.UBAfiuba
FACULTAD DE INGENIERÍA

Programación Lógica

TB025 - Paradigmas de Programación

Curso Essaya

Paradigmas de Programación

- **Imperativos** (énfasis en la ejecución de instrucciones)
  - Programación Procedimental (ej. Pascal)
  - Programación Orientada a Objetos (ej. Smalltalk)
- **Declarativos** (énfasis en la evaluación de expresiones)
  - Programación Funcional (ej. Haskell)
  - **Programación Lógica (ej. Prolog)**

La Programación Lógica se basa en expresar **proposiciones, reglas y hechos lógicos**, y utilizar un **motor de inferencia** para resolver problemas.

Es un paradigma **declarativo**, ya que se enfoca en *qué* se quiere lograr en lugar de *cómo* lograrlo, permitiendo que el motor de inferencia se encargue de la ejecución.

Programación Lógica

Base de datos

```
padre(homero, bart).
padre(homero, lisa).
padre(homero, maggie).

madre(marge, bart).
madre(marge, lisa).
madre(marge, maggie).

hermanos(X, Y) :- padre(P, X), padre(P, Y),
                  madre(M, X), madre(M, Y),
                  X \= Y.
```

Consulta / Objetivo

```
?- hermanos(bart, X).
X = lisa ;
X = maggie.
```

Historia

1960s: Las ideas fundamentales surgieron de la demostración automática de teoremas y la investigación en IA. El avance clave fue el **Principio de Resolución** en 1965, una regla de inferencia única y poderosa para la lógica de primer orden que las computadoras podían implementar.

1970s: Se desarrolla la regla de **inferencia SLD**, una extensión del Principio de Resolución que permite la ejecución de programas lógicos.

Inspirados por estas ideas, Alain Colmerauer y Philippe Roussel desarrollaron **Prolog** (**PROgrammation en LOGique**). Su propósito inicial era procesar el lenguaje natural (francés).

1980s: Prolog y lenguajes similares son utilizados principalmente en ámbitos académicos y de investigación, para aplicaciones como **demostración de teoremas**, **sistemas expertos**, **sistemas de tipos** y **procesamiento de lenguaje natural**.

1990s: Prolog es estandarizado como **ISO Prolog**.

Definiciones

Proposición: una afirmación lógica que puede ser verdadera o falsa.

Ejemplo: “Hoy es martes”

Axioma / Hecho: una proposición aceptada como verdadera sin necesidad de prueba.

Regla: una proposición que define una relación entre hechos y/o otras reglas.

Ejemplo: “Si llueve, entonces la calle está mojada”

Base de datos: un conjunto de hechos y reglas que representan el conocimiento en un dominio específico.

Objetivo / Consulta: una proposición que se desea demostrar o refutar.

Estrategia de resolución: el proceso de buscar una solución a un objetivo utilizando la base de datos.

Programa lógico: Compuesto por una **base de datos** y un **objetivo**. Al ser ejecutado, un **motor de inferencia** se encarga de **probar** o **refutar** el objetivo, utilizando alguna **estrategia de resolución**.

Prolog: tipos de datos

Prolog es **dinámicamente tipado**. Todo valor en Prolog se representa mediante un **término**, que tiene varios subtipos:

Términos atómicos:

Átomos: `x`, `azul`, `hola_mundo`, `'Hola, Mundo!'`.
Números: enteros y números de punto flotante. Ejemplos: `42`, `3.14`.
Variables: `X`, `Resultado`, `_temp`.

Las variables son **unificadas** con otros términos durante la resolución de objetivos.

La variable `_` representa una variable anónima y significa “cualquier término”.

Términos compuestos:

Estructuras: consisten en un átomo (el **functor**) seguido de una lista de argumentos entre paréntesis. Ejemplos: `padre(juan, maria)`, `suma(X, Y, Resultado)`.

La **aridad** es el número de argumentos que tiene. La notación `f/n` se usa para referirse a un functor `f` con aridad `n`. Por ejemplo: `padre/2`.

Azucar sintáctica: Algunos predicados que pueden ser escritos como operadores **infijos**. Por ejemplo: `=(X, Y)` es equivalente a `X = Y`.

Listas: `[]`, `[1, 2, 3]`, `[X, Y]`

Si `T` es una lista, `[H|T]` es una lista con cabeza `H` y cola `T`.

Cadenas: `"Hola, Mundo!"`

Prolog: Cláusulas

Una base de datos Prolog se compone de un conjunto de **cláusulas**, que pueden ser **reglas** o **hechos**:

```
q :- p.   % regla
h.        % hecho
```

El operador `:-` se lee como $\Leftarrow$, de forma que la regla se lee como "`p` entonces `q`". El **encabezado** de la regla es `q`, y el **cuerpo** es `p`.

Dos o más cláusulas con el mismo functor se consideran parte del mismo **predicado**. Por ejemplo:

```
padre(homero, bart).
padre(homero, lisa).
padre(homero, maggie).
```

La **consulta** se escribe como `?- t.` donde `t` es un término. Por ejemplo:

```
?- padre(homero, X).
```

Predicados predefinidos

Disponemos de una gran cantidad de predicados predefinidos, por ejemplo:

`true` / `false` (`fail`): Siempre evalúan a verdadero o falso, respectivamente.

`A, B`: Verdadero si `A` y `B` son ambos verdaderos (*and*).

`A; B`: Verdadero si `A` o `B` (o ambos) son verdaderos (*or*).

`X == Y`: Verdadero si `X` e `Y` son términos idénticos.

`X \== Y`: Verdadero si `X` e `Y` no son términos idénticos.

`X = Y`: Unifica `X` e `Y` (falso si no son unificables).

`X \= Y`: Verdadero si `X` e `Y` no son unificables.

`var(X)` / `atom(X)` / `number(X)` / ...: Verdadero si `X` es una variable, átomo, número, etc.

Unificación

Dos términos **unifican** si pueden representar la misma estructura. Si alguno de los términos contiene variables, éstas se **instancian** para lograr la unificación.

Por ejemplo:

- `padre(homero, bart)` unifica con `padre(homero, bart)`.
- `padre(homero, bart)` no unifica con `padre(homero, lisa)`.
- `padre(homero, X)` unifica con `padre(homero, bart)` instanciando `X = bart`.
- `padre(P, X)` unifica con `padre(homero, bart)` instanciando `P = homero`, `X = bart`.
- `2 + 3` no unifica con `5`.

Para **resolver una consulta**, el motor de Prolog:

- Intenta unificar el objetivo con alguna de las cláusulas de la base de datos (buscando de arriba hacia abajo).
- Para las cláusulas de tipo `q :- p`, el objetivo debe unificar con `q`.
- Si la cláusula unificada tiene un cuerpo `p`, se intenta resolver la consulta `?- p.`, con las variables instanciadas en la unificación de `q`.

Backtracking

Es posible que un objetivo pueda ser unificado con más de una cláusula. En este caso, Prolog sigue una estrategia de búsqueda de tipo **depth-first**: unifica el objetivo con la primera de las alternativas y continúa la búsqueda. En caso de que alguno de los sub-objetivos falle, la búsqueda retrocede (**backtracking**), deshace todas las unificaciones hechas, y prueba con la siguiente alternativa.

Notar que el orden de las cláusulas en la base de datos afecta el resultado de la consulta, ya que Prolog intentará unificar las cláusulas en el orden en que aparecen.

Árbol de búsqueda

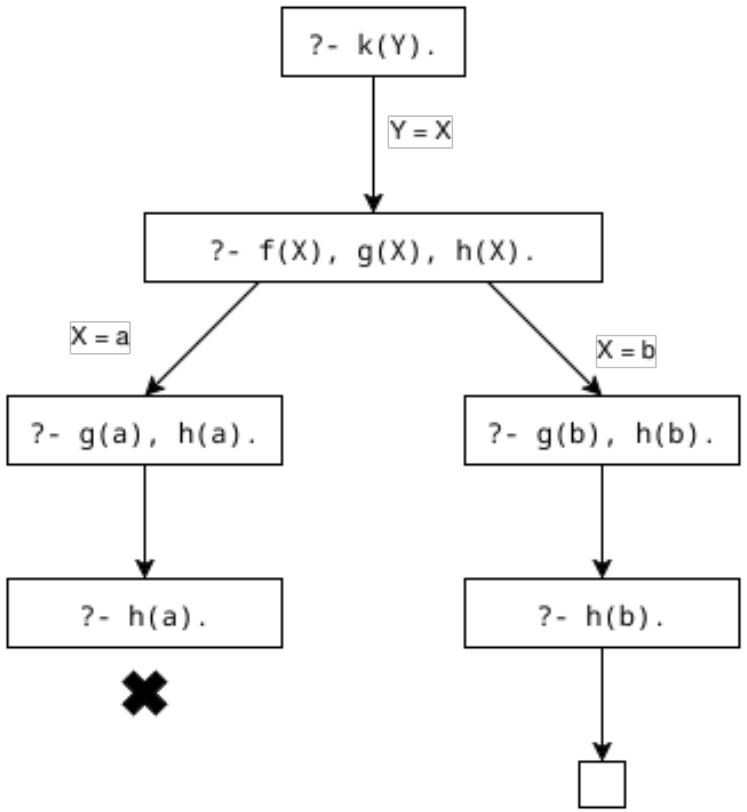
Un **árbol de búsqueda** es una manera de mostrar cómo Prolog resuelve una consulta.

Base de datos:

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X) :- f(X), g(X), h(X).
```

Consulta:

```
?- k(Y).
```



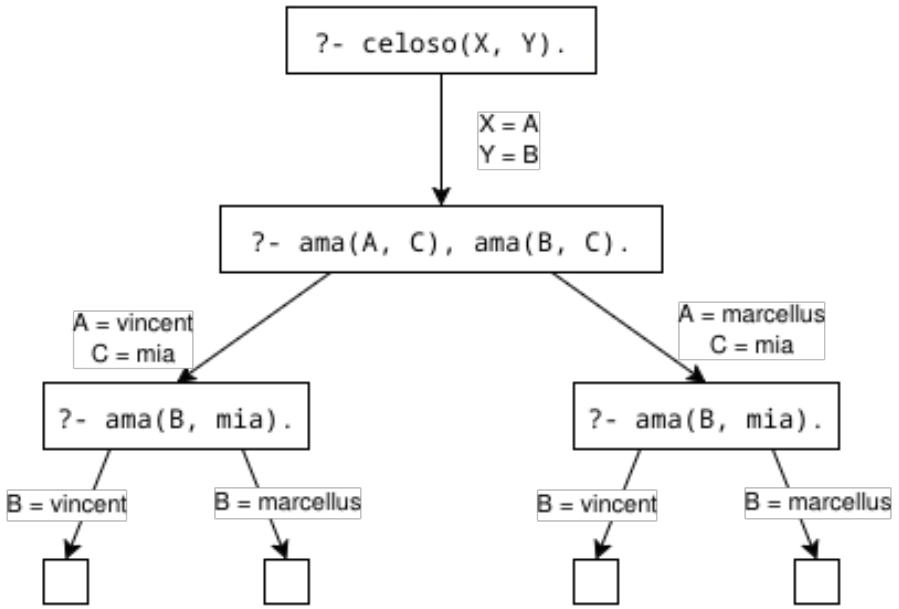
Árbol de búsqueda (cont.)

Base de datos:

```
ama(vincent, mia).
ama(marcellus, mia).
celoso(A, B) :- ama(A, C), ama(B, C).
```

Consulta:

```
?- celoso(X, Y).
```



El modelo de seguimiento de Prolog

Es podible *depurar* un programa en Prolog utilizando el predicado `trace`.

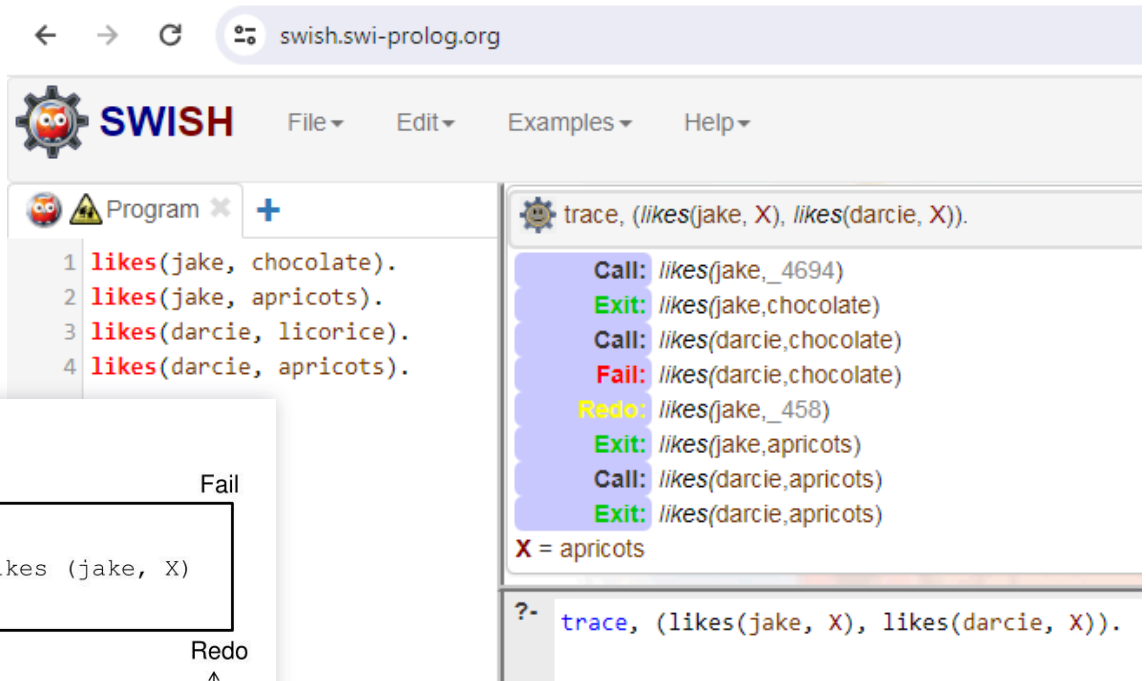
La ejecución del programa se describe de manera procedimental en términos de cuatro eventos:

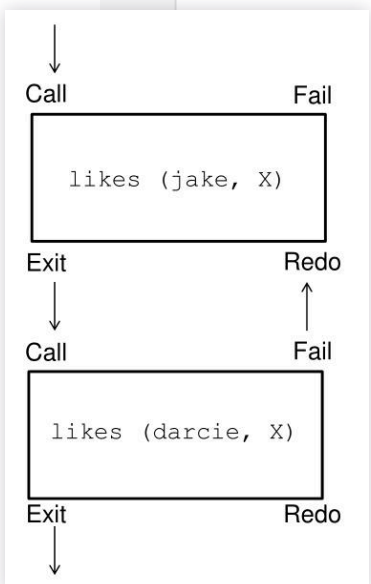
Call: Se intenta resolver un objetivo.

Exit: El objetivo se resuelve exitosamente.

Fail: El objetivo no puede ser resuelto.

Redo: Se reintenta resolver un objetivo para encontrar más soluciones.

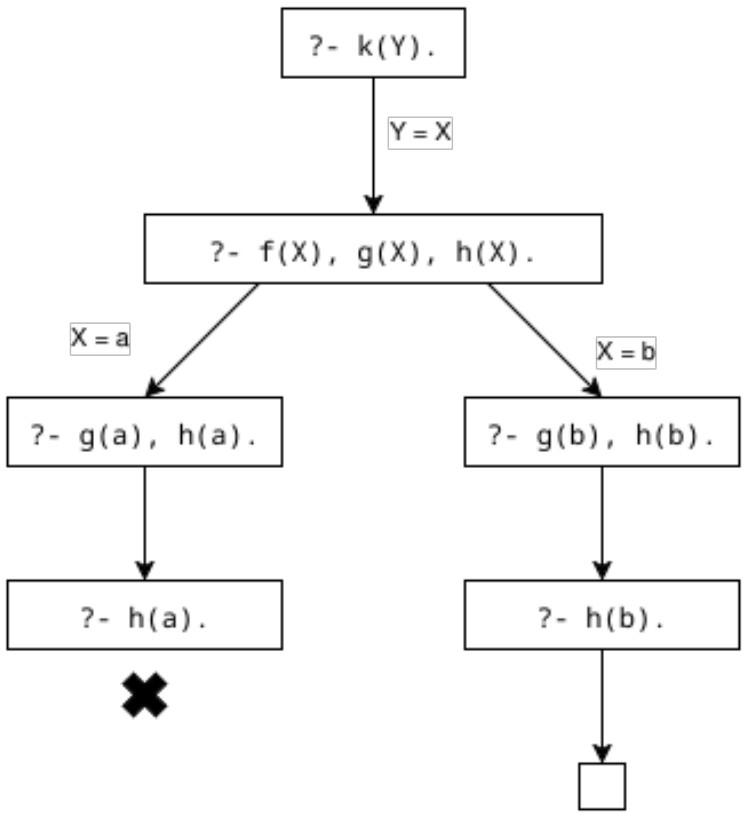


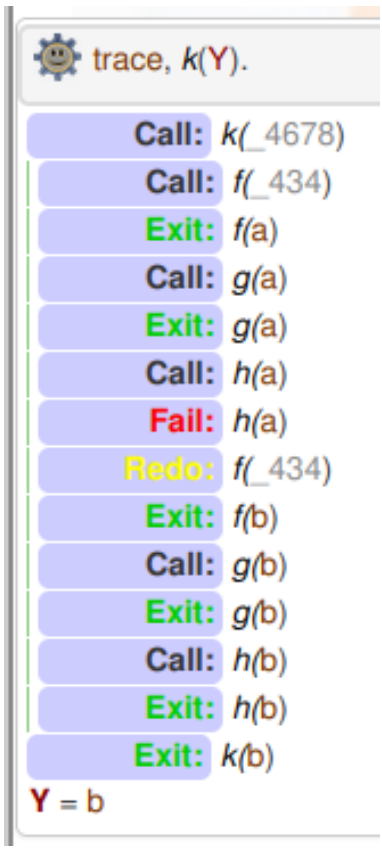


El modelo de seguimiento de Prolog (cont.)

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X) :- f(X), g(X), h(X).
```

```
?- k(Y).
```





Entrada/Salida básica

`read(X)`**:** Lee un **término** (terminado con `.`) de la entrada estándar y lo unifica con `X`.

`write(X)`**:** Escribe el **término** `X` en la salida estándar.

`nl`**:** Escribe un salto de línea en la salida estándar.

Predicados de control

`repeat`: Siempre tiene éxito, y provee un número infinito de soluciones mediante backtracking. Útil para generar bucles. Ejemplo:

```
pedir_clave :- repeat, write('Ingrese un numero: '), read(X), X == 42.
```

`!`: Proposición de corte. Siempre tiene éxito de inmediato, pero no puede ser resatisfecha a través del backtracking. Por lo tanto, las submetas a su izquierda, en una meta compuesta, tampoco pueden ser resatisfechas mediante backtracking.

```
f(a) :- !.
f(b).
```

`\+`: “No demostrable”. Verdadero si el objetivo no puede ser satisfecho.

```
ilegal(robar).
ilegal(evadir_impuestos).
legal(X) :- \+ ilegal(X).
```

El problema de la negación

Prolog adopta el **modelo de mundo cerrado**: **todo lo que no se pueda demostrar como verdadero, es falso**.

De esta manera el predicado `\+` implementa la **negación por falla**:

```
\+ P :- P, !, fail.
\+ _ :- true.
```

Esto puede llevar a resultados inesperados, dado que la *negación por falla* no es equivalente a la *negación lógica*.

Por ejemplo, dada la regla `legal(X) :- \+ ilegal(X).`, la consulta `?- legal(X)` no puede ser usada para listar todos los valores de `X` que son legales.

Aritmética

El operador `is` permite escribir predicados que involucran expresiones aritméticas, pero tiene algunas limitaciones. En su lugar, se recomienda usar la biblioteca `clp(fd)`:

```
:- use_module(library(clpfd)).
```

Comparaciones: `A #= B`, `A #< B`, `A #> B`, etc.
Operaciones: `A + B`, `A - B`, etc.

Ejemplo:

```
R = 2 + 3.   % R se unifica con el término +(2, 3)
R #= 2 + 3.  % R se unifica con el número 5
```

Listas

Los predicados que involucran listas suelen ser recursivos. Ejemplo:

```
longitud([], 0).
longitud([_|T], L) :- longitud(T, L1), L is L1 + 1.
```

Algunos predicados predefinidos para operar con listas:

`member(X, L)`: Verdadero si `X` es un elemento de la lista `L`.

`append(L1, L2, L3)`: Verdadero si `L3` es la concatenación de `L1` y `L2`.

`reverse(L, R)`: Verdadero si `R` es la lista `L` invertida.

`permutation(L1, L2)`: Verdadero si `L2` es una permutación de `L1`.

Links

SWI-Prolog:

Entorno de programación Prolog y recursos adicionales

SWISH:

Intérprete online de SWI-Prolog

SWISH Examples:

Ejemplos de programas escritos en SWI-Prolog

Learn Prolog Now!:

Libro online

The Power of Prolog:

Libro online

www.**ingenieria**.uba.ar

/ingenieriauba

/FIUBAoficial