

White paper: Paradigma lógico

Autor: Ramiro A. Gómez

Sitio web: www.peiper.com.ar



Introducción

Cuando hablamos de programación nos vienen a la mente casi de manera automática los algoritmos. Y los pensamos como una serie de pasos que se ejecutan secuencialmente paso por paso. En sí, eso es un algoritmo.

Pero hacer un algoritmo para un lenguaje estructurado, ya sea imperativo u orientado a objetos, es un proceso sumamente artesanal en el que nos debemos valer de nuestra inteligencia y razonamiento para comprobar que sus pasos producen los resultados que nosotros deseamos.

En este proceso artesanal son muy comunes los errores. Tanto lógicos, léxicos, o sintácticos (errores en el uso de las estructuras sintácticas del lenguaje de programación).

La probabilidad de tener errores lógicos aumenta al manipular estructuras de datos complejas, como árboles, grafos, etc. Y es ahí donde nos podemos valer de la estrategia llamada “divide y vencerás”. Esta se centra principalmente en dividir un problema grande y complejo en subproblemas pequeños y simples, de modo que la combinación de estos últimos sea equivalente a la primera.

Sin embargo, muy próximo a la invención de las computadoras, surgió una idea totalmente distinta acerca de la manera de resolver problemas. La idea consistía en no detallar los pasos exactos que se debían seguir para obtener una solución. En vez de ello, sólo se especifican como deben ser las características de la solución y el lenguaje se encarga de obtenerla utilizando distintos métodos.

Por ejemplo, en un lenguaje lógico para ordenar un arreglo podríamos escribir algo como:

```
ordenar(arreglo) {  
    a < b;  
    todos(arreglo[a]) < todos(arreglo[b])  
}
```

Como ven, el código es muy elegante. No es necesario poner complicados contadores que nos alejan de la lógica del problema. El lenguaje lógico se encargará de darnos una solución que cumpla con las 2 condiciones indicadas en el cuerpo de la función ordenar.

Capacidad de los lenguajes lógicos

Un lenguaje lógico, como puede ser Prolog, permite constestar preguntas con respuestas "Sí", "No", o con elementos que satisfagan condiciones.

Para que el lenguaje nos pueda responder debe contar con una serie de hechos y relaciones.

Por ejemplo, la sentencia siguiente se puede reescribir en Prolog:

Pablo aprobará el exámen si estudia.

Si Pablo aprueba el exámen, los padres le regalan una Playstation.

Pablo estudia.

aprueba(pablo) :- estudia(pablo).

regalan_playstation(pablo) :- aprueba(pablo).

estudia(pablo).

Usando el sentido común, vemos que Pablo va a aprobar el exámen si estudia, y si aprueba le regalan una Playstation. También sabemos que Pablo estudia (este es un hecho). Por las reglas que definimos, como Pablo estudia aprueba el exámen, y como aprueba le regalan una Playstation.

En el lenguaje lógico podemos consultar:

?- regalan_playstation(pablo)

La respuesta del lenguaje será: "Sí".

Podemos agregar variables (que comienzan con mayúscula):

?- aprueba(X)

La respuesta será: "Pablo". Porque Pablo es el que se sabe que aprueba.

Lo que ahora sería bueno que nos preguntemos es sobre los mecanismos que utiliza un lenguaje lógico para determinar la veracidad de una sentencia o los elementos que satisfacen nuestras condiciones.

Proceso de inferencia

Un lenguaje lógico usa dos tipos de elementos para inferir. Estos son hechos y reglas de inferencia, los dos son definidos en el programa lógico por el usuario.

Una regla es de la forma $A \rightarrow B$. Esta nos dice que si A es verdadera, B debe serlo también.

A su vez, si tenemos $B \rightarrow C$ vamos a poder inferir $A \rightarrow C$. Esta regla lógica de inferencia se conoce como “silogismo hipotético”. Se puede leer: A implica a B, y B implica a C, entonces por propiedad transitiva A implica C.



Otra regla de inferencia es la llamada “modus ponens”. Es de la forma $A, A \rightarrow B$. Entonces B. Significa que si tenemos un hecho A y A implica B, entonces también ocurre B.

Un lenguaje lógico se basa en los procesos llamados backtracking y negación. Todo comienza a partir de una pregunta ingresada por el usuario. Primero niega la pregunta y si encuentra en algún momento que es falsa, entonces la pregunta sin negar es verdadera.

Por ejemplo, si las sentencias son de la forma $A_1, A_2, \dots, A_n \rightarrow B$, busca todas las B que contienen la pregunta y las reemplaza por sus A_1, A_2, \dots, A_n . Luego reemplaza cada A_i al igual que hizo con B.

Hace esto de manera recursiva, y se va formando un árbol con todas las posibilidades.

Ejemplo:

$C1, C2, D, E1 \rightarrow B$
 $C1, C2 \rightarrow D$
 $E1 \rightarrow C2$
 $C1.$
 $E1.$
 $?- B$

En este programa lógico tenemos dos hechos y 3 reglas y la consulta por B. Encontramos una regla cuya implicación es B:

$C1, C2, D, E1 \rightarrow B.$

Luego reemplazamos D con C1, C2; y nos queda

$C1, C2, C1, C2, E1 \rightarrow B$

que se puede reducir a

$C1, C2, E1 \rightarrow B$

Reemplazamos C2 por E1

$C1, E1 \rightarrow B$

Como C1 y E1 son hechos, entonces B es verdadera.

En caso de llegar a un punto de reemplazo en que los predicados son todos verdaderos, retrocede pasos y prueba con otros reemplazos.

El programa termina de buscar cuando encuentra una sentencia que sea falsa o cuando recorrió todo el árbol encontrando que son todas las opciones verdaderas.

Todo este proceso puede tomar mucho tiempo, en especial si la consulta contiene variables, la base de conocimientos es muy extensa y si debe recorrer gran parte del árbol.

Elementos del paradigma lógico

Un lenguaje lógico tiene hechos, reglas de inferencia y variables. Los hechos y reglas de inferencia se componen en general de proposiciones (literales o letras que representan un suceso) o predicados (construcciones de la forma $f(L1, L2, \dots, Ln)$, donde f es una secuencia de letras). Los predicados tienen más poder expresivo que las proposiciones porque un mismo predicado puede operar sobre distintos elementos.

Los predicados no se combinan de cualquier manera, sino que lo hacen como cláusulas de Horn. Estas son un conjunto de predicados unidos por operadores lógicos OR, AND (o, y) que implican un solo predicado no negado. Hay maneras para convertir un conjunto de predicados en cláusulas de Horn.

Por el lado de los operadores lógicos, disponemos de

AND, OR, XOR, \rightarrow

Estos operadores son binarios, o sea que obtienen un resultado a partir de dos elementos.

- **AND:** Su símbolo es \wedge . Una sentencia que usa sólo este

símbolo es verdadera cuando TODOS los elementos son verdaderos. Si uno o más elementos son falsos el resultado final será falso.

- **OR:** Su símbolo es \vee . Una sentencia con este símbolo será verdadera si al menos UN elemento es verdadero. Será falsa si todos estos son falsos.
- **XOR:** No se usa en general. Su símbolo es $\underline{\vee}$. En $X \underline{\vee} Y$ sólo da un resultado verdadero si X es verdadero e Y falso, o X falso e Y verdadera. Si tienen el mismo valor de verdad el resultado es falso.
- **IMPLICACIÓN:** Su símbolo es \rightarrow . El resultado de $X \rightarrow Y$ será falso sólo si X es verdadero e Y es falso. En cualquier otro caso se obtiene el valor verdadero.

Lógica proposicional y de predicados

La lógica proposicional usa sólo literales. Cada uno de estos puede tener un solo valor de verdad. Al combinarlos con operadores lógicos se obtiene un único valor de verdad. Un ejemplo de sentencia proposicional es:

$$A \wedge B \vee C \rightarrow D.$$

(El símbolo " \wedge " significa "y", " \vee " significa "o"). Se lee: si A y B o C son verdaderas, entonces también lo es D.

La lógica de predicados agrega más potencia expresiva al componerse de construcciones parecidas a las funciones, que afectan a un conjunto de elementos. Un predicado puede tener distintos valores de verdad dependiendo de los elementos que tenga como parámetros. Ejemplos:

flaca(Maria): Equivale a María es flaca.

padre(Juan, Roberto): Equivale a Juan es el padre de Roberto

pertenece(a,b): Equivale a "a" pertenece a "b".

Implementación

Es posible implementar un lenguaje lógico usando un lenguaje orientado a objetos y hasta uno estructurado, como C, por decir alguno. Si analizamos detenidamente los elementos que componen al lenguaje lógico, muchas veces los conceptos son similares y tienen su proyección en otros paradigmas. Por ejemplo los hechos tienen su equivalente con los valores de las variables en los lenguajes imperativos o con las constantes.

Conclusión

El paradigma lógico es un enfoque muy abstracto y potente que

permite obtener inferencias a partir de hechos y reglas. A diferencia de los paradigmas más utilizados que requieren escribir todos los pasos que se deben realizar para llegar al resultado, en un lenguaje que utilice el paradigma lógico sólo debemos escribir las características de la solución. El lenguaje sólo, por medio de procesos de búsqueda en profundidad (recorrer en árbol) y backtracking, se encarga de obtenerla.

El paradigma lógico se usa en demostración automática de teoremas, reconocimiento del lenguaje humano, en bases de datos como Datalog, en enseñanza, en universidades y en ambientes de investigación.

Esperamos desde Peiper que les haya resultado interesante. Desde luego que hay mucho más detrás de este paradigma, pero elegimos presentar las ideas básicas omitiendo la mayor parte de la simbología, a fin de que a nuestros lectores les resulte lo más sencillo posible de comprender.

Autor: Ramiro A. Gómez (Ramix)



www.peiper.com.ar

Noticias y white papers de informática