1.1.1 Criterios de los lenguajes de programación

Descripciones bien definidas

Casi todos los lenguajes tienen descripciones bien definidas en manuales o algún documento de especificación del lenguaje que describe características, sintaxis, uso y codificación del lenguaje mediante ejemplos.

Forma Backus-Naur (BNF) y Forma Extendida Backus-Naur (EBNF)

La BNF es una forma de describir al lenguaje de manera mas concisa en su sintaxis de un lenguaje, definido como un metalenguaje que contiene meta símbolos y reglas propias las cuales se emplean para definir la sintaxis del lenguaje al que haga referencia.

También se puede decir que es una colección de instrucciones formadas para seguir un conjunto de reglas que diferencian a los programas validos de los que no lo son. El conjunto de símbolos de una BNF puede ser la siguiente

Meta símbolo	Significado
::-	Se define como
|	Alternativamente, o
<algo>	<algo> se reemplaza por su definición
algo	Una palabra escita en negritas se conoce como terminal o token que indica un elemento del lenguaje indivisible que no permite otros reemplazos

Un ejemplo de BNF se muestra a continuación

1. <programa> ::- <encabezado-programa> | <bloque-programa>
2. <encabezado-programa> ::- **program** <identificador>
3. <bloque-programa> ::- <bloque> **program**
4. <bloque> ::- <parte-definicion-constantes>
   <parte-definicion-tipos>
   <parte-declaracio-variables>
   <parte-declaracion-procedimientos-funciones>
   <parte-declaraciones>
5. <parte-declaraciones> ::- <declaración-compuesta>
6. <declaración-compuesta> ::- **begin** <secuencia-declaraciones> **end**

Los identificadores de una BNF

<identificador> ::- <letra> | <identificador><letra> | <identificador>

<letra> ::- a | b | c | d | e | f | g | h| i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<digito> ::- 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Los símbolos que son palabras (reservadas del lenguaje de programación)

<símbolo-palabra> ::- **program** | **const** | **type** | **procedure** | **function** | **var** |
begin | **end** | **div** | **mod** | **and** | **not** | **or** | **in** | **array** |
file | **record** | **set** | **case** | **of** | **for** | **to** | **downto** | **do** |
if | **then** | **else** | **repeat** | **until** | **while** | **with** | **nil**

Una EBNF es un superconjunto que contiene a una BNF y mas

Metasimbolo	Significado

[algo]	Ocurrencia de 0 a 1 de algo, opcional
{algo}	0 o mas ocurrencias de algo
(esto | eso)	Agrupación, ya sea esto o eso

Un identificador puede describirse como

<identificador> ::- <letra><letra> | <digito>

mas declaraciones

<secuencia-declaraciones> ::- declaración(declaracion)
<declaracion> ::- <declaración-simple> | <declaración-asignacion>
<declaración-simple> ::- <declaración-vacia> | <declaración-asignacion> |
<declaración-procedimiento>
<declaración-estructurada> ::- <declaración-compuesta> | <declaración-condicional> |
<declaración-repetitiva> | <declaración-with>
<declaración-condicional> ::- <declaración-if> | <declaración-case>
<declaración-if> ::- **if** <expresión-booleana> **then** declaración [<parte-else>] **end if**;
[<parte-else> ::- **else** declaracion

<expresión simple> ::- [signo] <termino> [<operador-suma><termino>]
<termino> ::- <factor> [<operador-multiplicacion><termino>]
<factor> ::- <identificador> | <constante-sin-signo> | <designador-funcion> |
<constructor-conjunto> | <expresiones> | **not** <factor>
<constante-sin-signo> ::- <numero-sin-signo> | <cadena-caracteres> | <identificador-constantes>
<numero-sin-signo> ::- <secuencia-digitos> | <real-sin-signo>
<secuencia-digitos> :: digito[digito]
<operador-suma> ::- + | - | **or**
<operador-multiplicacion> ::- * | / | **div** | **mod** | **and**

Semántica

Los lenguajes deben estar definidos semántica y sintácticamente, describiendo de manera precisa lo que significa una construcción en particular. Dos métodos se definen para realizar el trabajo, uno es axiomático (claro y evidente) y otro denotacional (formal y objetivo).

La semántica axiomática esta basada en el calculo de predicados, es decir, declarativo y basado en la lógica, donde las proposiciones son verdaderas o falsas evaluándose durante la ejecución del programa dentro de un ciclo iterativo hasta encontrar lo que se busca como comprobación o consulta.

La semántica denotacional esta basada en teoría de funciones que por lo regular son recursivas, en donde cada función recibe un conjunto de argumentos y devuelve un solo valor, siempre devuelve lo mismo bajo las mismas circunstancias.

Otro elemento de evaluación semántica consiste en emplear maquinas teóricas que dependen del lenguaje y de cómo funciona este, por lo regular pueden estar dentro del compilador.

Comprobabilidad

Debido a los costos elevados de creación de sistemas de software especializados o a medida, se hace indispensable comprobar que los programas y las matemáticas empleadas son correctas, además es un proceso lento que conlleva costes, ya que al encontrar errores deben coregirse. La comprobación de un programa o sistema por muy pequeño que sea involucra:

1. El programa cumple con la intención del programador

2. El compilador traduce correctamente a código de maquina la sintaxis y semántica del lenguaje correctamente.
3. Comprobar que el programa funciona correctamente en la maquina donde se desempeña.

Todo lo anterior incluye la perfecta traducción en la compilación de un BNF o EBNF, la sintaxis, semántica hasta llegar al código de la maquina donde se emplea el programa o sistema.

Confiabilidad

El usuario final de un programa o sistema espera que este haga lo que debe hacer con las entradas que recibe y devuelva cono resultado lo que el usuario desea todas las veces que lo utilice en lo que sea para lo cual fue creado, a esto se le conoce como confiabilidad.

Los lenguajes de programación han sido desarrollados para fomentar que la escritura de programas confiables, sin embargo, no todo es seguro, aun cuando nuestro programa no contenga errores de compilación, podría contener errores de lógica los cuales generan dolores de cabeza y son difíciles de encontrar, por ello es necesario tener una buena lógica de programación, seguir el procedimiento del usuario final al pie de la letra creando algoritmos que puedan ser comprobables.

Traducción rápida

Para este concepto debemos hablar del hecho de que muchos lenguajes fueron creados para ser independientes de la maquina, el problema es tener el (los) compilador (es) adecuados para el hardware en donde se desea ejecutar el programa, el código que se programa (código fuente) es el mismo, solo el traductor cambia. La maquina o hardware en donde se ejecuta el programa se le conoce como anfitrión, de la misma forma compiladores, lenguajes, etc.

Para crear el programa final que se ejecuta en un hardware cualquiera con un sistema operativo especifico terminación .exe para Windows, terminación .o para Linux o Unix, a veces requieren de traducir el código fuente a un lenguaje intermedio y después emplear otro programa para crear el ejecutable, por ejemplo para el lenguaje C el compilador crear código objeto (terminación .obj) y el linker (ligador) toma este y lo traduce a código nativo o de maquina diseñado para el anfitrión.

La traducción del código fuente involucra

- Análisis lexicográfico
  Rastreo e identificación de tokens (relacionados al lenguaje de programación) que representan valores, identificadores, operadores, ciclos, etc.
- Análisis sintáctico
  Reconoce declaraciones validas para aceptarlas o rechazarlas dentro del código fuente.
- Análisis semántico
  Determina el significado de una declaración.

En este punto de la traducción deben hacerse comprobaciones de memoria para que el compilador trabaje y para la disponibilidad de variables locale, globales, asi como el tamaño del código objeto, las características de depuración que se requiere para el código fuente desarrollado, detección y recuperación de errores para el archivo ejecutable.

Código objeto eficiente

El código intermedio denominado objeto también es compilado por el ligador para crear el programa ejecutable, pero para que el código intermedio sea eficiente el compilador debe ser eficiente en el trabajo de asignación de memoria, las localidades a emplear, tener cuidado con los tipos en las declaraciones, a veces el programador cuando realiza su trabajo no le pone las cosas fáciles al compilador, por ello algunos lenguajes emplean compiladores para optimizar el código generado ejecutando uno o dos pasos mas después del análisis semántico para incrementar la

eficiencia del código compilado.

Ortogonalidad

Proviene del griego y refiere a líneas rectas cruzándose en ángulos rectos (90 grados), esto es sinónimo de independencia y referente al lenguaje respecto a sus características todas son ortogonales y se comportan de igual forma bajo cualquier circunstancia, por ejemplo la declaración de un entero o un flotante siempre es la misma sintaxis, una función recibe argumentos y puede devolver un único valor hacia donde se le invoca. El hecho de no ser ortogonal puede ser molesta y producir errores.

Generalidad

Esta propiedad se relaciona con la ortogonalidad, lo que lleva al lenguaje a tener solo las características necesarias que conduzcan a tener efectos previsibles a la hora de programar, por ejemplo la estructura del lenguaje C en donde podemos tener un conjunto de características juntas en memoria siempre dará el mismo resultado.

Consistencia y notaciones comunes

Consistencia en las notaciones para declarar tipos (int, float, etc.), consistencia para realizar operaciones matemáticas como las aritméticas (+ para sumar enteros, flotantes y asi para los otros signos de operaciones), consistencia para declarar valores positivos o negativos (- negativo y + positivo).

Uniformidad

La consistencia esta relacionada con la uniformidad, nociones similares deben verse y comportarse de la misma manera, por ejemplo sumar dos números enteros y sumar dos números flotantes producen una suma.

Subconjuntos

Los subconjuntos van generando nuevas versiones de un lenguaje que va creciendo de acuerdo a la funcionalidad y propiedades que se le van añadiendo, por ejemplo el lenguaje A evoluciono en el lenguaje B y este en el lenguaje C y teste en el lenguaje C++.

Extensibilidad

La extensibilidad es lo contrario a los subconjuntos, el ejemplo básico para el lenguaje C es que para un computador donde los recursos son mayores el compilador y herramientas para programar con el es grande, pero si empleamos una tarjeta especifica donde el hardware es limitado solo empleamos una parte del lenguaje y se van creando librerías adecuadas al tamaño de los recursos de la tarjeta, por ejemplo el conjuntos de tarjetas de Arduino.

Transportabilidad

Un lenguaje es transportable si se puede compilar y ejecutar en diferente hardware o maquinas sin tener que reescribir el código fuente, esto tiene que ver mucho con los estándares en los lenguajes y que el programador se ajuste a ellos. ¿Cuál es la diferencia? Los compiladores son distintos y se programan de acuerdo al hardware