LENGUAJES DE PROGRAMACIÓN

Paradigma Lógico
Caso de Estudio : Prolog

Lenguajes de Programación 2018

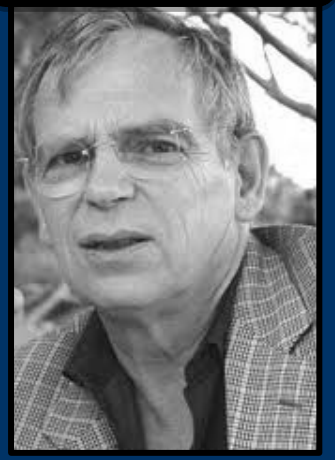
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca

1/2

- Una forma de razonar para resolver problemas se fundamenta en la lógica de primer orden.
- El conocimiento básico se puede representar en la lógica en forma de axiomas, a los cuales se le agregan reglas formales para deducir cosas verdaderas (teoremas).
- Los lenguajes que utilizan esta lógica son lenguajes declarativos porque todo lo que tiene que hacer el programador para solucionar un problema es describirlo vía axiomas y reglas de deducción (**Que** se quiere resolver **no Como**).

- Este concepto de programación lógica esta ligado históricamente a **Prolog** (Programmation en Logique), desarrollado por la Universidad de Marseille en 1972.
- Prolog es utilizado para:
  - el desarrollo de aplicaciones de I.A.,
  - la escritura de compiladores,
  - la construcción de sistemas expertos,
  - el procesamiento de lenguaje natural,
  - búsqueda de patrones y
  - programación automática.

Profesor
Alain Colmerauer



CARACTERÍSTICAS DEL PARADIGMA LÓGICO

A

ESQUEMA GENERAL

B

ELEMENTOS DEL PARADIGMA

C

PREDICADOS REVERSIBLES

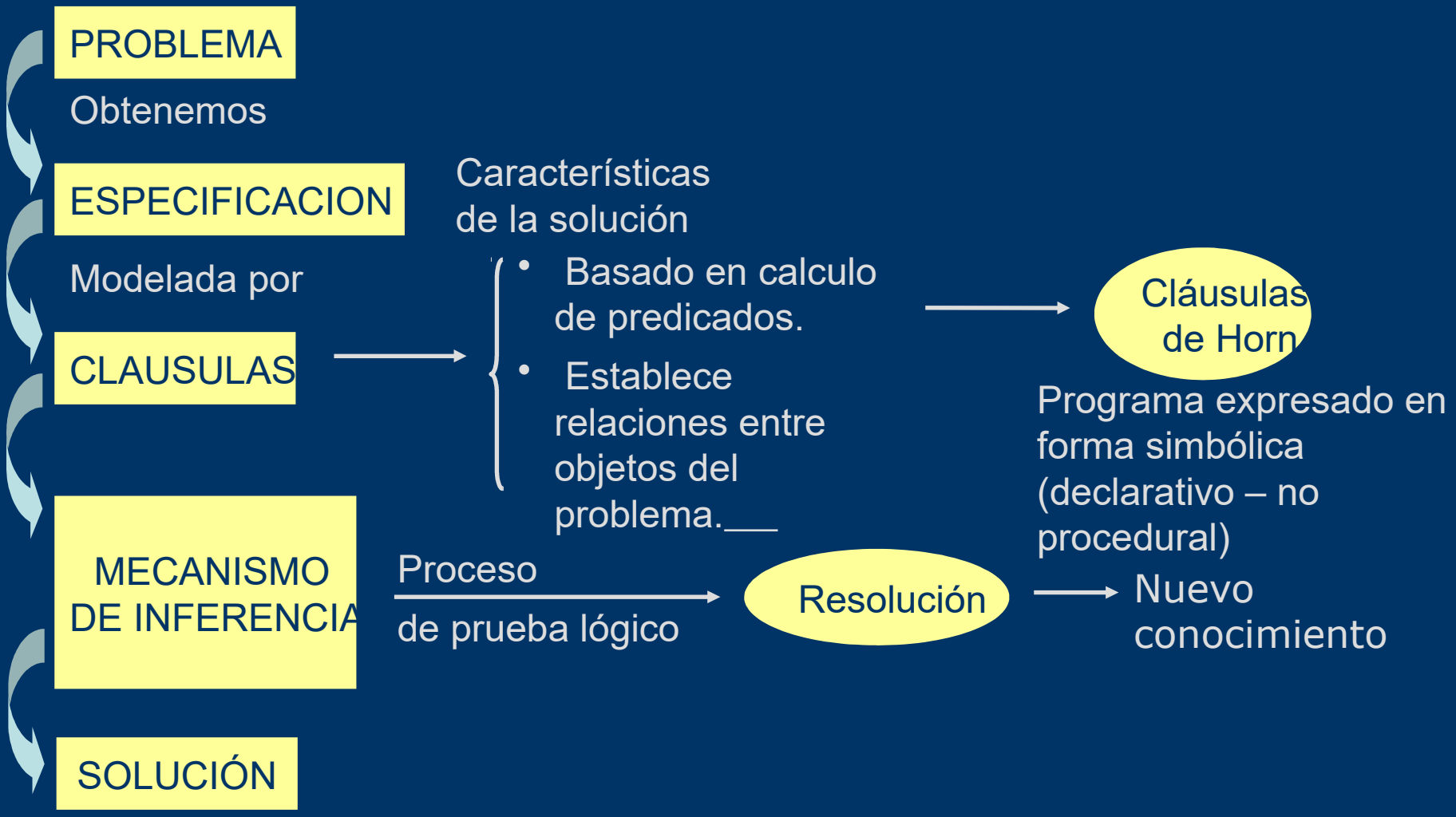
D

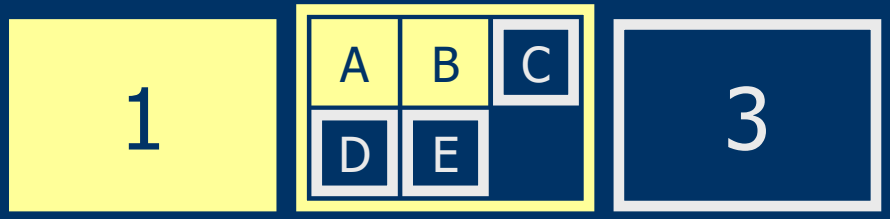
NOCIÓN DE TIPOS

E

CONSIDERACIONES SOBRE EL PARADIGMA

ESQUEMA GENERAL





Los términos en los lenguajes lógicos son **ciudadanos de primera clase!**

ELEMENTOS DEL PARADIGMA

- Los lenguajes lógicos constan de términos:
  - Una variable es un término.
  - Si f es un símbolo funcional n-ario y $t_1,\ldots,t_n$ son términos entonces $f(t_1..t_n)$ es un término (Si n = 0 → f es una constante).
  - Los términos son los argumentos de los predicados.
- Predicado: Si p es un símbolo relacional n-ario y $t_1..t_n$ son términos, entonces $p(t_1,..,t_n)$ es un predicado.

El conjunto de predicados conforman el programa lógico o base de conocimiento.

ELEMENTOS DEL PARADIGMA 2/9

- El paradigma lógico soporta el concepto matemático de variable.
- Las variables son nombres que representan valores desconocidos.
- No existe una asociación con una locación de memoria en la cual se almacenan distintos valores en distintos instantes de tiempo. Las variables se asocian a valores (Semántica de valores).
- El valor que puede tomar una variable consiste en cualquier término: j(3), 23.2, 'hola que tal', etc. Por eso decimos que los **datos** que se manejan **son términos**.

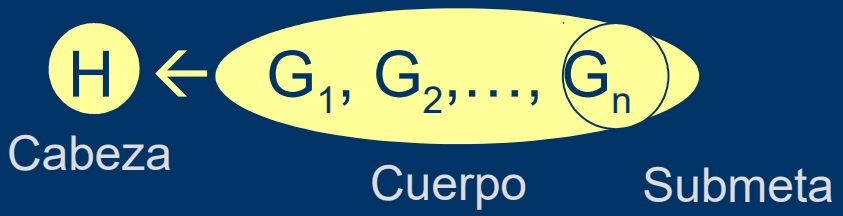
ELEMENTOS DEL PARADIGMA 2/9

- El paradigma lógico soporta el concepto matemático de variable.
- Las variables son nombres que representan valores desconocidos.
- No existe una asociación con una locación de memoria en la cuál se almacenan distintos valores en distintos instantes de tiempo. Las variables se asocian a valores (Semántica de valores).
- El valor que puede tomar una variable consiste en

Cuando una variable **no tiene valor** se dice que está **libre**.
Una vez que **obtiene valor**, éste ya no cambia,
por eso se dice que la variable está **ligada**.

ELEMENTOS DEL PARADIGMA

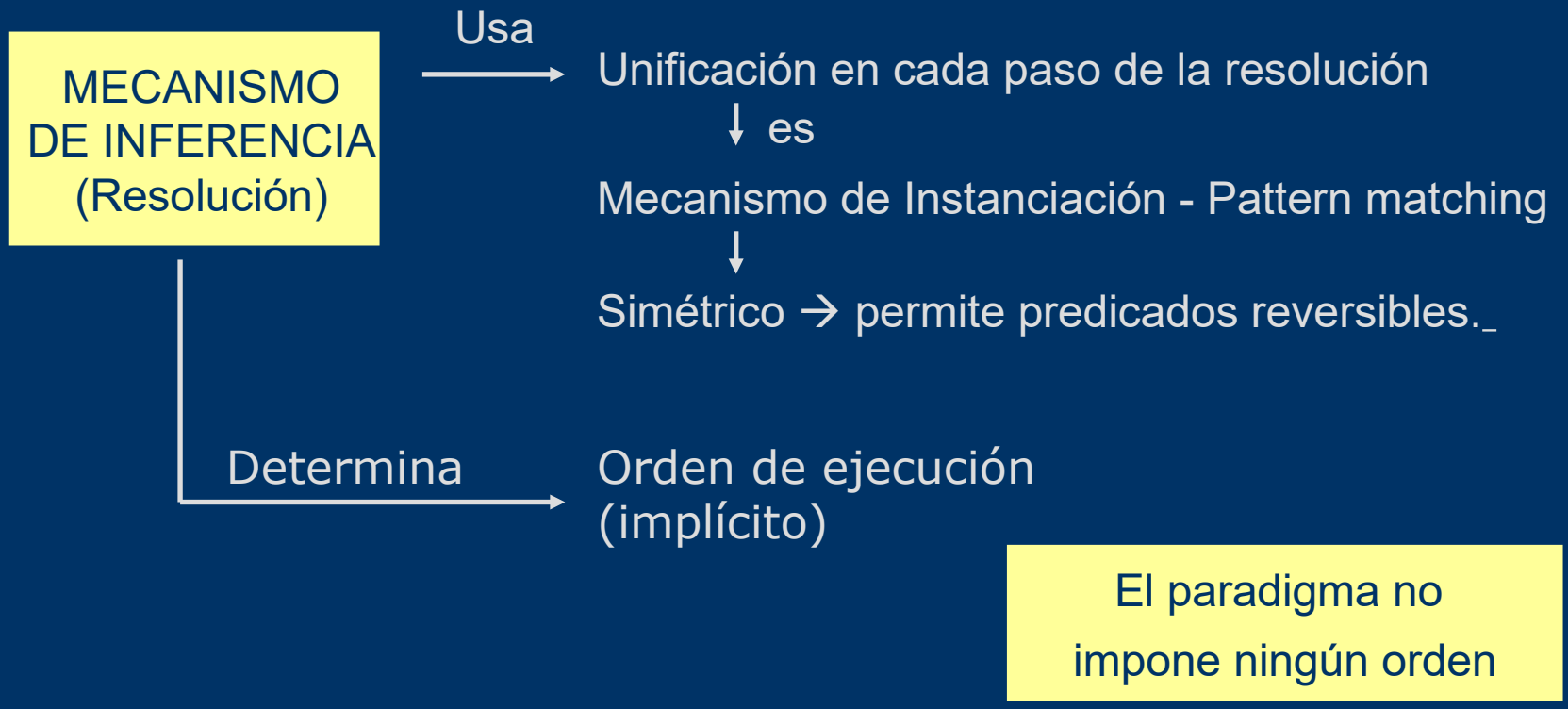
- Forma de una cláusula horn :



- El significado declarativo de la cláusula es:
  Si $G_1..G_n$ son todas verdaderas, entonces H es verdadera.
- Ejemplo de cláusulas de horn:
  - aprueba(X) ← estudia(X) ∧ rinde(X).
  - aprueba(X) :- estudia(X),rinde(X).

ELEMENTOS DEL PARADIGMA

- El mecanismo de inferencia permite derivar nuevo conocimiento y hallar soluciones.



ELEMENTOS DEL PARADIGMA

- Dado un programa lógico y una consulta compuesta por una o varias metas, el mecanismo de resolución intentará unificar alguna de dichas metas con la cabeza de alguna cláusula del programa, en algún orden.
- Como resultado se agregarán a la consulta todos los elementos del cuerpo de regla como metas

ELEMENTOS DEL PARADIGMA

- El proceso se repite hasta que:

  - alguna meta no pueda unificarse con ninguna cabeza, entonces la consulta fallará; o

  - para todas las metas existen hechos con los cuales unificar, por lo tanto la consulta es exitosa.

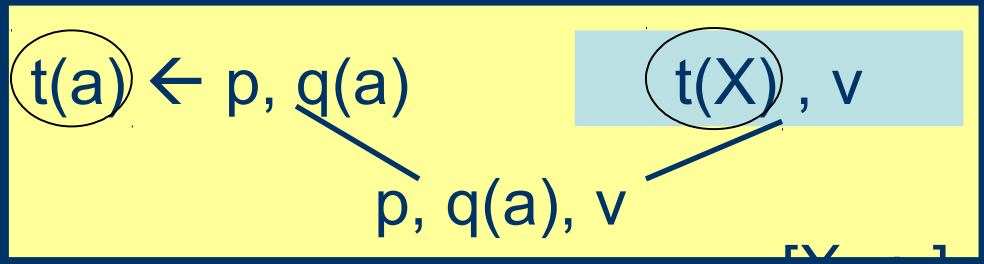
ELEMENTOS DEL PARADIGMA

Programa lógico

t(a) ← p, q(a)
p ← s, r
s.
r.
q(a).
v.

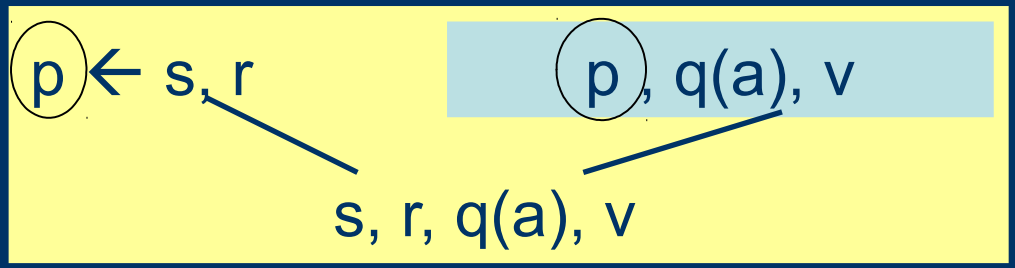
Consulta

t(X) , v ?



Nueva Consulta

p , q(a), v ?



Nueva Consulta

s , r, q(a) , v ?

ELEMENTOS DEL PARADIGMA

Comportamiento de las variables durante el proceso de resolución

- No todas las variables están obligadas a quedar ligadas
- Por ejemplo: h(X) y h(Y) unifican aunque las variables X e Y no quedan ligadas a un valor.
- No obstante, ambas variables permanecen unificadas entre sí.
- Si posteriormente ligamos X al valor 3, entonces automáticamente la variable Y tomará ese mismo valor.

ELEMENTOS DEL PARADIGMA 9/9

Para saber si dos términos unifican:

- Una variable siempre unifica con un término, quedando ésta ligada a dicho término.
- Dos variables siempre unifican entre sí, además, cuando una de ellas se liga a un término, todas las que unifican con ella se ligan a dicho término.
- Para que 2 términos unifiquen, deben tener el mismo functor y la misma aridad. Después se comprueba que los argumentos unifican uno a uno manteniendo las ligaduras que se produzcan en cada uno.
- Si 2 términos no unifican, ninguna variable queda ligada.

PREDICADOS REVERSIBLES

- Reversibilidad: propiedad que no existe en otros lenguajes, y que consiste en la habilidad de los argumentos de los predicados para actuar indistintamente como argumentos de entrada y/o salida.

PREDICADOS REVERSIBLES 2/2

- A las diferentes formas de invocar un predicado se las denomina modos de uso.
- Modos de uso:
  - Indican qué combinación de argumentos deben o no estar instanciados para que una consulta tenga sentido.
  - Se dice que un argumento está instanciado cuando no es una variable libre.

NOCIÓN DE TIPOS

- No se provee la noción de tipo de los lenguajes imperativos.
- Sólo se tienen términos y símbolos predicativos.
- La ausencia de un sistema de tipos permite hacer un mal uso de ciertos predicados. De esta manera, disminuye la seguridad y la confiabilidad.
- El programador debe encargarse de especificar axiomáticamente cualquier restricción que desea reflejar en el programa.

CONSIDERACIONES SOBRE EL PARADIGMA 1/2

- La lógica subyacente provee al paradigma un lenguaje riguroso, preciso, elegante, uniforme y ortogonal, donde se obtiene como resultado programas más legibles y concisos.
- Semántica de Valores.
- No existen efectos colaterales - Verificación del programa simplificada.
- No hay noción de Tipos de Datos → menor seguridad. Cualquier restricción es especificada por el programador mediante reglas.

CONSIDERACIONES SOBRE EL PARADIGMA 2/2

- El flujo de control es implícito ya que se rige por el mecanismo de resolución.
- El paradigma no establece un orden, por lo tanto es no determinista, lo que implica alguna forma de búsqueda, y backtraking.
- La unificación determina cuál o cuáles cláusulas matchean con la consulta realizada.
- No es eficiente en algunos casos → mayor tiempo en exploración en espacio de soluciones.
- El matching es simétrico, entonces permite predicados reversibles.

CASO DE ESTUDIO : PROLOG

A

PROLOG

B

CARACTERÍSTICAS QUE ALEJAN A PROLOG DEL PARADIGMA LÓGICO

C

PREDICADOS REVERSIBLES EN PROLOG

PROLOG

- Sintaxis y semántica del paradigma.
- Concepto de Ligadura: Unificación.
- Concepto de Variable: Semántica de valores.
- Evaluación de Expresiones
- Tipos de datos: No hay noción de tipos de datos.

CARACTERÍSTICAS QUE ALEJAN A PROLOG DEL PARADIGMA LÓGICO

1. Estrategia de búsqueda en el espacio de soluciones – Orden de evaluación.
2. Control del backtraking – Cut.
3. Negación por falla.
4. Evaluador aritmético is y predicados Relacionales – Aritmética simple.
5. Assert – Retract.

ESTRATEGIA DE BÚSQUEDA EN EL ESPACIO DE SOLUCIONES – ORDEN DE EVALUACIÓN

- Exploración del espacio de búsqueda:
  - Forward chainig: Hechos hacia consulta.
  - Backward chainig: Consulta hacia hechos.

Prolog

- Exploración del espacio de búsqueda (Backward chainig):
  - Recorrido a lo ancho (BFS): Garantiza encontrar una prueba si esta existe. Sensatez y completitud. Mayor tiempo de ejecución y menor eficiencia.
  - Recorrido en Profundidad (DFS): Sensatez, no completitud. Sensible a la forma de escribir la especificación.

ESTRATEGIA DE BÚSQUEDA EN EL ESPACIO DE SOLUCIONES – ORDEN DE EVALUACIÓN

- Exploración del espacio de búsqueda:

Prolog adopta:
- Backward chaining con
- DFS y
- Bactracking.

+ Este esquema utiliza menor cantidad de recursos
- Compromete la terminación de programas lógicos en caso de que se exploren ramas infinitas, aún cuando existen ramas exitosas de longitud finita.

- especificación.

ESTRATEGIA DE BÚSQUEDA EN EL ESPACIO DE SOLUCIONES – ORDEN DE EVALUACIÓN

- Orden de evalución: Prolog intenta unificar por la primera regla que aparece en el programa y evalúa las submetas de izquierda a derecha.
- Ejemplo: ?-conectada(a,b)

```
conectada(X,Y):- arco(X,Y).
conectada(X,Y):-  conectada(X,Z),
                  conectada(Z,Y).

arco(a,b).
```

```
conectada(X,Y):-  conectada(X,Z),
                  conectada(Z,Y).
conectada(X,Y):- arco(X,Y).

arco(a,b).
```

El programador debe tener en cuenta este orden para mejorar la performance de los programas y especialmente para no comprometer la terminación de los mismos

CONTROL DE BACKTRACKING – CUT

- Cut: predicado predefinido "!".
- Poda soluciones alternativas (hacia izquierda y abajo)
- Cuestión que debe ser analizada bajo interpretación procedural, permite al programador realizar control sobre el backtraking.
- Efecto colateral: acotar el espacio de búsqueda.
- Ventajas y desventajas:
  - \- Oscurece la semántica del programa lógico.
  + Implementaciones mas eficientes en algunos casos.
  + Elimina soluciones repetidas si se hace un uso cuidadoso del mismo.

CONTROL DE BACKTRACKING – CUT

- EJEMPLO: Supongamos que tenemos los siguientes hechos:

  p(a).
  p(b).
  p(c).

?- p(X),!.

X=a ;

no

Acá Prolog trata de buscar alguna respuesta más usando ';' pero ellas ya fueron cortadas.

CONTROL DE BACKTRACKING – CUT

- EJEMPLO: Supongamos que tenemos los siguientes hechos: r(a). r(b). r(c). s(a). s(b). s(c).
- Consideremos ahora:

```
?- r(X),!,s(Y).
X=a Y=a ;
X=a Y=b ;
X=a Y=c ;
no
```

Note que no hay backtracking en la primera meta.

- También,

```
?- r(X), s(Y), !.
X=a Y=a ;
no
```

NEGACIÓN POR FALLA

- La semántica de la negación por falla difiere de la negación de la lógica de predicados.

```
not(X) :- call(X), !, fail .
not(X).
```

- Ejemplo:

```
member(X, [X|_]) :- !.
member(X, [_|Xs]):- member(X,Xs).
```

```
?- not(not(member(X,[1]))).
X = _
```

EVALUADOR ARITMÉTICO *IS* Y PREDICADOS RELACIONALES

- El evaluador aritmético *is* y los operadores relacionales imponen restricciones en sus argumentos y no son reversibles.

Le quitan uniformidad al lenguaje

EVALUADOR ARITMÉTICO *IS* Y PREDICADOS RELACIONALES

- Cuando aparece una expresión en un predicado, no siempre es evaluada.

- p(2+2).
  ?- p(X).
  X = 2+2
  yes.

No se evaluó la expresió

EVALUADOR ARITMÉTICO *IS* Y PREDICADOS RELACIONALES

- Las expresiones son evaluadas cuando se utiliza el predicado *is* y cuando se utilizan operadores relacionales.

```
densidad( X, Y) :-
        poblacion( X, P),
        area( X, A),
        Y  is  P/A.
```

P y A deben estar instanciadas con valores aritméticos

¿Qué pasa si Y está instanciada al momento de la ejecución del *is*?

Los evalúa P/A y los compara por igualdad estructural

EVALUADOR ARITMÉTICO *IS* Y PREDICADOS RELACIONALES

```
?- X = 1 + 2, Y is X.
X = 1 + 2
Y = 3
Yes
```

```
?- 3 + 4  =  5 + 2.
No
```

```
?- 3 + 4  =:=  5 + 2.
Yes
```

- El predicado is y los operadores relacionales provocan la evaluación de sus argumentos.
- Ventajas y desventajas del predicado is:
  - \+ Facilidad para el programador.
  - \- Menos uniformidad para el lenguaje por restricciones en la ligadura entre variables y expresiones.
  - \- Falta de reversibilidad

PREDICADOS ASSERT Y RETRACT

- Permiten la incorporación y eliminación de claúsulas dinámicamente → efecto colateral.
- Ventajas y desventajas:
  + Facilita el desarrollo de aplicaciones que necesitan modificación dinámica de la base de conocimiento.
  - Uso cuidadoso, análisis procedural.
- Ejemplo:

```
a : - b, c.
c.
b :- retract(c).
```

```
?- a.
no
```

REFLEXIÓN Y META-PROGRAMACION

- Prolog provee herramientas para que el programador pueda utilizar y manipular su base de conocimiento y mecanismo de inferencia
- **assert, retract, !, fail**
- **clause(X,Y)**: Busca la una cláusula cuya cabeza es X y cuerpo Y.

Reflexión: un lenguaje es reflexivo si permite a sus programas razonar acerca de su propia estructura

Meta-computación: programas que pueden crear o manipular programas.

Prolog lo permite en su **máxima expresión**: crear y utilizar nuevos predicados en forma dinámica.

- Prolog provee herramientas para que el programador pueda utilizar y manipular su base de conocimiento y mecanismo de inferencia
- **assert, retract, !, fail**
- **clause(X,Y)**: Busca la una cláusula cuya cabeza es X y cuerpo Y.
- **call(X)**: llama a **X** como una **meta**.
- **=..** : Es un operador que permite transformar términos en listas y viceversa.

```
not(X) :- call(X), !, fail .
not(X).
```

PREDICADOS REVERSIBLES EN PROLOG 1/3

- EJEMPLO: Concatenación de listas.

```
conc([],L2,L2).
conc([X|L], L2, [X|Lr]):- conc(L,L2,Lr)
```

```
? – conc([1,2],[3],R).
R = [1,2,3]

? – conc([1,2],Z,[1,2,3,4]).
Z = [3,4]
```

```
? – conc(Y,[3,4],[1,2,3,4]).
Y = [1,2]

? – conc(Y,Z,[1,2,3,4]).
Y = []
Z = [1,2,3,4]
```

PREDICADOS REVERSIBLES EN PROLOG 2/3

- No todos los predicados son reversibles.
- EJEMPLO: Los predicados de comparación aritmética.

```
mayor(N1, N2) :- N1 > N2
?- mayor(3, 1).
yes

? - mayor(X, 1).
```

PREDICADOS REVERSIBLES EN PROLOG 3/3

- Dado el predicado suma/3 que suma en notación $s^n(0)$

```
suma(X,0,X).
suma(X, s(Y), s(Z)):- suma(X,Y,Z).
```

- Para sumar 1 + 1, podemos consultar:

```
?- suma(s(0), s(0), R).
R = s(s(0))
```

- El predicado suma/3 nos sirve para restar 1 – 1 consultando:

```
?- suma(s(0), R, s(0)).
R = 0
```

PREDICADOS REVERSIBLES EN PROLOG 3/3

- No hay noción de tipos de datos.
- Consideremos la definición anterior de suma. Es posible utilizar de la siguiente manera este predicado:

```
?- suma (algo, s(0), X).
X = s(algo)
```

- Para evitar este problema se puede restringir el formato de entrada de la siguiente manera:

```
num(0).
num(s(X)) :- num(X).
suma(X,0,X) :- num(X).
suma(X,s(Y),s(Z)) :- suma(X,Y,Z).
```

La seguridad y la confiabilidad de los programas está sujeta a la habilidad del programador.

BIBLIOGRAFÍA

- Scott Cap. 11.
- Sebesta Cap. 16