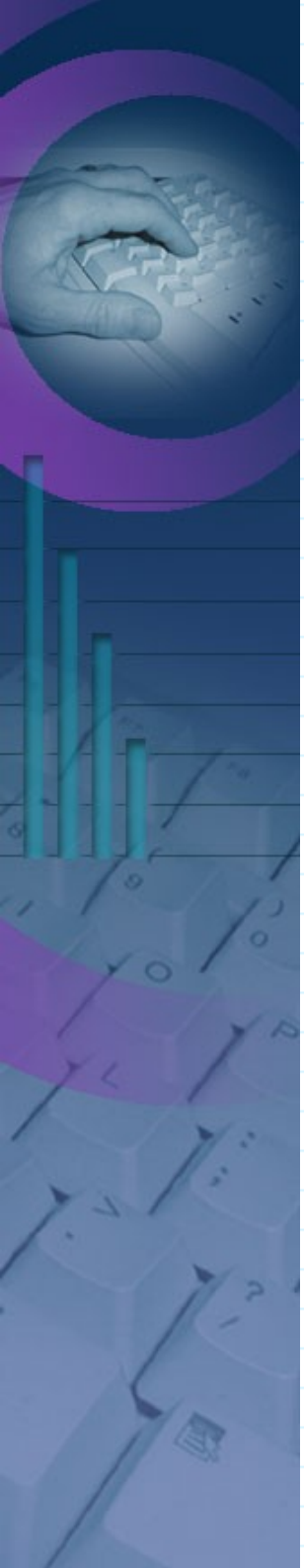

Tecnologías de Programación

Paradigma Lógico – ProLog

Predicados Predefinidos I

Predicados Predefinidos

- Los predicados predefinidos son aquellos que ya vienen definidos en Prolog, por lo que no necesitamos especificarlos.

- Dos grandes grupos
  - Predicados “comunes” predefinidos en Prolog pero que podríamos definir nosotros tranquilamente.
  - Predicados con un efecto colateral distinto a la ligadura de variables a valores.

El Esquema Condicional

- En Prolog la conjunción entre dos o más clausulas se expresa separando las mismas con coma (“,”):
  - clausula1, clausula2, ..., clausulaN.
- La disyunción es expresada declarando más de una cláusula para el mismo predicado:
  - pertenece(X, [X | _]) :- !.
  - pertenece(X, [_ | L]) :- pertenece(X, L).

El Esquema Condicional

- Otra forma de expresar la disyunción es a través del predicado predefinido punto y coma (“;”):
  - pertenece(X, [Y | L]) :- X = Y, ! ; pertenece(X, L).

El uso del punto y coma (“;”) es equivalente a la declaración de varias cláusulas para el mismo predicado, sin embargo se recomienda acotar su uso por cuestiones de legibilidad.

El Esquema Condicional

- La cláusula condicional if-then-else se representa en prolog como “if -> then; else”:
  - integer(1) -> write('entero'); write('no entero').
    - entero
  - integer(1.1) -> write('entero'); write('no entero').
    - no entero

Clasificación de Términos

- Permiten determinar el tipo de término al que nos referimos:
  - **var(+Term), novar(+Term)**: se cumplen si Term es una variable no instanciada, o si no lo es, respectivamente.
    - var(X). → true
    - var(1). → false
  - **integer(+Term), float(+Term), number(+Term)**: se cumple si X representa un entero, un punto flotante, o un número en general respectivamente.
    - integer(1). → true
    - number(1). → true

Clasificación de Términos

- **atom(+Term)**: se cumple si X representa un átomo en Prolog.
  - atom(1). → false
  - atom(juan) → true
  - atom(1+1) → false

- **atomic(+Term)**: se cumple si X representa un elemento atómico (átomo, string, número).
  - atomic(1). → true
  - atomic(juan) → true
  - atomic([]) → true
  - atomic(1+1) → false

Clasificación de Términos

- **ground(+Term)**: se cumple si X no tiene variables libres.
  - ground(1 + 2 + 3). → true
  - ground(1 + X + 3). → false
- **is_list(+Term)**: se cumple si Term es una lista.
  - is_list([]). → true
  - is_list([1, 2, 3, [a, b]]). → true
  - is_list(1). → false
  - is_list(X). → false

Predicados de Control

- Permiten controlar la evaluación de otros predicados:
  - **“!” (corte)**: operador de corte
  - **true, fail**: objetivos que siempre se cumple o fracasa respectivamente.
  - **not(+Goal)**: siendo Goal un término que puede interpretarse como objetivo, not(+Goal) se cumple si el intento por satisfacer Goal fracasa
  - **repeat**: siempre es exitoso, provee una forma de insertar infinitos puntos de elección al momento de la evaluación del mismo

Predicados de Control

- **call(+Goal)**: siendo Goal un término que puede interpretarse como objetivo, call(Goal) se cumple si se cumple el intento por satisfacer Goal.
  - X = integer(1), call(X). → true
  - X = integer(a), call(X). → false

- **call(+Goal, +ExtraArg1, ...)**: siendo Goal un predicado que se pueda invocar y el resto de los argumentos, los argumentos de dicho predicado.
  - call(append, [1, 2, 3], [a], X).

    → X = [1, 2, 3, a]

Predicados de Control

- **ignore(X)**: invoca X, y se evalúa siempre verdadero.
  - ignore(append([1], [2], X)).
    → X = [1, 2]
  - ignore(append([1], [2], [1])).
    → true
- **“,” (coma)**: especifica una conjunción de objetivos.
  - a, b. → a ∧ b
- **“;” (punto y coma)**: especifica una disyunción de objetivos.
  - a; b. → a ∨ b

Predicados de Control

- **findall(+Template, :Goal, -Bag)**: Crea una lista en donde Bag contendrá una unificación de Template por cada posible unificación de Goal que será evaluado contra el cuerpo de conocimiento.
  - f(1).
  - f(2).
  - f(3).
  - findall(g(X), f(X), Y). → Y = [g(1), g(2), g(3)]
- **forall(:Cond, :Action)**: Se evalúa verdadero si para todas las unificaciones de Cond, se puede probar Action.
  - forall(f(X), write(X)). → 123

Predicados de Control

- **apply(+Term, +List)**: Agrega los términos en la lista a los argumentos de Term

  - plus(1, 2, X). → X = 3
  - apply(plus, [1, 2, X]). → X = 3

Construcción y Acceso a Componentes de Estructuras

- Permiten construir y manipular componentes compuestos:
  - **functor(?T, ?F, ?A)**: Se satisface si T es un término con functor F y aridad A.
    - X = punto(1, 2), functor(X, punto, 2). → true
    - X = punto(1, 2), functor(X, punto, 3). → false
  - **arg(?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(2, punto(5, 3), X). → X = 3.

Construcción y Acceso a Componentes de Estructuras

- **setarg(+A, +T, +V)**: los argumentos representan lo mismo que en arg/3, pero ahora se asigna el valor V al argumento A.
  - X = punto(5, 3), setarg(2, X, 5).

    → X = punto(5, 5).

- **=../2**: se utiliza para construir una estructura dada una lista de argumentos, el primer argumento representa el functor y el resto los argumentos del mismo.
  - X =..[punto, 5, 3].

    → punto(5, 3)

Manipulación de Base de Datos

- Prolog ofrece mecanismos para manipular la base de conocimiento en forma dinámica
  - **:-dynamic(X)**: se utiliza para especificar que un predicado en particular se manipulara dinámicamente. X se expresa en la forma "predicado" o "predicado/aridad".
    - :-dynamic(f/1).
  - **asserta(+Term) / assertz(+Term)**: Añade cláusulas al inicio/final del conjunto de cláusulas en la BD que tienen el mismo nombre de predicado.
    - asserta(f(1)).
    - assertz(f(1)).

Manipulación de Base de Datos

- **retract(+Term)**: Elimina de la base de conocimientos las cláusulas que unifican con la expresión ingresada como argumento, dejando tantos puntos de elección como unificaciones se encuentren.
  - retract(f(2)).

- **retractall(+Head)**: Elimina de la base de conocimientos las cláusulas que unifican con el átomo ingresado como argumento sin dejar puntos de elección en el proceso.
  - retractall(f(X)).

Manipulación de Base de Datos

- **listing**: lista todas las cláusulas definidas en la base de conocimiento

- **listing(+Pred)**: lista todas las cláusulas definidas en la base de conocimiento que unifican con el predicado que se pasa como parámetro. El parámetro pasado puede ser de la forma "predicado" o "predicado/aridad"
  - listing(f(X)).
  - listing(f/1).