UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Estructuras Discretas

Grupo: 06

Funciones universales e intérpretes

Nombre del profesor: ING. Orlando Zaldivar Zamorategui

Nombre de los integrantes:

- Rangel de la Rosa José Refugio
- Rodríguez Galindo Axel
- Villegas Condado Alexander

INDICE

1. INTRODUCCIÓN

En matemáticas, una función es una regla de correspondencia que relaciona los elementos de un primer conjunto con los de un segundo conjunto. El primer conjunto o el conjunto inicial es conocido como dominio, mientras que el segundo conjunto o conjunto final es conocido como rango. Una característica de las funciones es que a cada valor del dominio le corresponde un solo valor del rango.

1.1. Objetivo

El alumno comprenderá y aplicará la teoría de la computabilidad para determinar el estado computacional de funciones y problemas.

2. Máquina de Turing

La primera definición de Máquina de Turing se encuentra en el artículo del mismo Alan M. Turing del 1936 ***"On Computable Numbers, with an Application to the Entscheidungsproblem"***. En este artículo se introduce la idea de Máquina de Turing, la máquina universal, los números computables, y el problema de la parada. Finalmente, el material introducido se aplica al Entscheidungsproblem o Problema de la Decisión. Turing se apoya en la aritmetización de Gödel.

Intuitivamente, una Máquina de Turing es una cinta infinita (tiene un inicio, pero no un fin) de casillas iguales. Una cabeza se mueve sobre la cinta: en cada paso, se puede mover como mucho una casilla hacia la derecha o una hacia la izquierda. La cabeza puede leer, escribir o borrar símbolos en la casilla en la que se encuentra (no puede observar otras casillas ni actuar sobre ellas). La cabeza se encuentra en cada instante en un estado interno y, en cada paso, actúa según este mismo estado y el símbolo leído en la cinta. La acción elegida en cada paso consiste en escribir un símbolo nuevo en la cinta o en mover la cabeza, y se acompaña a un cambio del estado interno. Más formalmente, una Máquina de Turing (MT) es una quíntupla hK, Σ, s, H, δi donde:

• K es el conjunto finito de los estados internos de la cabeza;

• Σ es el conjunto finito de los símbolos que pueden aparecer en la cinta;

• s ∈ K es el estado inicial;

• H ⊆ K es el conjunto de los estados finales: normalmente incluye el estado h de terminación normal (es decir, el estado en el que la cabeza no tiene nada más que hacer y termina su trabajo) y el estado e de error (es decir, el estado en el que se encuentra la cabeza cuando algo ha ido mal);

• δ es la función de transición: su dominio es2 (K \ H) × Σ, es decir, el estado actual (que no puede ser un estado final) y el símbolo leído en la cinta; su codominio es K × (Σ ∪ {←, →}), es decir, δ devuelve el nuevo estado interno y una acción a ejecutar (escribir un símbolo en la cinta o mover la cabeza).

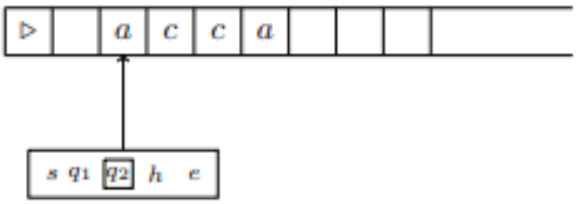

Figura 3.1: Una MT con su cinta (arriba) y su cabeza (con los estados internos)

El significado de δ es el siguiente: su dominio representa el input (la condición inicial) en cada paso: la cabeza se encuentra en un estado interno q ∈ K \ H (es decir, no final) y observa el símbolo σ ∈ Σ en la casilla donde se encuentra. El resultado (q ′ , x) = δ(q, σ) es la acción ejecutada, o paso: la cabeza pasa del estado q al estado q ′ y

• si x ∈ Σ entonces escribe el símbolo x en la cinta, y no se mueve;

• si x =← entonces da un paso a la izquierda;

• si x =→ entonces da un paso a la derecha.

El conjunto Σ incluye un símbolo ⊲ que marca el inicio de la cinta. Toda Máquina de Turing, al encontrar este símbolo, se mueve inmediatamente hacia la derecha sin cambiar de estado. Σ también incluye el símbolo “␣”, que indica que la casilla está vacía.

Composición de Máquinas de Turing Cuando hablamos de componer Máquinas de Turing no pensamos en distintas MTs, cada una con su cabeza, cinta y función de transición, que se comunican entre ellas a través de algún sofisticado protocolo. Hablamos de fusionar distintas funciones de transición, cada una dedicada a realizar una parte de la computación principal en una única MT con una única cabeza y una única cinta.

3. Tesis Church-Turing

"Nada se puede calcular o decidir que no se pueda calcular con una Máquina de Turing, o un programa while, o una función μ-Recursiva, u otro de los formalismos conocidos que tienen la misma expresividad".

Esta afirmación se llama Tesis de Church, y se puede presentar en distintas versiones equivalentes: por ejemplo, la Tesis de Church-Turing es la versión más conocida, y sólo se refiere a las MTs: Cualquier función que se pueda calcular es Turing-computable, es decir, existe una MT que la calcula. Queda claro que, en el caso de la Tesis de Church, lo que se mantiene insuperable no es una velocidad (la eficiencia de una Máquina de Turing se puede aumentar, y muchísimo, mejorando su hardware/software), sino la clase de funciones que se pueden calcular o lenguajes que se pueden decidir.

Hecho 6.1 Según la Tesis de Church, toda función efectivamente calculable puede ser calculada por una MT, una Función μ-Recursiva o un programa while

Hecho 6.2 Las MTs se consideran como el formalismo que mejor describe la idea de efectiva computabilidad.

4. Funciones Universales

Existe un tipo de dato conocido como el tipo W. En el caso de cadenas de caracteres, de valores lógicos y numéricos, de estructuras agregadas como pilas y árboles, la mayoría de estos problemas que se plantean son computables y estos ya están resueltos desde hace tiempo ("en algunos casos desde la civilización egipcia"), en estos mismos la incompatibilidad sucede muy pocas veces.

Las funciones universales es una forma abstracta de una máquina de turing universal. En donde esta máquina universal puede simular el comportamiento de otras máquinas de Turing. Gracias a la tesis de Church, sabemos que toda función computable tiene una representación en una máquina de Turing. La función universal simula el comportamiento de un conjunto de funciones ante una entrada dada. La entrada a esta función universal es la variable y un vector que es el conjunto de funciones a simular. Tiende a ser representada de la siguiente forma

$U\langle\phi,x\rangle = \phi i(x)$

Donde U es nuestra función universal, ϕ es el conjunto de funciones y x es la entrada. Una característica de la función universal es que el resultado de una simulación de una función en particular debe ser igual a la función original evaluada al mismo con el mismo valor.

4.1. Máquina de Turing Universal

Máquina de Turing Universal: ¿Cuál es la característica fundamental de la MT universal? Esta máquina, que llamamos U, recibe como input en la cinta la descripción "M" de una Máquina de Turing M y la descripción de un input w de esta misma M, y su trabajo es ejecutar M sobre el input. El resultado final de U será el mismo resultado M(w) que se obtendría ejecutando M sobre w, si esta computación termina. Si M calcula la función f, el resultado U("M'', w) es igual que f(w). Si M no termina para el input w, entonces U tampoco terminará su computación para "M" y w.

U se puede definir como una MT con 4 cintas.

La actividad de U consiste de tres fases: (1) fase U1: inicializar el contenido de todas las cintas; (2) fase U2: simular reiteradamente los pasos de M hasta llegar a uno de sus estados finales, si esto sucede en un tiempo finito; y (3) fase U3: si U2 termina, limpiar las cintas manteniendo sólo el output en la primera.

• La cinta C1 contiene inicialmente el input: la descripción de M y su input w. La fase U1 mueve w a la cinta C2, así que esta cinta sólo contiene la representación de M al empezar U2. Al terminar U2, si es que termina, C1 contendrá únicamente el output M(w).

• La cinta C2 contiene, a lo largo de toda la computación de U, una representación de la cinta de M.

• La cinta C3 contiene una representación del estado actual de M, que va cambiando según su función de transición δM almacenada en C1.

• La cinta C4 contiene un número que simula la actual posición de la cabeza de M en su cinta.

Teorema 8.1 (enumeración o universalidad) Existe una función calculable parcial de dos argumentos φz tal que, para todo x e i, φz(i, x) = φi(x)

5. Intérpretes

Un intérprete es un programa que se encarga de analizar y ejecutar un programa en lenguaje fuente de manera simultánea.

Los intérpretes cuentan con dos entradas, las cuales son:

- Un lenguaje(P) escrito en lenguaje fuente (LF), esto se denota como P/LF

Con estos datos de entrada y mediante un proceso de interpretación se producen resultados.

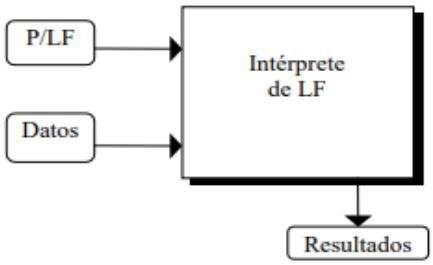

Figura 1: Esquema general de un intérprete

Labra Gayo, Jose Emilio. (2003). Esquema general de un intérprete. http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Por lo general, los intérpretes se especializan en, al menos, dos idiomas, su lengua materna y una extranjera de su elección.

5.1. Estructura de los intérpretes

La organización interna de la mayoría de los intérpretes se puede separar en los siguientes módulos:

- **Traductor a Representación Interna:** Toma como entra el código de un programa en lenguaje fuente, analiza y transforma el programa en su respectiva representación interna.
- **Representación Interna (P/RI):** Esta representación es consistente con el programa inicial. Dentro de estas representaciones, los árboles sintácticos suelen ser los más usados, además de utilizar estructuras de pila para una mayor eficiencia.
- **Tabla de símbolos:** Es se crea y utiliza en el proceso de traducción, en esta se van almacenando los símbolos dependiendo de la complejidad del lenguaje. Además de que se pueden almacenar etiquetas para instrucciones de salto, o de cualquier tipo que se necesite utilizar para la etapa de evaluación.
- **Evaluador de Representación Interna:** Se llevan a cabo las acciones convenientes para obtener buenos resultados. Dentro de éste proceso de evaluación se debe considerar la posible aparición de errores.
- **Tratamiento de errores:** En este proceso pueden aparecer errores que el intérprete debe considerar.

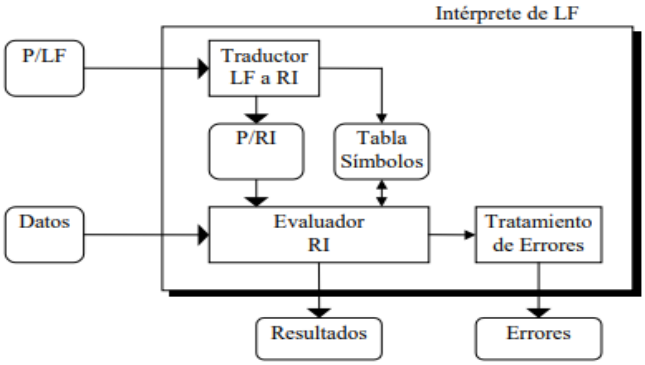

Figura 3: Organización interna de un intérprete

Labra Gayo, Jose Emilio. (2003). Organización interna de un intérprete. http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

La mayoría de los sistemas interpretados evitan que el programador maneje de manera explícita la memoria, mediante técnicas de recolección de basura.

5.2. Metodologías de Intérpretes

Dentro de la representación interna tenemos dos métodos fundamentales:

- **Interpretación Iterativa:** Se utiliza principalmente en lenguajes sencillos, en ellos se analiza y ejecuta cada expresión de manera directa, algunos ejemplos de esto son los códigos de máquinas abstractas o lenguajes de sentencia simple. Consiste en un ciclo básico de búsqueda, análisis y ejecución de instrucciones.

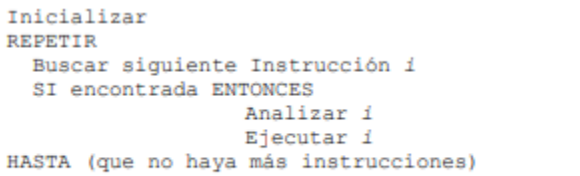

Figura 4: Interpretación iterativa

Labra Gayo, Jose Emilio. (2003). Interpretación iterativa. http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Las instrucciones se buscan en el almacenamiento, en ocasiones el usuario las ingresa. Enseguida se analizan sus componentes y se ejecutan. En la mayoría de los casos las instrucciones se ejecutan y descomponen en varios casos, una por cada instrucción.

- **Interpretación Recursiva:** Una primera fase de especificación semántica mediante la construcción de un intérprete prototipo que actúa como una especificación ejecutable y una segunda fase de implementación del compilador de dicho lenguaje.

Los intérpretes recursivos no son apropiados para aplicaciones prácticas debido a su ineficiencia y se utilizan únicamente como prototipo ejecutable del lenguaje.

El problema de especificar un lenguaje mediante un intérprete prototipo es decidir en qué lenguaje se implementa dicho intérprete.

5.3. Tipos de intérpretes

Intérpretes puros: Los intérpretes puros son los que analizan y ejecutan sentencia a sentencia todo el programa fuente. Los intérpretes puros se han venido utilizando desde la primera generación de ordenadores al permitir la ejecución de largos programas en ordenadores de memoria reducida, ya que sólo debían contener en memoria el intérprete y la sentencia a analizar y ejecutar en cada momento.

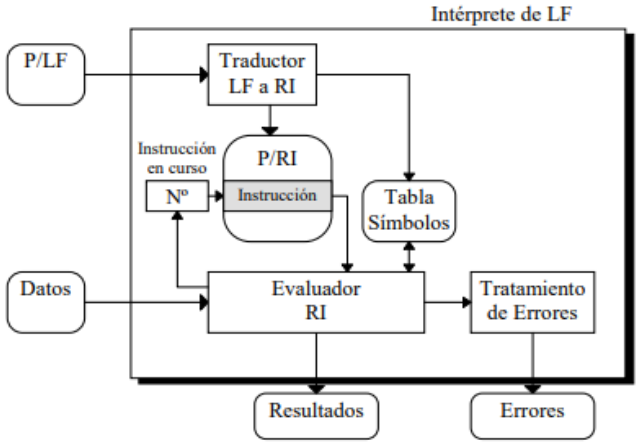

Figura 6: Esquema de un intérprete puro

Labra Gayo, Jose Emilio. (2003). Esquema de un intérprete puro. http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Una vez que este proceso ha finalizado, comienza la ejecución por la primera instrucción del código, que se envía al evaluador de instrucciones, éste la ejecuta. El evaluador de instrucciones también determina la instrucción siguiente a ejecutar, en algunos casos previa consulta a la tabla de etiquetas.

Intérpretes avanzados: En éste se incluye un paso previo al análisis. Posteriormente generan un lenguaje intermedio que ejecutan ellos mismos. De esta manera en casos de errores sintácticos, estos no pasan de la fase de análisis.

- **Intérpretes incrementales:** Existen ciertos lenguajes que, por sus características, no se pueden compilar directamente. La idea es compilar aquellas partes estáticas del programa en lenguaje fuente, marcando como dinámicas las que no puedan compilarse. Posteriormente, en tiempo de ejecución, el sistema podrá compilar algunas partes dinámicas o recompilar partes dinámicas que hayan sido modificadas.
- **Evaluadores parciales:** El primer conjunto, se conoce como datos de entrada dinámicos (Din), mientras el segundo, serían los datos de entrada estáticos (Est). Dado un programa P, el proceso de evaluación parcial radica en construir otro programa especializado P_{Est} para los datos estáticos de P. El programa PEst suele estar escrito en el mismo lenguaje fuente que P y se debe garantizar que cuando se le presenten

los datos dinámicos produzca los mismos resultados que si se hubiesen presentado todos los datos al programa P original.

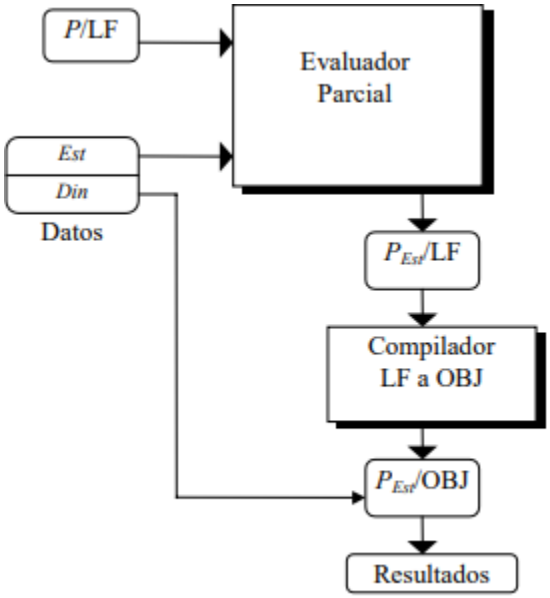

Figura 7: Evaluación Parcial

Labra Gayo, Jose Emilio. (2003). Evaluación Parcial.
http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Ejemplo: Considérese el siguiente fragmento de programa P1 que toma la entrada de dos ficheros diferentes, fichEst y fichDin y escribe el resultado en la salida estándar.

```
read(fichEst,a);
while (a > 0) do
begin
 read(fichDin,b);
 if (a > 10) then write (b - 2 * a)
             else write (a * a + b);
 read(fichEst,a);
end
```

Figura 8: Programa *P1*

Labra Gayo, Jose Emilio. (2003). Programa P1.
http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Si el contenido de fichEst fuese siempre 5 12 7 -1 el evaluador parcial podría generar un programa especializado para dicho conjunto de datos, obteniendo P1Est:

```
read(fichDin,b); write(25 + b);
read(fichDin,b); write(b - 24);
read(fichDin,b); write(49 + b);
```

Figura 9: Programa $P1_{Est}$

Labra Gayo, Jose Emilio. (2003). Programa P1 Est.
http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

Si se conoce de antemano que un programa P va a ejecutarse muchas veces con un mismo conjunto de datos Est pero diferentes datos Din, será más eficiente evaluar parcialmente P para obtener PEst y ejecutar luego Pest. Considérese que en el programa P1 se elimina la última sentencia «read» del bucle. Entonces el evaluador parcial, podría entrar en un bucle infinito intentando generar el programa especializado. Por este motivo, los evaluadores parciales deben realizar un complejo análisis del programa fuente para detectar que el proceso no genere un bucle infinito.

- **Compiladores "Just in Time"**

Con la aparición de Internet surge la necesidad de distribuir programas de una forma independiente de la máquina permitiendo su ejecución en una amplia variedad de plataformas. La interpretación de códigos de bytes supone una demora en los tiempos de ejecución.

En este modelo, una unidad de compilación o clase se transmite en el formato de códigos de bytes, pero no se realiza la interpretación.

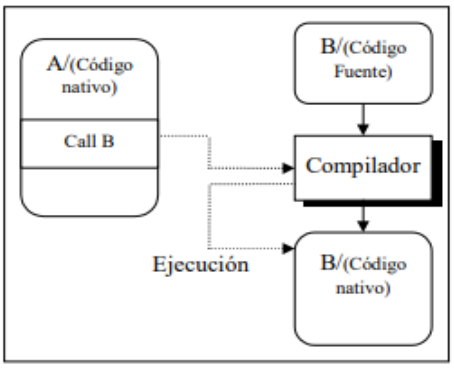

Figura 11: Compilación de una clase *"Just in Time"*

Labra Gayo, Jose Emilio. (2003). Compilación de una clase "Just in Time".
http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf

El sistema realiza dos acciones:

a. Compila la unidad B a código nativo.

b. Continúa la ejecución con el código nativo compilado de la unidad B

- **Compilación continua**

La compilación continua surge como un intento de mejorar la compilación «Just in Time». El sistema mezcla el proceso de compilación a código nativo con el proceso de interpretación.

Código: El código tiene una mezcla de código fuente y código nativo del programa. Inicialmente todo el código está sin compilar, a medida que el programa es ejecutado, el compilador genera traducciones a código nativo de las unidades de compilación.

Compilador: Traduce las unidades de compilación a código nativo. A medida que se finaliza la traducción de una unidad, la versión en código nativa se deja disponible al intérprete.

Intérprete: Comienza interpretando el código fuente, haciendo saltos a las versiones en código nativo a medida que éstas están disponibles.

Monitor: Se encarga de coordinar la comunicación entre los dos módulos anteriores.

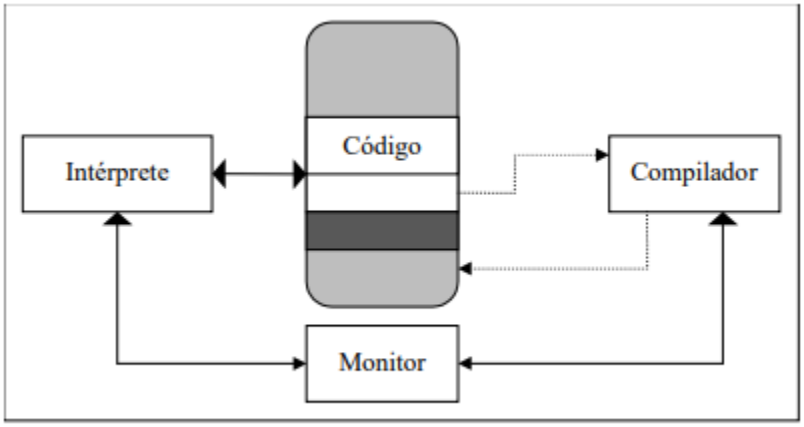

Figura 12: Compilación Continua

Labra Gayo, Jose Emilio. (2003). Compilación continua.
http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/35_InterpretesDLP.pdf