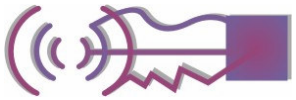
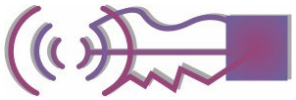


Semántica

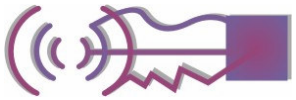
La *semántica* describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático

- **Tipos de semantica:**
  - Semántica estática
  - Semántica dinámica



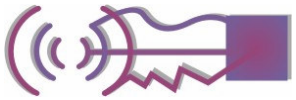
Semántica

- **Semántica estática**
  - está relacionado con las formas validas.
  - el análisis para el chequeo puede hacerse en compilación.
  - Para describir la sintaxis y la semántica estática formalmente sirven las denominadas gramáticas de atributos, inventadas por Knuth en 1968.
  - Generalmente las gramáticas sensibles al contexto resuelven los aspectos de la semántica estática.



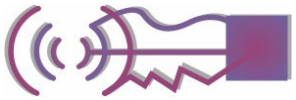
Semántica

- **Semántica dinámica**.
  - describe el efecto de ejecutar las diferentes construcciones en el lenguaje de programación.
  - Los programas solo se pueden ejecutar si son correctos para la sintáxis y para la semántica estática.



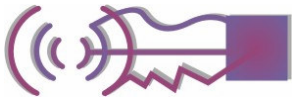
Semántica

- **¿Cómo se describe la semántica?**
  - No existen herramientas estandar como en el caso de la sintáxis (diagramas sintácticos y BNF)
  - soluciones formales:
    - Semántica axiomática
    - Semántica denotacional
    - Semántica operacional



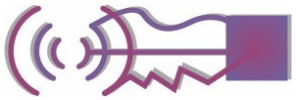
Semántica

- **Semántica axiomática**
  - La notación empleada es el “cálculo de predicados”.
  - Se desarrolló para probar la corrección de los programas.
  - Ve al programa como una máquina de estados.
  - Los costructores de un lenguajes de programación se formalizan describiendo como su ejecución provoca un cambio de estado.



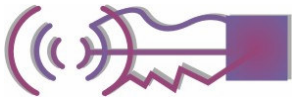
Semántica

- **Semántica axiomática**
  - Un estado se describe con un predicado que describe los valores de las variables en ese estado
  - Existe un **estado anterior** y un **estado posterior** a la ejecución del constructor.
  - Cada sentencia se precede y se continúa con una expresión lógica que describe las restricciones y relaciones entre los datos.
    - Precondición
    - Poscondición



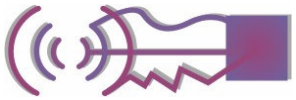
Semántica

- **Semántica denotacional**
  - Se basa en la teoría de funciones recursivas
  - Describe los estados a través de funciones.
  - Se define una correspondencia entre los constructores sintácticos y sus significados



Semántica

- **Semántica Operacional**
  - El significado de un programa se describe mediante otro lenguaje de bajo nivel implementado sobre una máquina abstracta
  - Los cambios que se producen en el estado de la máquina cuando se ejecuta una sentencia del lenguaje de programación definen su significado
  - Es un método informal
  - Es el más utilizado en los libros de texto



Semántica

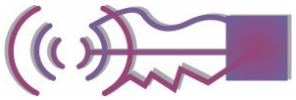
- **Semántica Operacional**

Lenguajes

for i := pri to ul do
begin
.....
end

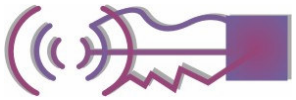
Máquina abstracta

i := pri
lazo if i > u goto sal
.....
i := i +1
goto lazo
sal .....



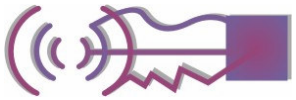
Procesamiento de un lenguaje Interpretación y traducción

- Las computadoras ejecutan lenguajes de bajo nivel
- cómo los programas escritos en lenguajes de alto nivel pueden ser ejecutados sobre una computadora cuyo lenguaje de muy bajo nivel?
- Alternativas:
  - **Interpretación**
  - **Traducción**

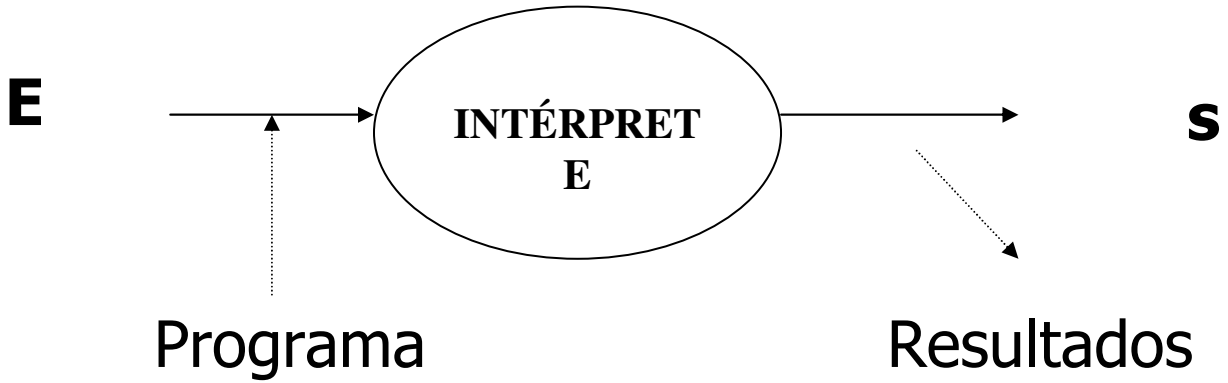


Interpretación

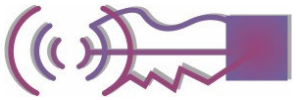
- “**Intérprete**” progama capaz de leer,analizar, decodificar y ejecutar una a una las sentencias de un programa escrito en un lenguaje de programación.
- Por cada posible acción hay un subprograma que ejecuta esa acción.
- La interpretación se realiza llamando a estos subprogramas en la secuencia adecuada.



Interpretación

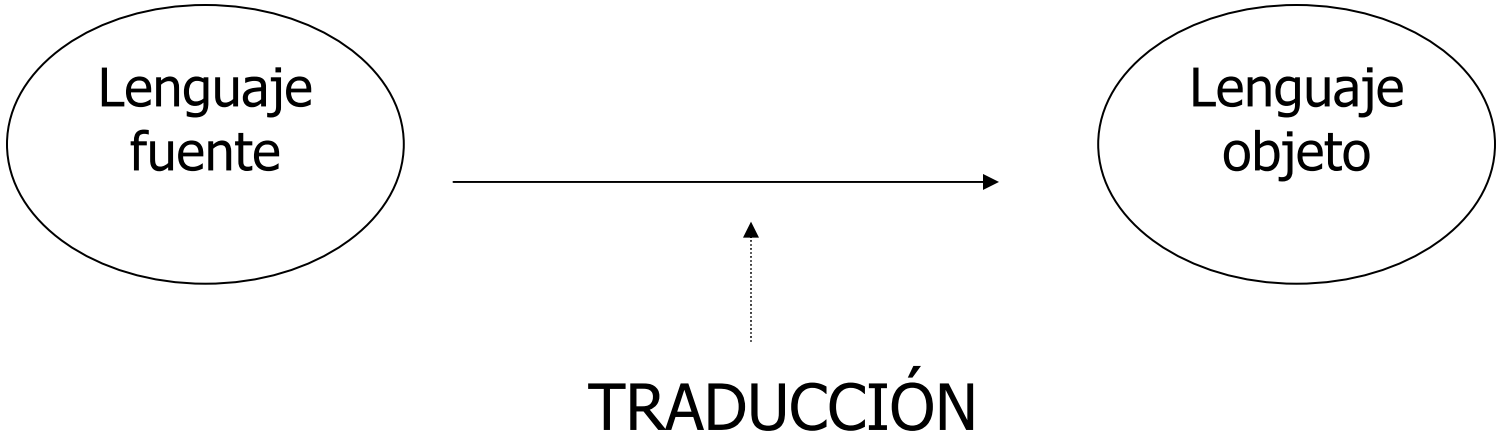


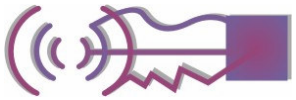
- Un intérprete ejecuta repetidamente la siguiente secuencia de acciones:
  - Obtiene la próxima sentencia
  - Determina la acción a ejecutar
  - Ejecuta la acción



Traducción

- Los programas escritos en un lenguaje de alto nivel se traducen a una versión en lenguaje de máquina antes de ser ejecutados.



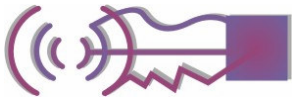


Traducción

- La traducción lleva varios pasos.

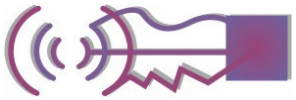
Ej: Pasos que se realizan para traducir un programa escrito en Fortran:

- Compilado a assembler ← **Compilador**
- Ensamblado a código reubicable ← **Assembler**
- Linkeditado ← **Link-editor**
- Cargado en la memoria ← **Loader**



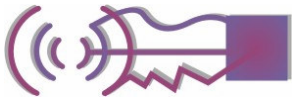
Traducción

- **Tipos de traductores:**
  - **Compilado**
  - **Assembler**
  - **Link-editor**
  - **Loader**



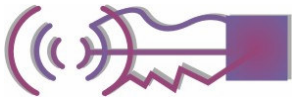
Traducción

- **Preprocesador o macro-procesador:**
- A veces antes del compilador se ejecuta otro traductor llamada **"Macro-Procesador o Procesador"**
  - **Macro:** fragmento de texto fuente que lleva un nombre



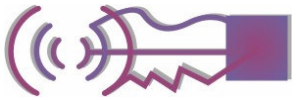
Traducción

- **Comparación entre Traductor e Intérprete:**
  - **Forma de ejecución**
  - **Orden ejecuión**
  - **Tiempo de ejecución**
  - **Eficiencia**
  - **Espacio ocupado**
  - **Deteccion de errores**



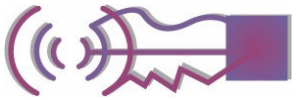
Traducción

- **Combinación de ambas técnicas:**
  - **Los compiladores y los interpretes se diferencian en la forma que ellos reportan los errores de ejecución.**
  - Algunos ambientes de programación contienen las dos versiones **interpretación y compilación**.
    - Utilizan el ***intérprete*** en la etapa de desarrollo, facilitando el diagnóstico de errores.
    - Luego que el programa ha sido validado se ***compila*** para generar código mas eficiente.



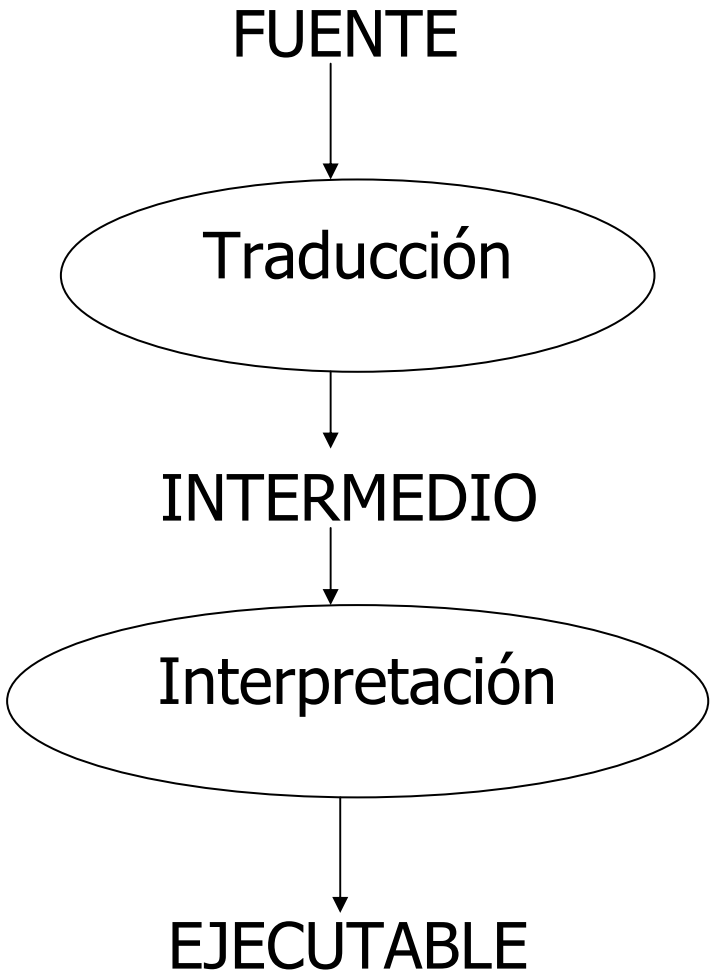
Traducción

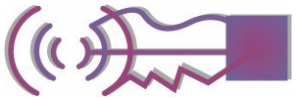
- **Combinación de ambas técnicas**
  - un programa puede traducirse en un **código intermedio** que luego se interpretará.
  - Diferentes soluciones:
    - **Más cerca de la interpretación**
    - **Más cerca de la traducción**



Traducción

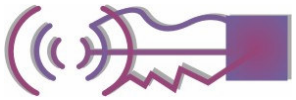
- **Combinación de ambas técnicas:**





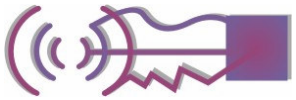
Compiladores

- ejecución más rápida.
  C, Ada, Cobol, etc.
- un paso o en dos pasos.
- Etapa
  - **Análisis**
    - **Análisi léxico (Scanner)**
    - **Análisis sintáctico (Parser)**
    - **Análisis semántico (Semnántica estática)**
  - **Síntesis**
    - **Optimización del código**
    - **Generación del código**



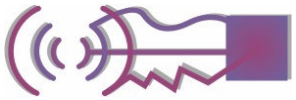
Compiladores

- **Análisis del programa fuente**
  - **Análisis léxico (Scanner):**
    - Hace el análisis a nivel de palabra
    - Divide el programa en: identificadores, delimitadores, símbolos especiales, números, palabras clave, delimitadores, comentarios, etc.
    - Analiza el tipo de cada token
    - Convierte a representación interna los números en punto fijo o punto flotante
    - Poner los identificadores en la tabla de símbolos
    - Reemplaza cada símbolo por su entrada en la tabla
    - items léxicos o tokens.



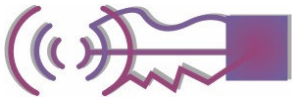
Compiladores

- **Análisis sintáctico (Parser):**
  - Se identifican las estructuras; sentencias, declaraciones, expresiones, etc. ayudándose con los tokens.
  - El analizador sintáctico se alterna con el análisis semántico. Usualmente se utilizan técnicas basadas en gramáticas formales.
  - Aplica una gramática para construir el árbol sintáctico del programa.



Compiladores

- **Análisis semántica (semántica estática):**
  - Es la mas importante
  - Las estructuras sintácticas reconocidas por el analizador sintáctico son procesadas y la estructura del código ejecutable toma forma.
  - Se realiza la comprobación de tipos
  - Se agrega la información implícita
  - Se agrega a la tabla de símbolos los descriptores de tipos, etc. a la vez que se hacen consultas para realizar comprobaciones.
  - Se hacen las comprobaciones de nombres. Es el nexo entre el análisis y la síntesis



Compiladores

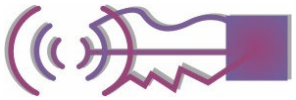
- **Generación de código intermedio:**
  - Debe ser fácil de producir
  - Debe ser fácil de traducir al programa objeto

Ejemplo: Un formato de código intermedio es el **código de tres direcciones**.
Forma: *A:= B op C*, donde *A,B,C* son operandos y *op* es un operador binario
Se permiten condicionales simples simples y saltos.

while (a >0) and (b<(a*4-5)) do a:=b*a-10;

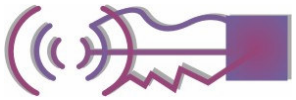
```
L1: if (a>0) goto L2
      goto L3
L2: t1:=a*4
      t2:=t1-5
      if (b < t2) goto L4
goto L3
```

```
L4: t1:=b*a
      t2:=t1-10
      a:=t2
      goto L1
L3: …….
```



Compiladores

- **Síntesis:**
  - se construye el programa ejecutable.
  - genera el código necesario y se optimiza el programa generado.
  - Si hay traducción separada de módulos, es en esta etapa cuando se linkedita.
  - Se realiza el proceso de optimización. Optativo



Compiladores

– Optimización (ejemplo):

Posibles optimizaciones locales:

•Cuando hay dos saltos seguidos se puede quedar uno solo

P/E El ejemplo anterior quedaría así:

```
L1: if (a<=0) goto L3          t1:=b*a
    t1:=a*4                    t2:=t1-10
    t2:=t1-5                   a:=t2
    if (b >= t2) goto L3       goto L1
                          L3: .....
```

•Eliminar expresiones comunes en favor de una sola expresión

a:=b+c+d Quedaría t1:=b+c b:=t1+e

b:=b+c+e a:=t1+d

Compiladores