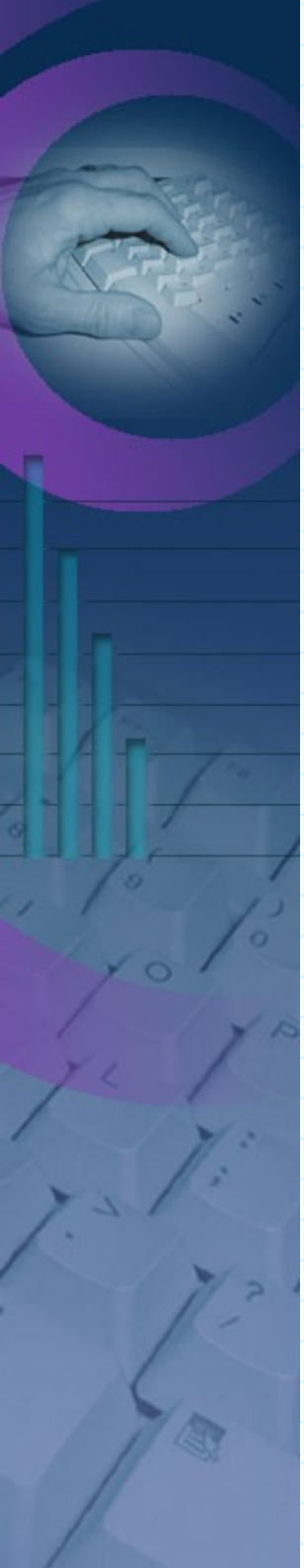

Tecnologías de Programación

Lenguaje de Programación Lógica

ProLog

Corte
Esquema condicional
Predicados Predefinidos

Prolog

El Corte

- Prolog realiza backtracking automático para resolver objetivos
- El programador queda librado de tener que programarlo en forma explícita
- Pero hay situaciones en las cuales para mejorar la eficiencia queremos prevenir efectos causados por backtracking
- Para controlar la búsqueda se utiliza el símbolo “!”, el cual aparece como un predicado más
- “!” siempre tiene éxito y provoca el descarte de todas las (ramas) alternativas que quedaron pendientes de ser exploradas **desde el instante en que se utilizó la regla que contiene** dicho “!”

Prolog

El Corte

- Supongamos las siguientes reglas:
  - R1: Si X < 3 entonces Y = 0
  - R2: Si 3 ≤ X < 6 entonces Y = 2
  - R2: Si 6 ≤ X entonces Y = 4
- En Prolog:
  - f(X, 0) :- write('Regla 1'), X < 3.
  - f(X, 2) :- write('Regla 2'), X >= 3, X < 6.
  - f(X, 4) :- write('Regla 3'), X >= 6.

Prolog

El Corte

- Si consultáramos:
  - f(1,Y), 2<Y.
- El resultado sería:
  - Regla 1Regla 2 Regla 3

    fail.
- Como se ve, trata de unificar las tres reglas, por más que sepamos son excluyentes según el valor del primer argumento

Prolog

El Corte

- Una solución más eficiente a la primera podría ser:
  - f(X, 0) :- write('Regla 1'), X < 3, !.
  - f(X, 2) :- write('Regla 2'), X >= 3, X < 6, !.
  - f(X, 4) :- write('Regla 3'), X >= 6, !.
- El resultado ahora sería:
  - Regla 1

    fail.

Prolog

El Corte

- Ahora que eliminamos el backtraking podemos reescribir las reglas de la siguiente forma:
  - f(X, 0) :- write('Regla 1'), X < 3, !.
  - f(X, 2) :- write('Regla 2'), X < 6, !.
  - f(X, 4) :- write('Regla 3').
- El punto a tener en cuenta en ésta última solución, es que se está cambiando el significado declarativo de las reglas.

Prolog

El Corte

- Los cortes se usan normalmente para:
  - No buscar soluciones en predicados alternativos
    - En éste caso se sabe que se ha escogido **la regla** adecuada para el objetivo y no tiene sentido seguir buscando otra, no la hay
  - Combinación de corte y fallo
  - Solo es necesaria la primera opción
    - En éste caso se sabe que se ha llegado a la **única solución**, o solo se necesita **alguna**, en cualquier caso no tiene sentido seguir buscando otras

Prolog

El Corte

- Combinación de corte y fallo
  - Nos permite indicarle a Prolog que falle inmediatamente sin intentar encontrar soluciones alternativas a la regla
  - La combinación de corte y fallo definen la negación (not) en prolog:
    - not(P):- P, !, fail.
    - not(P).
  - Tener en cuenta que la negación “not” definida de ésta forma siempre lleva implícito el corte

Prolog

El Corte

- Combinación de corte y fallo
  - Por legibilidad conviene usar la negación como fallo (not), en vez del corte-fallo. Siempre es posible sustituir el corte/fallo por el not.
  - Supongamos que necesitemos calcular la dosis de remedio a indicarle a una persona según su peso, podríamos indagar por: dosis(juan, 70, Dosis).
  - Supongamos que para indicarle el remedio a una persona, ésta debe cumplir ciertas condiciones, podríamos escribir:

Prolog

El Corte

- Combinación de corte y fallo

```
dosis(Persona, _, _) :- enfermo_renal(Persona), !, fail.
dosis(Persona, _, _) :- con_ulcera_gastrica(Persona), !, fail.
dosis(Persona, _, _) :-
        hipersensible_bromexhina(Persona), !, fail.
dosis(Persona, Peso, Dosis) :- Dosis is Peso * 2.
```

- O utilizando la negación:
  - dosis(Persona, Peso, Dosis) :-

```
not(enfermo_renal(Persona)),
not(con_ulcera_gastrica(Persona)),
not(hipersensible_bromexhina(Persona)),
Dosis is Peso * 2.
```

Prolog

El Corte

- Problemas
  - Cortes verdes: No afectan el significado declarativo de los programas. Los programas se pueden leer sin tener en cuenta los cortes
  - Cortes rojos: Afectan el significado declarativo de los programas. Es más fácil cometer errores, se debe prestar especial atención al significado procedural en los mismos.

Prolog

El Corte

- Ejemplo
  - p:- a, b. % $p \Leftarrow (a \wedge b) \vee c$
  - p:- c.

- El siguiente corte es rojo y cambia el significado
  - p:- a, !, b. % $p \Leftarrow (a \wedge b) \vee (\neg a \wedge c)$
  - p:- c.

- Invirtiendo los términos
  - p:- c. % $p \Leftarrow c \vee (a \wedge b)$
  - p:- a, !, b.

Prolog

El Esquema Condicional

- En Prolog la conjunción se simboliza con la coma (“,”) y la disyunción mediante la escritura de más de una regla para la misma cabecera.
- Otra forma de implementar la disyunción es mediante el punto y coma (“;”)

  - p :- a, b.
  - p :- a, c.
  - P :- d.

  - p :- a, b; a, c; d.

Prolog

El Esquema Condicional

- Por cuestiones de legibilidad, se recomienda usar varias cláusulas para el mismo predicado. Si se usa la notación de la derecha, es altamente recomendable separar criteriosamente los bloques en líneas distintas y usar paréntesis para mejorar la lectura
  - p :- (a, b);
    (a, c);
    d.

Prolog

El Esquema Condicional

- La cláusula condicional if-then-else se representa en prolog como “if -> then; else”

  - integer(1) -> write('es entero'); write('no es entero').

    --> es entero

  - integer(1.1) -> write('es entero'); write('no es entero').

    --> no es entero

Prolog

Predicados Predefinidos

- Son aquellos predicados ya definidos en prolog, según su uso pueden usarse para
  - Clasificación de términos
  - Control de otros predicados
  - Lectura/escritura y manejo de ficheros.
  - Construcción y acceso a componentes de estructuras.
  - Base de conocimientos

Prolog

Predicados Predefinidos

- Clasificación de términos
  - var/1: Se cumple si su argumento es una variable libre
  - Ejemplos
    - var(X). --> true.
    - X = 1, var(X). --> fail.

Prolog

Predicados Predefinidos

- Clasificación de términos
  - novar/1: Se cumple si su argumento no es una variable libre
  - Ejemplos
    - novar(X). --> fail.
    - X = 1, novar(X). --> true.
    - novar(a). --> true.
    - nonvar(1). --> true.

Prolog

Predicados Predefinidos

- Clasificación de términos
  - Integer/1, float/1 y number/1: Se cumplen respectivamente si el argumento se puede vincular a un valor entero, a un valor de punto flotante, o a un valor numérico en general.
  - Ejemplos
    - integer(a). --> fail.
    - integer(1). --> true.
    - integer(1.1). --> fail.
    - float(1.1). --> true.
    - number(1). --> true.
    - number(1.1). --> true.

Prolog

Predicados Predefinidos

- Clasificación de términos
  - groung/1: Se cumple si su argumento no contiene variables libres
  - Ejemplos
    - ground(X). --> fail.
    - X = 1, ground(X). --> X = 1.
    - ground(1 + 2). --> true.
    - ground(1 + X). --> fail.

Prolog

Predicados Predefinidos

- Control de otros predicados
  - true/0: Siempre se cumple
    - true. --> true.
    - not(true). --> fail.
  - fail/0: Siempre fracasa
    - fail. --> fail.
    - not(fail). --> true.
  - not/1: fracasa si el intento de satisfacer su argumento tiene éxito. Su argumento debe poder evaluarse como objetivo

Prolog

Predicados Predefinidos

- Control de otros predicados
  - repeat/0: siempre es exitoso, provee una forma de insertar infinitos puntos de elección

```
f(0, X) :- X = 1.
f(1, X) :-  X = 2.
f(2, X) :-  X = 3.
f(3, X) :-  X = 4.
iniciar :- repeat, read(X), not(X>3), not(X<0), f(X, Y),
                    string_concat('Y = ', Y, Z), write(Z).
```

Prolog

Predicados Predefinidos

- Control de otros predicados
  - call/1: se cumple si el intento de satisfacer su argumento tiene éxito. Su argumento debe poder evaluarse como objetivo

    - X = integer(1.1), call(X), Y is X.

      --> fail.

    - X = integer(1), call(X), Y is X.

      --> X = integer(1),
      Y = 1.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - write/1: siempre es evalúa como verdadero, escribe el término recibido como argumento en el canal de salida activo
    - write(a). --> a, true.
    - write(fail). --> fail, true.
    - write(X is 1 + 2) --> _G161 is 1 + 2, true.
  - nl/0: escribe un salto de línea en el canal de salida activo
    - write(a), write(b). --> ab.
    - write(a), nl, write(b). --> a
b.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - read/1: lee un término del canal de entrada activo y lo unifica con la variable indicada en su argumento. Para completar la entrada, se debe ingresar un punto “.” final.
    - read(X).

      – : **2.** %se ingresa el la cadena “2.”

      --> X = 2. %resultado%

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - display/1: funciona igual que write, pero no tiene en cuenta las declaraciones de operadores
    - display(a). --> a.
    - write(1 + 2) --> 1 + 2.
    - display(1 + 2) --> +(1, 2).
    - write([a, b, c]). --> [a, b, c].
    - display([a, b, c]) --> .(a, .(b, .(c, []))).

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - put/1: Escribe en el canal de salida activo el carácter cuyo código ASCII se recibe en el argumento
    - put(65). --> A, true.
    - put(X) --> Error si X no está instanciada.
    - X = 65, put(X) --> A, X = 65.
    - put(a) --> a, true.
    - put('A') --> A, true.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - get/1: Se cumple si su argumento corresponde al siguiente carácter en el canal de entrada activo
    - get(65). --> |: A --> true.
    - get(65). --> |: a --> fail.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - tell/1: Abre el fichero indicado en su argumento y lo define como canal de salida activo. Si el fichero no existe es creado. Si se utiliza tell con un fichero ya existente, el contenido de dicho fichero se destruye.
    - tell('prueba.txt'). --> true.
  - append/1: Abre un fichero para agregar datos, no destruye su contenido sino que agrega al final
    - append('prueba.txt'). --> true.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - telling/1: Se cumple si su argumento coincide con el identificador correspondient al stream de salida activo
    - telling(X). --> '$stream'(311756).
    - telling(X) --> user. % salida por defecto %
  - told/0: Cierra el canal de salida activo
    - told. --> true.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - see/1: Abre el fichero indicado en su argumento y lo define como canal de entrada activo. Si el fichero no existe se reporta un error.
    - see('prueba.txt'). --> true.
  - seeing/1: Se cumple si su argumento coincide con el identificador correspondient al stream de entrada activo
    - telling(X). --> '$stream'(311756).
    - telling(X) --> user. % salida por defecto %
  - seen/0: Cierra el canal de entrada activo
    - told. --> true.

Prolog

Predicados Predefinidos

- Lectura/escritura y manejo de ficheros
  - Ejemplo

    tell('pp.txt'). --> true.

    write('prueba.'). --> true.

    told. --> true.

    see('pp.txt'). --> true.

    read(X). --> X = prueba.

    seen. --> true.

Prolog

Predicados Predefinidos

- Construcción y Acceso a Estructuras
  - functor/3 (T, F, A): Se satisface si T es un término con functor F y aridad A
    - functor(a, a, 0). --> true.
    - functor(2 + 3, +, 2). --> true.
    - functor(2 + 3, X, 2). --> X = +.
    - functor(2 * 2 + 3, X, 2). --> X = +.

Prolog

Predicados Predefinidos

- Construcción y Acceso a Estructuras
  - arg/3 (A, T, V): T debe estar instanciado como un término y A a un entero entre 1 y la aridad de T. V se unifica con el valor del argumento indicado por A.
    - arg(1, 5 + 6, X). --> X = 5.
    - arg(2, 5 + 6, X). --> X = 6.
  - setarg/3 (A, T, V): los argumentos representan lo mismo que en arg/3, pero ahora se asigna el valor V al argumento A.
    - X = 5 + 6, arg(1, X, Y), setarg(1, X, 6), arg(1, X, Z), R is X.

    --> X = 6+6, Y = 5, Z = 6, R = 12.

Prolog

Predicados Predefinidos

- Construcción y Acceso a Estructuras
  - =../2: se utiliza para construir una estructura dada una lista de argumentos, el primer argumento representa el functor y el resto los argumentos del functor.
    - X =.. [+, 1, 2], Y is X. --> X = 1+2, Y = 3.
    - X =.. [integer, 3.1]. --> X = integer(3.1).

Prolog

Predicados Predefinidos

- Base de Conocimiento
  - Prolog ofrece mecanismos para manipular la base de conocimiento en forma dinámica
  - Se puede:
    - Observar la base actual: listing
    - Agregar clausulas: assert
    - Quitar cláusulas: retract

Prolog

Predicados Predefinidos

- Base de Conocimiento
  - listing/0: lista todas las cláusulas definidas en la base de conocimiento
  - listing/1: lista todas las cláusulas definidas en la base de conocimiento que tienen como predicado el átomo que se pasa como parámetro. El parámetro pasado puede ser de la forma functor o functor/aridad
    - listing.
    - listing(f).
    - listing(f/1).

Prolog

Predicados Predefinidos

- Base de Conocimiento
  - assert/1: permite añadir una nueva cláusula a la base de conocimientos.
  - asserta/1 y assertz/1: idénticas al anterior, pero la a y la z le indican si la cláusula se agrega al principio o al final del cuerpo de conocimiento.
    - asserta(f(2)).
    - asserta(f(1)).
    - assertz(f(3)).
    - listing(f/1). --> f(1). f(2). f(3). true.

Prolog

Predicados Predefinidos

- Base de Conocimiento
  - retract/1: Elimina de la base de conocimientos las cláusulas que unifican con el átomo ingresado como argumento
  - retractall/1: Elimina de la base de conocimientos las cláusulas cuya cabecera unifican con el átomo ingresado como argumento

Supongamos:

```
1) f(0).
2) f(1).
3) f(2).
4) f(3):- write('hola').
```

```
retract(f(0)). %elimina 1
retract(f(X)). %elimina 1, 2, 3
retractall(f(0)). %elimina 1
retractall(f(X)). %elimina todos
```