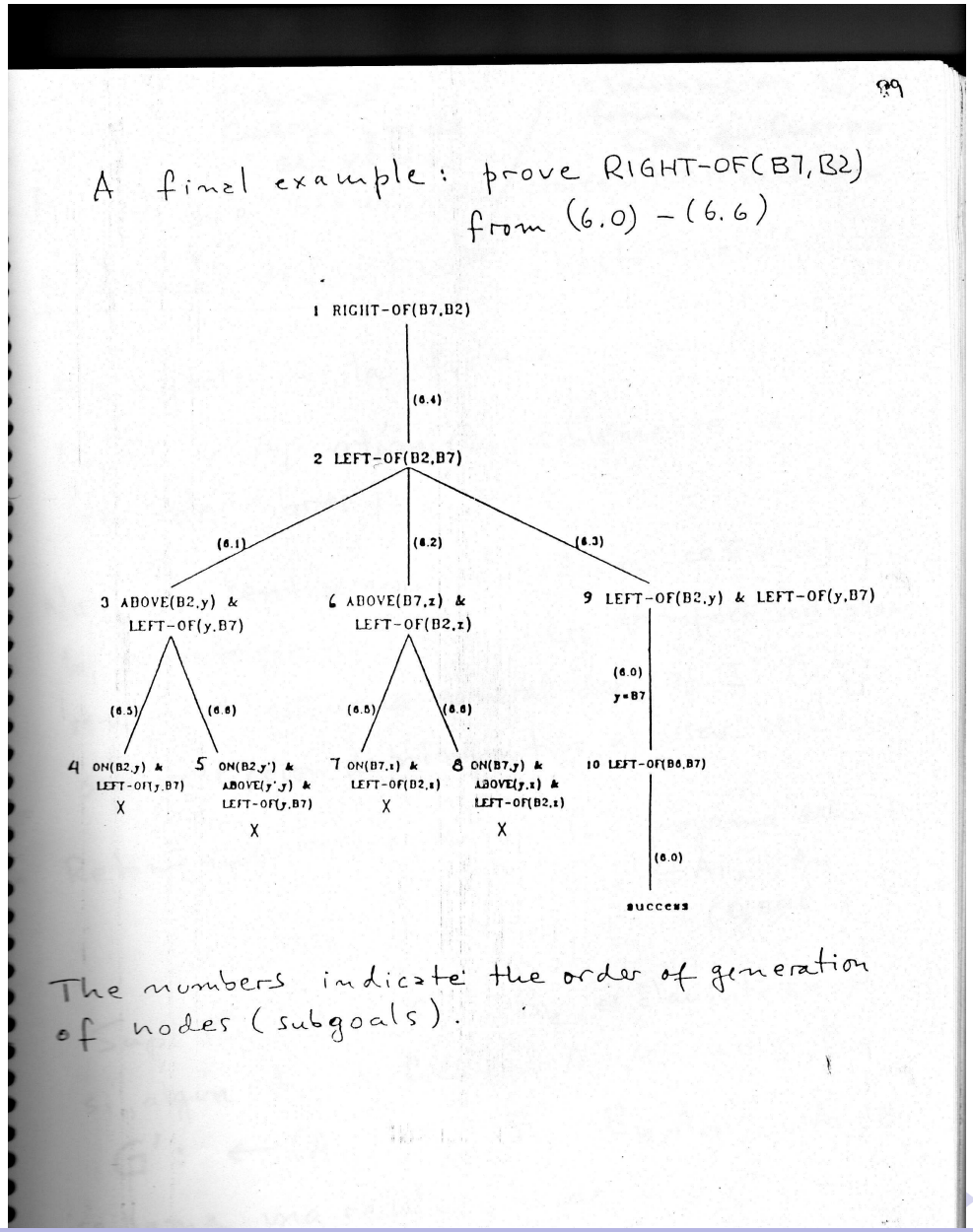



We continue this way generating subgoals. The tree obtained is the following:

# Ejemplo de resolución SLD



# Ejemplo de resolución SLD



# Reglas de selección

El arbol de resolución no es completo: está basado en una regla de selección de átomos a unificar en un objetivo.

Pregunta: ¿Cuál es la regla de selección que se aplica en el ejemplo anterior?

# Reglas de selección

El arbol de resolución no es completo: está basado en una regla de selección de átomos a unificar en un objetivo.

Pregunta: ¿Cuál es la regla de selección que se aplica en el ejemplo anterior?

Siempre unificar el primer átomo del objetivo. Esta es la regla de selección que utiliza PROLOG.

La regla de selección no es relevante ni para la corrección ni la completitud de la resolución SLD, aunque sí para su eficiencia.

# Objetivos con variables libres

Recuerde que agregar objetivo $G$ a $P$ equivale a agregar una fórmula de la forma $\neg\exists x_1, \ldots, x_s\, Q_1 \wedge \cdots \wedge Q_n$ a $P$.

Por tanto, si $P \cup \{G\} \vdash_{\mathrm{SLD}} \square$ mediante secuencia $\theta_0, \theta_1, \ldots, \theta_m$ de umgs, lo que concluimos es que $(Q_1 \wedge \cdots \wedge Q_n)\theta_0\theta_1 \cdots \theta_m$ es consecuencia lgica de $P$.

La restricción de $\theta_0\theta_1 \cdots \theta_n$ a las variables en $(Q_1 \wedge \cdots \wedge Q_n)$ se llama una respuesta computada para $P \cup \{G\}$.

Podría ser que en $(Q_1 \wedge \cdots \wedge Q_n)$ aún aparecieran variables. ¿Cómo deberíamos interpretar eso?

# Correctitud y completidad de SLD

Decimos que el proceso de resolución SLD es correcto.

- ¿Qué significa eso y cómo se demuestra?

Sea $P$ programa definitivo. Definamos $S_P$ como el conjunto

$$\{A \in B_P \mid P \cup \{\leftarrow A\} \vdash_{\mathrm{SLD}} \square\}.$$

Por el comentario anterior, $S_P \subseteq M_P$.

Pregunta: ¿Es cierto que $M_P = S_P$?

# Completidad de SLD

Antes de demostrar que resolución SLD es completa para programas SLD, demostraremos lo anterior:

Proposición: $M_P \subseteq S_P$ (y, por tanto, $M_P = S_P$).

Para demostrar esto necesitamos del siguiente lema:

Lema de sustitución: Si $P \cup \{G\theta\} \vdash_{\mathrm{SLD}} \square$, para alguna sustitución $\theta$, entonces $P \cup \{G\} \vdash_{\mathrm{SLD}} \square$.

Postpondremos la demostración del lema por ahora, y veremos primero por qué $M_P \subseteq S_P$ se sigue de él.

# Completidad de SLD

Recordemos que $M_P = \bigcup_{i \geq 0} f_P^i(\emptyset)$.

Es suficiente entonces si demostramos, por inducción en $i$, que $f_P^i(\emptyset) \subseteq S_P$.

Para $i = 0$, tenemos por definición que $f_P^0(\emptyset) = \emptyset$. Por tanto, trivialmente $f_P^0(\emptyset) \subseteq S_P$.

Asumamos por inducción entonces para $i \geq 0$. Demostraremos para $i + 1$.

Si $f_p^i(\emptyset) = f_P^{i+1}(\emptyset)$, entonces no hay nada que demostrar.

Asumamos entonces que existe $A \in B_P$ tal que $A \in f_P^{i+1}(\emptyset) \setminus f_P^i(\emptyset)$.

# Completidad de SLD

Esto implica que existe una instanciación $A \leftarrow B_1, \ldots, B_n$ de una cláusula en $P$ tal que $B_1, \ldots, B_n \in f_P^i(\emptyset)$.

Por hipótesis inductiva, $B_i \subseteq S_P$ para cada $1 \leq i \leq n$.

Pero, por tanto, $P \cup \{\leftarrow B_1, \ldots, B_n\} \vdash_{\mathrm{SLD}} \square$.

Considere ahora una resolución SLD de $\{\leftarrow A\}$ con la regla de $P$ cuya instanciación es $A \leftarrow B_1, \ldots, B_n$.

La resolvente SLD de éstos dos es una cláusula negativa que tiene como instanciación a $\leftarrow B_1, \ldots, B_n$.

Usando el lema concluimos que $P \cup \{\leftarrow A\} \vdash_{\mathrm{SLD}} \square$. Esto es lo que queríamos demostrar.

# Completidad de SLD

Usando la proposición y el lema demostraremos que resolución SLD es completo para programas definitivos.

Es decir, que si $P \cup \{G\}$ es insatisfacible, entonces $P \cup \{G\} \vdash_{\mathrm{SLD}} \square$.

Sea $G = \leftarrow A_1, \ldots, A_n$. Sabemos que $M_P \models P$, y , por tanto, $M_P \not\models G$.

Entonces , para alguna instanciación $\theta$ se tiene que $\{A_1\theta, \ldots, A_n\theta\} \subseteq M_P$.

Dado que $M_P \subseteq S_P$, se tiene que $P \cup \{\leftarrow A_1\theta, \ldots, A_n\theta\} \vdash_{\mathrm{SLD}} \square$.

Por el lema concluimos que $P \cup \{\leftarrow A_1, \ldots, A_n\} \vdash_{\mathrm{SLD}} \square$.

# Demostración del lema de sustitución

Finalmente demostraremos el lema de sustitución.

Sea $P$ un programa, $G$ un objetivo de la forma $\leftarrow A_1, \ldots, A_n$, y $\theta$ una sustitución. Asumimos que las variables de $G$ no aparecen en $P$ y que $\theta$ no actúa en las variables de $P$.

Demostraremos el lema mediante inducción en el largo $i > 0$ de la demostración por resolución de la cláusula vacía $\square$ desde $P \cup \{G\theta\}$.

Asuma primero que $i = 1$ (caso base). Entonces $n = 1$, y $A_1\theta$ unifica con un átomo $A \leftarrow$ de $P$. Por tanto, $A_1$ también unifica con ese átomo.

# Demostración del lema de sustitución

Asuma entonces por inducción para $i > 0$. Demostraremos para $i + 1$.

Sea $B_0 \leftarrow B_1, \ldots, B_m$ la primera cláusula de $P$ usada en la resolución SLD de $\square$ desde $P \cup \{G\theta\}$, y sea $A_i$, $1 \leq i \leq n$, el átomo de $G$ seleccionado por la resolución.

Luego, existe umg $\eta$ de $B_0$ y $A_i\theta$, i.e. $B_0\eta = A_1\theta\eta$.

Pero dado que $\theta$ no actúa en las variables de $P$, $B_0\theta\eta = A_1\theta\eta$.

Por tanto, $B_0$ y $A_1$ unifican mediante sustitución $\theta\eta$. Sea $\mu$ el umg de $A_1$ y $B_O$.

Por propiedades de umg, $\theta\eta = \mu\sigma$ para alguna sustitución $\sigma$.

# Demostración del lema de sustitución

De nuevo, porque $\theta$ no actúa en las variables de $P$ tenemos que:

$$P \cup \{\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\theta\eta\} \vdash_{\mathrm{SLD}} \square,$$

y la resolución toma $i$ pasos.

Por hipótesis inductiva,

$$P \cup \{\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\mu\} \vdash_{\mathrm{SLD}} \square.$$

Dado que $\leftarrow A_1, \ldots, A_n$ se resuelve en un paso con $B_0 \leftarrow B_1, \ldots, B_m$ en

$$\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n)\mu,$$

obtenemos el resultado.

# PROLOG

PROLOG (Colmerauer, Marsella, '70s) es el paradigma de la programación en lógica, en oposición a la programación tradicional.

PROLOG es un lenguaje de programación en lógica basado en las siguientes observaciones:

- Las reglas de un programa definitivo pueden ser vistas como procedimientos (por eso usamos las flechas en el sentido contrario al usual).
- La búsqueda de demostraciones descendentes puede ser entendida como una interpretación del programa.

Es decir, que para programas definitivos una lectura declarativa del programa es lo mismo que una lectura procedural.

Aplicaciones de PROLOG incluyen: Bases de datos, comprensión de lenguaje natural, web semántica, compiladores, etc.

# PROLOG: Tipo de búsqueda

Como dijimos, PROLOG implementa resolución SLD y la regla de selección que elige unificar siempre al primer átomo del objetivo.

Ninguno de estas dos características hace perder ni corrección ni completidad con respecto a programas definitivos.

Sin embargo, para tener un lenguaje de programación propiamente tal, PROLOG debe también fijar el tipo de búsqueda en el árbol de resolución.

PROLOG utiliza una estrategia en profundidad, donde el orden del árbol está dado por el orden en que las cláusulas aparecen en el programa.

Si el llamado a un procedimiento a través de una rama del árbol falla, entonces se realiza el *backtracking*.

# Estructuras de datos en PROLOG

Veremos que PROLOG presenta bastantes más funcionalidades que las que hemos visto hasta ahora.

Una de las más importantes es que en PROLOG podemos representar nuestras estructuras de datos favoritas, además de razonar acerca de ellas.

Para esto haremos uso extensivo de los símbolos de función.

# Estructuras de datos en PROLOG: Listas

Considere una función binaria *cons* y una constante *NIL*.

Una lista se define recursivamente como sigue:

- *NIL* es una lista;
- si $l$ es una lista y $b$ es una constante, entonces $cons(b, l)$ es una lista.

Ejemplos: $cons(a, NIL)$, $cons(a, cons(b, cons(b, NIL)))$, etc.

Es decir, en PROLOG una lista no es más que un tipo particular de término.

# Estructuras de datos en PROLOG: Listas

De ahora en adelante representaremos una lista no vacía por $[b_1, b_2, \cdots, b_n]$ (así es como las representa PROLOG), donde cada $b_i$ es una constante u otra lista.

Ejemplo: Defina en PROLOG el predicado $Member(x, y)$, que se tiene de todos aquellos elementos $x$ que pertenecen a la lista $y$.

# Estructuras de datos en PROLOG: Listas

De ahora en adelante representaremos una lista no vacía por $[b_1, b_2, \cdots, b_n]$ (así es como las representa PROLOG), donde cada $b_i$ es una constante u otra lista.

Ejemplo: Defina en PROLOG el predicado $Member(x, y)$, que se tiene de todos aquellos elementos $x$ que pertenecen a la lista $y$.

$$Member(X, [X, Y]) \quad \leftarrow$$
$$Member(X, [Y, Z]) \quad \leftarrow \quad Member(X, Z)$$

# Estructuras de datos en PROLOG: Listas

De ahora en adelante representaremos una lista no vacía por $[b_1, b_2, \cdots, b_n]$ (así es como las representa PROLOG), donde cada $b_i$ es una constante u otra lista.

Ejemplo: Defina en PROLOG el predicado $Member(x, y)$, que se tiene de todos aquellos elementos $x$ que pertenecen a la lista $y$.

$$Member(X, [X, Y]) \quad \leftarrow$$
$$Member(X, [Y, Z]) \quad \leftarrow \quad Member(X, Z)$$

Demuestre que $a$ es miembro de $[b, c, a, d]$.

# Concatenación de listas

Otro ejemplo: El predicado $Append(x, y, z)$ que es cierto si y sólo si $z$ se obtiene concatenando a $x$ e $y$.

# Concatenación de listas

Otro ejemplo: El predicado $Append(x, y, z)$ que es cierto si y sólo si $z$ se obtiene concatenando a $x$ e $y$.

$$Append([\,], X, X) \quad \leftarrow$$
$$Append([X, Y], Z, [X, W]) \quad \leftarrow \quad Append(Y, Z, W)$$

Ejercicio: ¿Cuál es el resultado de $Append(X, Y, [a, b, c, d])$?

# Ejercicios

Ejercicio: Defina el predicado $Last(x, y)$ que es cierto si $x$ es el último elemento de la lista no vacía $y$.

Ejercicio: Defina el predicado $Subset(x, y)$ que es cierto si todo elemento de la lista $x$ pertenece a la lista $y$.

Ejercicio: Defina el predicado $Reverse(x, y)$ que es cierto si $y$ es la lista inversa de $x$.

Ejercicio: Asuma que existe un predicado $Less(x, y)$ definido sobre una lista, y que significa que $x$ es menor que $y$.

Defina un predicado $Order(x, y)$ que para cada lista $x$ entrega la lista ordenada $y$ asociada con $x$.

# Ejercicios

Ejercicio: Defina el predicado $Insert(x, y, z)$, donde $y$ es una lista ordenada y $z$ es el resultado de insertar el elemento $x$ en $y$ preservando el orden de los elementos en $z$.

Ejercicio: Especifique un programa en PROLOG que mezcle dos listas ya ordenadas, entregando una lista ordenada.

# Ejercicios sobre operaciones aritméticas

Ejercicio: Defina las siguientes funciones aritméticas en PROLOG:

- Suma y multiplicación (ternarias);
- factorial (binaria);
- divisiń entera (ternaria).

Ejercicio: Defina un programa en PROLOG cuya salida sea el largo de una lista.

Ejercicio: Defina un programa en PROLOG que compute el máximo común divisor de dos enteros no negativos.