Clase 1.1: Introducción a la Programación Declarativa y Programación Lógica

Profesora: Dra. Yisel Garí

En esta unidad se estudiará el modelo de la programación declarativa a través de uno de sus más importantes realizaciones la programación lógica (PL) y la realización estándar de esta en el lenguaje Prolog.

Características de los lenguajes convencionales

Antes de abordar el asunto que nos concierne analicemos un panorama de la programación convencional que hasta ahora el lector conoce, para detectar algunas deficiencias de la misma.

La programación convencional fue la primera en desarrollarse respondiendo directamente a la arquitectura establecida por von Neumann de máquinas computadoras que cargan, modifican y dejan los resultados de un proceso computacional en registros de memoria. De hecho los recursos expresivos proporcionados por los lenguajes convencionales pueden verse como abstracciones de los componentes u operaciones elementales que la máquina de von Neumann incorpora. Esta modalidad de programación recibe la calificación de imperativa, proveniente del modo verbal imperativo con el cual los seres humanos nos comunicamos en ciertas ocasiones para expresar una orden o comando, tal y como ocurre en las instrucciones de estos lenguajes. El primer ejemplo significativo de lenguaje de programación imperativa fue FORTRAN.

La instrucción más relevante dentro de los lenguajes imperativos es la asignación. Dicha operación genera polémicas matemáticas, porque en el ámbito matemático, un símbolo de variable siempre denota el mismo objeto, cualquiera que sea la ocurrencia de ese símbolo, sin embargo la asignación utiliza las variables de modo matemáticamente impuro. Por ejemplo, en C, a = 2+a (asignar a la variable a el valor de la expresión aritmética 2+a), donde la ocurrencia de la variable en la parte derecha de la asignación no denota lo mismo que la ocurrencia de la parte izquierda (en general, la parte izquierda de una instrucción de asignación debe entenderse como una referencia a una dirección de memoria, mientras que la parte derecha como una expresión que se evaluará para almacenar el resultado en la palabra cuya dirección es la variable de la izquierda).

En la tarea de programación podemos distinguir dos aspectos fundamentales:

- Aspectos lógicos: Esto es, ¿Qué debe computarse? Esta es la cuestión principal y la que motiva el uso de un ordenador como medio para resolver un determinado problema.
- Aspectos de control, entre los que podemos distinguir:
  - Organización de la secuencia de cómputos en pequeños pasos.
  - Gestión de la memoria durante la computación.

Dado que dichos aspectos lógicos y de control se refieren a componentes claramente distintos e independientes de las tareas de programación (especificación del problema e implementación del programa que lo resuelve, respectivamente), parece deseable que los lenguajes de programación nos permitan mantener las distancias entre ambos, sin necesidad de que las cuestiones implicadas en las tareas de especificación e implementación interfieran entre sí. Concretemos algunos de

los defectos de un lenguaje de programación imperativo considerando el siguiente programa en C# que especifica cómo ha de llevarse a cabo el proceso de concatenar dos listas de enteros:

```
public int[] concatena(int[] l1, int[] l2)
{
   int[] l3 = new int[l1.Length + l2.Length];
   for (int i = 0; i < l1.Length; i++)
       l3[i] = l1[i];
   int sigue = l1.Length;
   for (int i = 0; i < l2.Length; i++)
       l3[sigue + i] = l2[i];
   return l3;
}
```

Podemos resaltar que:

1. La presencia de instrucciones de control de flujo (if, for, while,etc.) y la gestión de memoria (distinción entre declaración y definición de una estructura o variable del programa) oscurecen el contenido lógico del programa.

2. La operación de asignación cambia el estado de las variables por lo que obliga a conocer el contexto de la variable en cada momento.

3. Las secuencias de instrucciones que constituyen el programa son órdenes a la máquina.

4. Para entender el programa debemos ejecutarlo mentalmente estudiando cómo cambian los contenidos de las variables y otras estructuras en la memoria.

5. El programa tiene falta de generalidad debido a la rigidez del lenguaje a la hora de definir nuevos tipos de datos.

6. La lógica y el control están mezclados, lo que dificulta la verificación formal del programa.

Vistas estas críticas podemos decir que en esencia un programa imperativo especifica cómo ha de llevarse a cabo un proceso de computación y responde en general a la pregunta ¿cómo? (how?). Nótese que el programa imperativo anterior es un procedimiento para hallar la concatenación de dos listas, es decir, el programa anterior responde a la pregunta ¿cómo hallar la concatenación de dos listas? Se denomina procedimental a la programación que genera programas que responden esencialmente a la pregunta ¿Cómo? El lector analizará la frecuencia en el uso de proposiciones imperativas cuando sigue mentalmente las instrucciones para realizar una tarea o bien cuando comunica a otros cómo realizarla. Hemos puesto de manifiesto algunos puntos débiles de los lenguajes convencionales, pero el lector debe ser consciente de que ellos reúnen otras ventajas que los han hecho preferidos entre los programadores. Entre estas ventajas podríamos citar: eficiencia en la ejecución, modularidad, herramientas para la compilación separadas, herramientas para la depuración de errores.

Programación declarativa

Primeramente intentemos responder la pregunta ¿Qué es la declaratividad? Si fijamos la atención en los usos que hacemos del lenguaje con diferentes fines y en diferentes situaciones comprobaremos que en cada caso se emiten proposiciones que expresan significados disímiles. Tradicionalmente estas proposiciones emitidas de forma diferente se han denominado en la lógica

como modalidades. La declarativiad es una modalidad lógica del intercambio de información entre los seres humanos, es decir, una modalidad de ciertas proposiciones de un lenguaje con las cuales intercambiamos y compartimos conocimiento. Por ejemplo, en la lengua española las siguientes proposiciones son declarativas:

- Julio es abuelo de Ana.
- El agua hierve a 100 grados centígrados de temperatura a nivel del mar.
- Un paciente tiene presión alta si su sistólica es mayor que 140 y su diastólica es mayor que 90. (Caracterización de presión alta).

Hay en ocasiones un uso procedimental de estas últimas (si se quiere comprobar si un paciente tiene presión alta, determine si su sistólica es mayor que 140 y si su diastólica es mayor que 90). Con las proposiciones declarativas en el lenguaje natural intercambiamos conocimiento afirmando(negando) hechos, enunciando principios, leyes, reglas, definiendo conceptos, describiendo objetos, relaciones entre objetos, situaciones, sucesos, etc. Cuando afirmamos (negamos), enunciamos, definimos o describimos con proposiciones de un lenguaje, utilizamos proposiciones declarativa.
Una característica de las proposiciones declarativas es que a las mismas puede asociarse una función lógica de verdad. La más frecuente de estas funciones son las denominadas funciones booleanas en una lógica bivalente estándar que asigna a toda proposición el valor verdadero o falso como se estudió en el curso de Lógica y como es visible en la utilización de las proposiciones booleanas en programación no-declarativa.
Ya analizados los aspectos esenciales de la declarativiad podemos adentrarnos en su utilización dentro de las tareas de programación.
La programación declarativa puede entenderse como un estilo de programación en el que el programador especifica qué (what?)debe computarse más bien que cómo deben realizarse los cómputos, y a la pregunta ¿qué? se responde declarativamente con una definición o con el enunciado o descripción de un hecho o suceso. En este paradigma de programación, de acuerdo con el aserto inicial que especifica que programa = lógica + control, la tarea de programación consiste en centrar la atención en la lógica dejando el control, que se asume automático, al sistema. El componente lógico determina el significado del programa mientras que el componente de control solamente afecta su eficiencia. Con más precisión, la característica fundamental de la programación declarativa es el uso de la lógica como lenguaje de programación.
Casi simultaneando con FORTRAN en el tiempo aparece la programación declarativa con el lenguaje LISP, un lenguaje basado en el concepto de función y de definición recursiva de funciones. El concepto de recursividad fue introducido por la programación declarativa. La recursión es un poderoso instrumento de definición y de ahí su carácter declarativo y el hecho de su esencialidad y utilización sistemática en los lenguajes de programación declarativos para definir estructuras de datos y procesos. Posteriormente fue adoptado en lenguajes imperativos como ALGOL. El siguiente ejemplo de la definición recursiva de la función de potencia en ALGOL muestra su proceder declarativo:
potencia(m, n) = if n = 1 then m else potencia(m, n-1) * m
Note que lo que se denomina “declarativo” en un lenguaje imperativo, tales como la declaración de variables y sus tipos, los modos de los parámetros (de entrada, de salida) en los procedimientos es sólo información estática para el uso del compilador, es decir, no forma parte del algoritmo para resolver una clase de problemas que representa el programa. Sin embargo, las proposiciones booleanas utilizadas en las proposiciones condicionales y en algunas estructuras de control en los lenguajes imperativos son esencialmente declarativas al igual que la adopción de la recursividad.

Esto demuestra que la distinción entre declarativo y procedimental no es absoluta: de la misma manera que los lenguajes procedimentales no pueden prescindir hasta cierto punto de la declaratividad, los lenguajes declarativos no pueden prescindir de la imperatividad como se podrá ver en el desarrollo de este curso, al menos en las acciones con las cuales debe receptar valores de entradas, imprimir valores de salida y sobre todo buscando eficiencia en la ejecución de un algoritmo para resolver un problema.
Analicemos el siguiente problema: determinar si una persona es abuelo de otra persona. Para resolver este problema los seres humanos parten de la definición de abuelo, es decir, de una proposición declarativa, que podría ser enunciada utilizando variables de la siguiente forma:
X es abuelo de Y si X es padre de Z y Z es padre o madre de Y.
La anterior proposición define qué es ser abuelo. Resulta importante señalar cómo a partir de esa definición se puede extraer un procedimiento (se la puede interpretar procedimentalmente) para determinar si dados valores para X, Y, X es abuelo de Y. Dejamos al lector la extracción de tal procedimiento (no es único).
La programación declarativa aspira a especificar problemas de una manera más cercana a la forma en que los especifican los seres humanos. La especificación de problemas es eminentemente declarativa tanto en lo que concierne a su definición como al conocimiento que se posee para resolverlo y a los datos de la instancia particular del problema que se desea resolver. Por otra parte los operadores aplicables al conocimiento y a los datos frecuentemente son no-numéricos, por ejemplo, cuando se transforman expresiones simbólicas como en problemas algebraicos o cuando se razona deductivamente como en el razonamiento diario de los seres humanaos o en la demostración de teoremas en matemática.
Establezcamos una comparación entre los lenguajes declarativos (LD) y los imperativos (LI).

1. Diseño del lenguaje y escritura de programas:
    - Sintaxis sencilla: los LD cumplen con esta característica, mientras que la sintaxis de los LI suele ser muy compleja.
    - Modularidad y compilación separada: Esta característica facilita la estructuración de los programas, la división del trabajo y la depuración durante la fase de desarrollo. La mayoría de los LI proporcionan estas facilidades, sin embargo los LD no suelen tener herramientas para la compilación separada, principalmente porque muchos de ellos nacieron como lenguajes interpretados.
    - Mecanismos de reutilización del software: esto es para eliminar la repetición de trabajo. En este aspecto, al estar muy ligado al punto anterior, los LI ofrecen mejores condiciones de reutilización del software. Por su parte los LD suelen realizar esta función mediante la inclusión de "preámbulos de función", un mecanismo más ineficiente y rudimentario.
    - Facilidades de soporte al proceso de análisis: los LD son fuertes en este rublo pues pueden considerarse como lenguajes de especificación, útiles en la fase de análisis de una aplicación. Sin embargo el relativo bajo nivel de los LI hace que dependan más de herramientas externas de ayuda al análisis y diseño como UML.
    - Entornos de programación: puede parecer secundario, pero este es un factor muy importante porque un buen entorno de programación puede hacer que sea más fácil trabajar. Aquí los LD encuentran una debilidad frente a los LI, ligada principalmente a que hasta el momento son más utilizados en ambientes universitarios e investigativos. Esta situación cambiará en la medida que las técnicas de la programación declarativa sean más apreciadas en sectores industriales.
2. Verificación de programas:

- Verificación de la corrección: consiste en que dado un programa P y su especificación o significado S ¿computan P y S la misma función?. Los LD por poseer definiciones sencillas y estar basados en sólidos fundamentos matemáticos, pueden responder con relativa facilidad a esta pregunta. No siendo el caso de los LI.
- Terminación: los LD están más adaptados a realizar este tipo de pruebas motivo por el cual existe una amplia literatura sobre terminación de programas lógicos.
- Depuración de programas: los LD poseen una buena legibilidad, pero no ofrecen buenas herramientas de depuración. Mas los LI poseen potentes herramientas de depuración, aunque su legibilidad suele ser peor.

3. Mantenimiento: cuando un programa ya está en uso y hay que repararlo o mejorarlo. Esto incluye la facilidad de uso y lectura y la modularidad y compilación separada, aspectos que fueron analizados en el punto 1.

4. Coste y eficiencia:

   - Coste de ejecución: Es uno de los puntos débiles de los LD pues suelen ser menos eficientes que los lenguajes convencionales porque resulta difícil implementar en máquinas convencionales las operaciones de las que dependen estos lenguajes así como los mecanismos de búsqueda de soluciones. Por otro lado los LI son eficientes pues su diseño es reflejo del modelo de computación de von Neumann en el que se basa la arquitectura da las computadoras actuales.
   - Coste de desarrollo: comprende los costes asociados a las fases de análisis, programación y verificación de un programa. Las estadísticas indican que, para cierto tipo de aplicaciones, los costos de desarrollo son muy bajos cuando se utilizan LD y, por el contrario, los costes de utilizar lenguajes convencionales suelen ser muy altos. Generalmente el número de líneas de un programa declarativo es una fracción del número de líneas de uno imperativo. Por ejemplo: la relación entre el número de líneas escritas en Prolog y el de escritas en C o Pascal es 1/10 en aplicaciones que desarrollan sistemas expertos y de 1/8 para aplicaciones que desarrollan compiladores.

Esta comparación revela que un lenguaje de programación no es bueno para todas las tareas. Por consiguiente cada lenguaje tiene su dominio y aplicación. Particularmente podemos enumerar algunos campos en los que la programación declarativa es utilizada en la actualidad: Procesamiento del lenguaje natural, representación del conocimiento, química y biología molecular, desarrollo de Sistemas de Producción y Sistemas Expertos, resolución de problemas, metaprogramación, prototipado de aplicaciones, bases de datos deductivas, servidores y buscadores de información inteligentes, diseño de herramientas de soporte al desarrollo del software.
La programación declarativa incluye como paradigmas más representativos la programación lógica y la funcional. La programación lógica se basa en fragmentos de la lógica de predicados, siendo el más popular la lógica de cláusulas definidas, mientras que la programación funcional se basa en el concepto de función (matemática) y su definición mediante ecuaciones (generalmente recursivas), que constituyen el programa, centrándose, desde el punto de vista computacional, en la evaluación de expresiones (funciones) para obtener un valor. Ambas tienen diferentes realizaciones en lenguajes como Prolog y Haskell respectivamente. En este curso veremos más aspectos de las mismas comenzando con la programación lógica.

Primeros pasos para la programación lógica

Durante esta unidad el lenguaje de programación lógica que se empleará será Prolog por lo cual es necesario ver los principales aspectos del mismo.

Notación

- las letras mayúsculas representan variables
- el símbolo :- representa la operación lógica $\Rightarrow$ utilizada en su notación $< consecuente > \Leftarrow < antecedente >$ (lectura: "$< consecuente >$ *si* $< antecedente >$")
- la coma representa la operación lógica de la conjunción ($\wedge$)
- el punto y coma representa la operación lógica de disyunción ($\vee$)
- el punto representa el final de la fórmula.

Listas

Las listas constituyen una estructura de datos de amplio uso en la programación y en particular en la programación declarativa. Desde un punto de vista lógico una lista es un tipo particular de término construido mediante el símbolo constructor binario *"."* y el símbolo de constante *nil* que denota la lista vacía, de acuerdo con las siguientes especificaciones:
i) *nil* es una lista.
ii) Si s es un término cualquiera y t es una lista, entonces *.(s, t)* es una lista.
En la lista *.(s, t)*, s se denomina el primer elemento de la lista y t el resto de la lista. Obsérvese que el resto de una lista es también una lista. Los siguientes son ejemplos de listas:

- *nil*
- *.(2, nil)*
- *.(2, .(5, nil))*
- *.(.(1, nil), .(1, .(3, nil)))*

Introduzcamos desde ahora la notación externa de Prolog que facilita la manipulación de listas en un programa mediante ejemplos:

- `[]` denota la lista vacía.
- `[1, 2, 3]` denota la lista .(1, .(2, .(3, nil))) que tiene como únicos elementos los números 1, 2 y 3.
- `[X |Y]` denota una lista con un primer elemento X y un resto Y. Se trata de un esquema o patrón de lista que representa cualquier lista que tiene al menos un elemento.
- `[X, Y |Z]` es el esquema de lista que representa cualquier lista con al menos dos elementos.

Un programa en Prolog es un conjunto de fórmulas lógicas de cierta estructura que se definirá más adelante. A continuación analizaremos un primer programa Prolog en el cual se da una definición recursiva de la concatenación de dos listas. El programa concatena hace uso de la recursividad tanto en la manipulación de las listas que son definidas recursivamente, como en la definición de la operación que aplica (concatenar): la concatenación de una lista con otra se define recursivamente colocando primero los elementos de una de las listas y luego los de la otra.

```
concatena([], X, X).
% La concatenación de la lista vacía  []  y otra lista  X es la propia lista  X
concatena([X|Rx], Y, [X|Z]) :- concatena(Rx, Y, Z).
% La concatenación de dos listas  [X|Rx]  e  Y  es la lista que resulta de añadir
% el primer elemento  X  de la lista  [X|Rx]  a la lista  Z  que se obtiene al
% concatenar el resto  Rx  de la primera lista a la segunda lista  Y .
```

El programa anterior responde a la pregunta *qué* es concatenar dos listas, es decir, da esencialmente una *definición* de concatenar como se puede apreciar a través de los enunciados que constituyen una lectura declarativa de las fórmulas del programa.
Luego, si en la programación declarativa las tareas lógicas de un programa están bien planteadas, ¿Quién se encarga entonces del control o ejecución de un programa declarativo? Como se estudiará en esta unidad con respecto al lenguaje Prolog, un *intérprete* de los programas convenientemente construido hace una *lectura procedimental de un programa declarativo y lo ejecuta*. En especial el intérprete introduce el *control* necesario para la ejecución del programa.

Resumen a modo de conclusión

En la programación procedimental los procesos no se definen, más bien los programas describen cómo realizarlos. Un programa procedimental responde a la pregunta cómo, ofreciendo más que una definición un procedimiento. Se denominará procedimental a la programación cuyos programas son efectivamente procedimientos.
En la programación declarativa se especifican los problemas que se quieren resolver, mediante el uso de definiciones, descripciones y en general proposiciones declarativas. De esta manera los programas declarativos responden a la pregunta qué.
Ambas formas de programación, declarativa y procedimental son necesarias en la programación actual, no se contraponen y hasta pueden ocurrir en un mismo programa. El uso de una u otra forma de programación depende de lo que computacionalmente se intente resolver.
Algo muy importante con relación a la programación declarativa que se verificará a través de su estudio es que:
La programación declarativa permite al programador concentrarse fundamentalmente en la representación y definición de los datos y procesos relacionados con la solución de un problema haciendo abstracción en un primer momento de detalles de implementación y en especial del control en la interpretación y ejecución de su programa.