Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Apuntes semana 4 (Oct 12-16, 2020)

Thinking Humanly
“The exciting new effort to make computers
think ...
machines with minds, inthfull and literal sense.” (Haugeland, 1985)
“[The automation of] activities that wassociate with human thinking, activities
such as decision-making, problem solving,
learning ...
” (Bellman, 1978)

Thinking Rationally
“The study of mental faculties through the
use of computational models.”
(Charniak and McDermott, 1985)
“The study of the computations that make
it possible to perceive, reason, and act.”
(Winston, 1992)

Acting Humanly
“The art of creating machines that perform
functions that require intelligence
when performed by people.” (Kurzweil,
1990)
“The study of how to make computers do
things at which, at the moment, people are
better.” (Rich and Knight, 1991)

Acting Rationally
“Computational Intelligence is the study
of the design of intelligent agents.” (Poole
et al., 1998)
“AI . . . is concerned with intelligent behavior
in artifacts.” (Nilsson, 1998)

The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.

• natural language processing to enable it to communicate successfully in English;
• knowledge representation to store what it knows or hears;
• automated reasoning to use the stored information to answer questions and to draw new conclusions;
• machine learning to adapt to new circumstances and to detect and extrapolate patterns.

computer vision to perceive objects, and
robotics to manipulate objects and move about.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

The final element in the philosophical picture of the mind is the connection between knowledge and action. This question is vital to AI because intelligence requires action as well
as reasoning. Moreover, only by understanding how actions are justified can we understand how to build an agent whose actions are justifiable (or rational). Aristotle argued (in De Motu Animalium) that actions are justified by a logical connection between goals and knowledge of the action's outcome (the last part of this extract also appears on the front cover of this book, in the original Greek)

The first nontrivial algorithm is thought to be Euclid's algorithm for computing greatest common divisors. The word algorithm (and the idea of studying them) comes from al-Khowarazmi, a Persian mathematician of the 9th century, whose writings also introduced Arabic numerals and algebra to Europe. Boole and others discussed algorithms for logical

Neuroscience is the study of the nervous system, particularly the brain. Although the exact way in which the brain enables thought is one of the great mysteries of science, the fact that it
does enable thought has been appreciated for thousands of years because of the evidence that
strong blows to the head can lead to mental incapacitation.

Princeton was home to another influential figure in AI, John McCarthy. After receiving his PhD there in 1951 and working for two years as an instructor, McCarthy moved to Stanford and then to Dartmouth College, which was to become the official birthplace of the field. McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence.
They organized a two-month workshop at Dartmouth in the summer of 1956. The proposal states:
10

1.3.1 The gestation of artificial intelligence (1943–1955)

The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943). They drew on three sources: knowledge of the basic physiology and function of neurons in the brain; a formal analysis of propositional logic due to Russell and Whitehead; and Turing's theory of computation. They proposed a model of artificial neurons in which each neuron is characterized as being "on" or "off," with a switch to "on" occurring in response to stimulation by a sufficient number of neighboring neurons. The state of a neuron was conceived of as "factually equivalent to a proposition which proposed its adequate stimulus."

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons. His rule, now called Hebbian HEBBIAN LEARNING learning, remains an influential model to this day. Two undergraduate students at Harvard, Marvin Minsky and Dean Edmonds, built the first neural network computer in 1950. The SNARC, as it was called, used 3000 vacuum tubes and a surplus automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons.

There were a number of early examples of work that can be characterized as AI, but Alan Turing's vision was perhaps the most influential. He gave lectures on the topic as early as 1947 at the London Mathematical Society and articulated a persuasive agenda in his 1950 article "Computing Machinery and Intelligence." Therein, he introduced the Turing Test, machine learning, genetic algorithms, and reinforcement learning.

Looking at the proposal for the Dartmouth workshop (McCarthy et al., 1955), we can see why it was necessary for AI to become a separate field. Why couldn't all the work done in AI have taken place under the name of control theory or operations research or decision theory, which, after all, have objectives similar to those of AI? Or why isn't AI a branch of mathematics? The first answer is that AI from the start embraced the idea of duplicating human faculties such as creativity, self-improvement, and language use. None of the other fields were addressing these issues. The second answer is methodology. AI is the only one of these fields that is clearly a branch of computer science (although operations research does share an emphasis on computer simulations), and AI is the only field to attempt to build machines that will function autonomously in complex, changing environments.

Newell and Simon's early success was followed up with the General Problem Solver, or GPS. Unlike Logic Theorist, this program was designed from the start to imitate human problem-solving protocols. Within the limited class of puzzles it could handle, it turned out that the order in which the program considered subgoals and possible actions was similar to that in which humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach. The success of GPS and subsequent programs as models of cognition led Newell and Simon (1976) to formulate the famous physical symbol system hypothesis, which states that "a physical symbol system has the necessary and PHYSICAL SYMBOL SYSTEM sufficient means for general intelligent action."

1.4 The state of the art.

What can AI do today? A concise answer is difficult because there are so many activities in so many subfields. Here we sample a few applications; others appear throughout the book. Robotic vehicles: A driverless robotic car named STANLEY sped through the rough terrain of the Mojave dessert at 22 mph, finishing the 132-mile course first to win the 2005 DARPA Grand Challenge. STANLEY is a

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Volkswagen Touareg outfitted with cameras, radar, and laser rangefinders to sense the environment and onboard software to command the steering, braking, and acceleration (Thrun, 2006).

Game playing: IBM's DEEP BLUE became the first computer program to defeat the world champion in a chess match when it bested Garry Kasparov by a score of 3.5 to 2.5 in an exhibition match (Goodman and Keene, 1997). Kasparov said that he felt a "new kind of intelligence" across the board from him. Newsweek magazine described the match as "The brain's last stand." The value of IBM's stock increased by $18 billion. Human champions studied Kasparov's loss and were able to draw a few matches in subsequent years, but the most recent human-computer matches have been won convincingly by the computer.

Spam fighting: Each day, learning algorithms classify over a billion messages as spam, saving the recipient from having to waste time deleting what, for many users, could comprise 80% or 90% of all messages, if not classified away by algorithms. Because the spammers are continually updating their tactics, it is difficult for a static programmed approach to keep up, and learning algorithms work best (Sahami et al., 1998; Goodman and Heckerman, 2004).

Logistics planning: During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, and conflict resolution among all parameters. The AI planning techniques generated in hours a plan that would have taken weeks with older methods. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

Robotics: The iRobot Corporation has sold over two million Roomba robotic vacuum cleaners for home use. The company also deploys the more rugged PackBot to Iraq and Afghanistan, where it is used to handle hazardous materials, clear explosives, and identify the location of snipers. Machine Translation: A computer program automatically translates from Arabic to English, allowing an English speaker to see the headline "Ardogan Confirms That Turkey Would Not Accept Any Pressure, Urging Them to Recognize Cyprus." The program uses a statistical model built from examples of Arabic-to-English translations and from examples of English text totaling two trillion words (Brants et al., 2007). None of the computer scientists on the team speak Arabic, but they do understand statistics and machine learning algorithms.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Apuntes semana 5 (Oct 19-23, 2020)

Video 1 (Mod-01 Lec-01 An Overview of a Compiler)

APLICACIONES DE LA TECNOLOGÍA DE COMPILADORES

* IMPLEMENTACIÓN DE LENGUAJES DE ALTO NIVEL.

*TRADUCCIONES DE PROGRAMAS.

*GENERACIÓN DE CÓDIGO MÁQUINA PARA LENGUAJES DE ALTO NIVEL.

*HERRAMIENTAS DE PRODUCTIVIDAD DE SOFTWARE.

*PRUEBA DE SOFTWARE.

*DISEÑO DE ARQUITECTURA DE COMPUTADORAS.

COMPLEJIDAD DE LA TECNOLOGÍA DE COMPILADORES

*Utiliza algoritmos y técnicas de un gran número de áreas de informática.

*Traduce la teoría compleja a la práctica.

* Es el software de sistema más complejo.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

SISTEMA DE PROCESAMIENTO DEL LENGUAJE

programa fuente

1.- PREPROCESADOR

Programa fuente final

2.- COMPILADOR

Código ensamblador

3.- ENSAMBLADOR

Objetos en código máquina

4.- ENLAZADOR

Programa final

ETAPAS DE UN COMPILADOR

Programa fuente

1.- ANALIZADOR LÉXICO

Tokens

2.- ANALIZADOR SINTÁCTICO

Árbol sintáctico

3.- ANALIZADOR SEMÁNTICO

Árbol sintáctico

4.- GENERADOR DE CÓDIGO OBJETO

Código intermedio

5.- OPTIMIZADOR DE CÓDIGO

Código intermedio

6.- GENERADOR DE CÓDIGO OBJETO

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Código objeto

Video 2 (*Mod-02 Lec-02 Lexical Analysis)*

El análisis léxico es la primera fase de un compilador, tiene como entrada el código fuente en cualquier lenguaje de programación el cual es leído carácter por carácter por el compilador y éste mismo nos da como salida componentes léxicos o tokens, que son posteriormente proporcionados al analizador sintáctico.

También nos habla sobre la importancia de resaltar las diferencia entre el análisis léxico y el análisis sintáctico, dándonos como principal razón una simplificación del diseño y mejora de eficiencia de un compilador que va de la mano de la optimización y gestión de un software usando la ingeniería de software.

Los tokens, patrones y lexemas, los componente léxicos o tokens se definen como una secuencia de caracteres con significado sintáctico propio y que son pertenecientes a una categoría léxica (identificador, palabra reservada, literales, operadores o caracteres de puntuación) y estos pueden contener uno o mas lexemas. El lexema es una secuencia de caracteres cuya estructura se corresponde con el patrón de un token y los patrones son la regla que describe los lexemas correspondientes a un token. El patrón es la regla que describe el conjunto de lexemas que pueden representar a un determinado componente léxico en los programas fuente. En otras palabras, es la descripción del componente léxico mediante una regla.

El análisis léxico no es perfecto y tiene sus restricciones o dificultades, para ejemplificar y demostrarlos no pone el ejemplo de las palabras reservadas, ponen de ejemplo a C y PL/1, C tiene las palabras reservadas como while, do, if, else, las cuales PL/1 no contiene, por lo que un compilador diseñado para el lenguaje C no reconocería de manera correcta los tokens en PL/1. El análisis léxico no puede detectar ningún error significativo, excepto errores simples, como símbolos ilegales y otros más simples.

El reconocimiento y especificación de tokens nos habla que se puede realizar mediante un autómata finito, un autómata finito (FA) es una máquina abstracta simple que se utiliza para reconocer patrones dentro de la entrada tomada de algún conjunto de caracteres (o alfabeto) tomando como ejemplo el lenguaje C. El trabajo de un FA es aceptar o rechazar una entrada dependiendo de si el patrón definido por FA ocurre en la entrada.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

T: Terminales / Alfabeto de Símbolos de Entrada, podemos también llamarlas las terminales, o símbolos terminales.

V: Conjunto de variables, también los llaman no terminales o categorías sintácticas. Cada variable representa un conjunto de strings, digamos un sub lenguaje.

S: Conjunto de variables “Especiales”, representa un lenguaje bien definido, lo llaman también el símbolo inicial, también puede ser que haya otras variables que sean auxiliares.

P: Una función de Producción / Derivación, son un conjunto finito de reglas que muestran de manera recursiva una definición del lenguaje.

Es más, esto vale la pena verlo, veamos cómo:

Cada Producción Consta de:

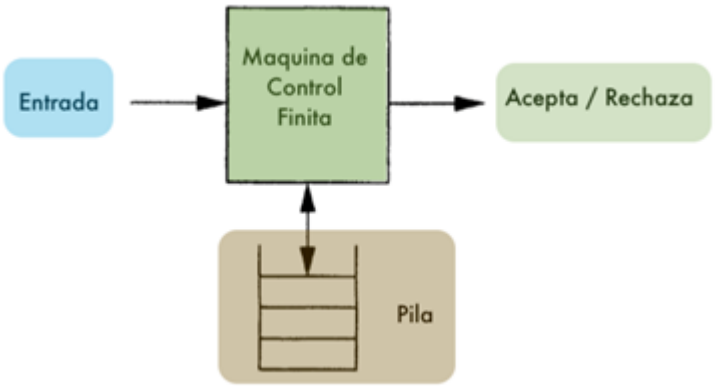
$$Variable \rightarrow Terminales \ \& \ Variables$$

Una variable que está siendo (parcialmente) definida por la producción, esta variable la llaman de forma común la cabeza de la producción.

Una cadena de cero o más variables y quizá algunos terminales, el string lo suelen llamar el cuerpo de la producción, que representa una forma de generar los strings, donde simplemente basta con sustituir las variables que estén dentro del cuerpo.

Podemos pensar en los Autómatas PushDown de la siguiente manera:

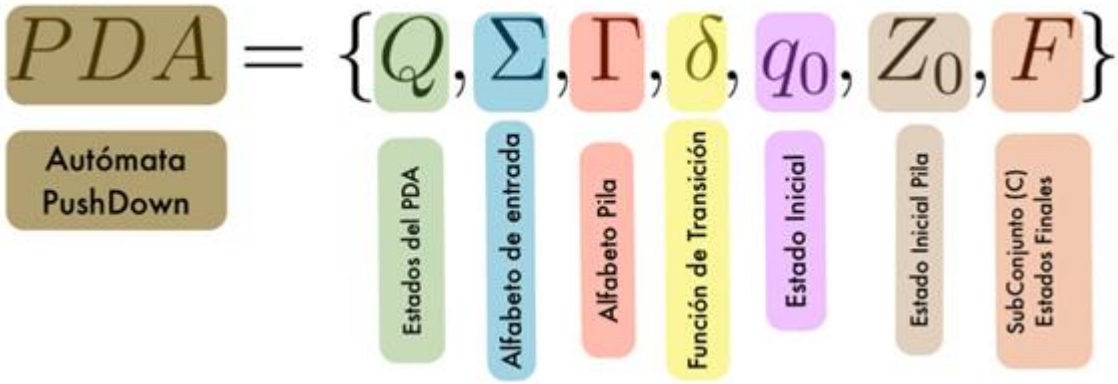
- Un Automata ε-NFA
- Un Pila bastante especial



Así con cada simbolo que va leyendo el autómata puede hacer 2 acciones:

- Moverse a otro estado
- También si lo necesita afecta (push, pull) la pila

De manera totalmente formal podemos definir cualquiera de los PDA como:



Estado Inmediato

Podemos describir el “estado” en cada momento con toda la información del autómata con 3 cosas:

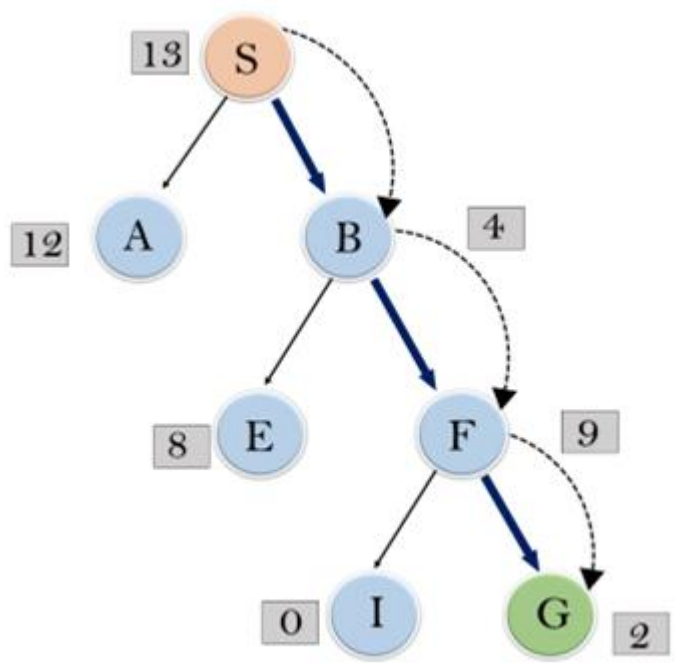
- El estado actual del autómata
- El símbolo que estamos leyendo
- El simbolo en el tope de la pila

Apuntes semana 6 (Oct 26-30, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

informed search

Una búsqueda que utiliza conocimientos específicos del dominio. Suponga que tenemos una forma de estimar qué tan cerca está un estado de la meta, con una función de evaluación. Estrategia general: primero expanda el mejor estado de la lista abierta. A esto se le llama búsqueda "mejor primero" o búsqueda ordenada en el espacio de estados.



Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Estas gramáticas, conocidas también como gramáticas de tipo 2 o gramáticas independientes

del contexto, son las que generan los lenguajes libres o independientes del contexto. Los

lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila

determinístico o no determinístico.

Como toda gramática se definen mediante una cuadrupla G = (N, T, P, S), siendo

- N es un conjunto finito de símbolos no terminales

- T es un conjunto finito de símbolos terminales $N \cap T = \varnothing$

- P es un conjunto finito de producciones

- S es el símbolo distinguido o axioma $S \notin (N \cup T)$

En una gramática libre del contexto, cada producción de P tiene la forma

$A \in N \cup \{S\}$

$A \rightarrow \omega$

$\omega \in (N \cup T)^*$

- {ε}

Es decir, que en el lado izquierdo de una producción pueden aparecer el símbolo distinguido o

un símbolo no terminal y en el lado derecho de una producción cualquier cadena de símbolos

terminales y/o no terminales de longitud mayor o igual que 1.

La gramática puede contener también la producción $S \rightarrow \varepsilon$ si el lenguaje que se quiere

generar contiene la cadena vacía.

Las gramáticas libres del contexto se escriben, frecuentemente, utilizando una notación

conocida como BNF (Backus-Naur Form). BNF es la técnica más común para definir la

sintaxis de los lenguajes de programación.

En esta notación se deben seguir las siguientes convenciones:

- los no terminales se escriben entre paréntesis angulares < >

- los terminales se representan con cadenas de caracteres sin paréntesis angulares

- el lado izquierdo de cada regla debe tener únicamente un no terminal (ya que es una

gramática libre del contexto)

- el símbolo ::=, que se lee “se define como” o “se reescribe como”, se utiliza en lugar de →

- varias producciones del tipo

<A> ::= <B1>

<A> ::= <B2>

<A> ::= <Bn>

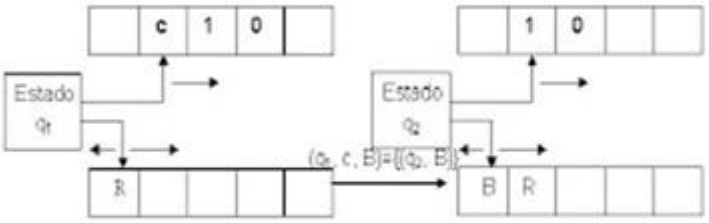
se pueden escribir como <A> ::= <B1> <B2> ... <Bn>

Un autómata de pila o Push-Down es un autómata que cuenta con un mecanismo que permite almacenamiento ilimitado y opera como una pila. El autómata de pila (se abrevia PDA de sus siglas en inglés Push-Down Automata) tiene una cinta de entrada, un control finito y una pila. La pila es una cadena de símbolos de algún alfabeto. El símbolo que se encuentra más a la izquierda se considera como que está en la “cima”.

El dispositivo será no determinístico y tendrá un número finito de alternativas de movimiento en cada situación como se muestra en la siguiente figura un autómata de pila. Los movimientos serán de dos tipos. En el primer tipo de movimiento se utiliza un símbolo de entrada. Dependiendo del símbolo de entrada, del símbolo de la cima y el estado de control finito, es posible un número de alternativas.

Cada alternativa consiste en un estado posterior para el control finito y una cadena (posiblemente vacía) de símbolos, para sustituir al símbolo que se encuentra en la

cima de la pila. Después de seleccionar una alternativa, la cabeza de entrada avanza un símbolo.



El segundo tipo de movimiento conocido como movimiento ε es parecido al primero, excepto que el símbolo de entrada no se utiliza y la cabeza de la entrada no avanza después del movimiento. Este tipo de movimiento permite al PDA manipular la pila sin leer símbolos de entrada

Existen dos modos de aceptar un lenguaje por un autómata de apilamiento. El primero consiste en definir el lenguaje aceptado como el conjunto de todas las entradas para las cuales una sucesión de movimientos ocasiona que el autómata de pila vacíe su pila.

La segunda manera es designando algunos estados como estados finales y definimos el lenguaje aceptado como el conjunto de todas las entradas para las cuales alguna selección de movimiento ocasiona que el autómata de pila accede un estado final.

Apuntes semana 7 (Nov 2-6, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Los **árboles de decisión** son modelos predictivos formados por reglas binarias (si/no) con las que se consigue repartir las observaciones en función de sus atributos y predecir así el valor de la variable respuesta.

Muchos métodos predictivos generan modelos globales en los que una única ecuación se aplica a todo el espacio muestral. Cuando el caso de uso implica múltiples predictores, que interaccionan entre ellos de forma compleja y no lineal, es muy difícil encontrar un único modelo global que sea capaz de reflejar la relación entre las variables. Los métodos estadísticos y de *machine learning* basados en árboles engloban a un conjunto de técnicas supervisadas no paramétricas que consiguen segmentar el espacio de los predictores en regiones simples, dentro de las cuales es más sencillo manejar las interacciones. Es esta característica la que les proporciona gran parte de su potencial.

Los métodos basados en árboles se han convertido en uno de los referentes dentro del ámbito predictivo debido a los buenos resultados que generan en problemas muy diversos. A lo largo de este documento se explora la forma en que se construyen y predicen los árboles de decisión (clasificación y regresión), elementos fundamentales de modelos predictivos más complejos como *Random Forest* y *Gradient Boosting Machine*.

La tabla de análisis es un poco más compleja, tiene dos partes, la parte de acción y la parte GOTO. La parte de acción tiene cuatro tipos de entrada, cambio, reducción, aceptación y error; la tabla GOTO se utiliza para proporcionar la siguiente información de estado, que es realmente necesaria después de un movimiento de reducción.

Entonces, antes de eso, veamos esta tabla del analizador para comprender la acción e ir a la entrada es un poco mejor. Entonces, la tabla del analizador está indexada por los números de estado en un lado y en el otro lado para la tabla de acción está indexada por los tokens, presentes en la entrada y para la tabla de ir la tabla está indexada por los no terminales presentes en la gramática.

Un autómata de pila o Push-Down es un autómata que cuenta con un mecanismo que permite almacenamiento ilimitado y opera como una pila. El autómata de pila (se abrevia PDA de sus siglas en inglés Push-Down Autómata) tiene una cinta de entrada, un control finito y una pila.

La pila es una cadena de símbolos de algún alfabeto. El símbolo que se encuentra más a la izquierda se considera como que está en la “cima”.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

El dispositivo será no determinístico y tendrá un número finito de alternativas de movimiento en cada situación. Los movimientos serán de dos tipos. En el primer tipo de movimiento se utiliza un símbolo de entrada. Dependiendo del símbolo de entrada, del símbolo de la cima y el estado de control finito, es posible un número de alternativas.

Cada alternativa consiste en un estado posterior para el control finito y una cadena (posiblemente vacía) de símbolos, para sustituir al símbolo que se encuentra en la cima de la pila.

Entonces, de manera similar, si toma el estado 10, todas las cadenas que pueden llevarlo de 0 a 10 son los prefijos viables para los que son válidos los elementos del

estado 10. Entonces, eso es sobre él validez de los elementos, la construcción de L R 0 DFA utilizando conjuntos de elementos es muy simple, por lo que veamos

ese procedimiento de vez en cuando y luego veamos la relevancia de estos dos

problemas.

Entonces, hay una operación llamada cierre, así que primero déjame explicarte la operación de cierre. Con respecto a estos ejemplos y luego observe el algoritmo en sí.

Entonces, digamos que se les da un elemento S que va al punto E hash, el cierre dice que mire el símbolo después del punto si eso digamos no terminal, luego agregue todas las producciones de ese no terminal en particular con un punto en el y ponga un punto en la posición más a la izquierda, para E hay 3 producciones E a E más T E a E menos T y E a T. Entonces, todos estos 3 elementos se han agregado con un punto en la posición más a la izquierda, ahora haga esto para el elemento que acabamos de agregar, nuevamente tenemos un símbolo E aquí y no hay nada más que agregar para E. Pero, esto nos da un nuevo símbolo T, así que agregue todas las producciones y el elemento asociado con él al estado, por lo que T yendo al paréntesis E paréntesis y T yendo a i d con un punto en la posición más a la izquierda.

Apuntes semana 8 (Nov 9-13, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Un atributo es una variable que representa una propiedad de un símbolo (terminal o no terminal).

Ejemplos:

El proceso de calcular un atributo y asociar su valor calculado con la construcción del lenguaje en cuestión se define como fijación del atributo. El momento en el que se realiza este proceso de fijación, que puede ser compilación o ejecución se denomina tiempo de fijación.

Gramáticas de atributos

Las gramáticas independientes del contexto, las de tipo 2 de la jerarquía de Chomsky, son las que utilizamos para definir la sintaxis de un lenguaje de programación.

De esta forma definimos la estructura que tendría cualquier programa escrito en ese lenguaje de programación elegido. Una vez hemos validado la estructura del programa, y para poder realizar determinados controles semánticos necesitamos utilizar información, representada por los atributos, que asociamos a los símbolos terminales y no terminales de la gramática que hemos establecido para ese lenguaje. Estas gramáticas se denominan gramáticas de atributos.

Para poder realizar este paso de valores de los atributos, para cada regla gramatical se establecen unas operaciones con estos atributos, y se denominan reglas semánticas, puesto que nos permiten realizar los controles semánticos que se han establecido para este lenguaje de programación.

La forma de denotar los atributos asociados a un símbolo es de la siguiente forma: s i X es un símbolo gramatical y a es un atributo de este símbolo gramatical, escribimos X.a para indicar que el valor de a está asociado a este símbolo.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

La forma de denotar los atributos asociados a un símbolo es de la siguiente forma: si **X** es un símbolo gramatical y **a** es un atributo de este símbolo gramatical, escribimos **X.a** para indicar que el valor de **a** está asociado a este símbolo.

Atributo	Dominio (posibles valores)	Descripción
X.tipo	Entero, real, booleano, carácter, array, registro, puntero o función.	El tipo que tiene el símbolo no terminal X.
X. valor	Real	El valor que obtendrá el símbolo no terminal X.

En función de cómo se obtienen los atributos, estos pueden ser de dos tipos (posteriormente entraremos en más detalle):

1. Sintetizados: el valor que se asigna a un nodo depende del valor de los nodos hijos.

2. Heredados: el valor de un nodo depende del valor de los hermanos y del padre.

Las relaciones entre los valores que pueden tener los atributos de una gramática de atributos se especifican mediante reglas semánticas.

Gramática incontextual

Un conjunto de atributos semánticos que se asociarán a

las categorías sintácticas

categoriaX.a1, categoriaX.a2, …, categoriaX.an

categoriaY.a1, categoriaY.a2, …, categoriaY.an…

Un conjunto de ecuaciones semánticas que servirán para

calcular el valor de los atributos de la categoría sintáctica del

lado izquierdo de las producciones usando los valores de los

atributos de las categorías sintácticas del derecho

a1 = f1 (ax, ax+1, ax+2 …, ay)

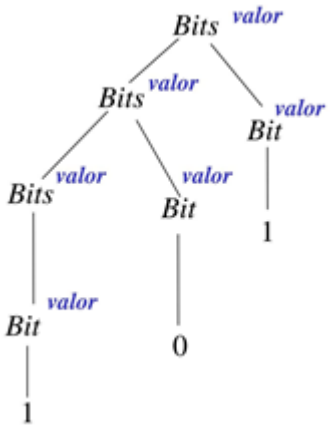
a2 = f2 (ax, ax+1, ax+2 …, ay)…

Las funciones f1, f2,… se llaman funciones semánticas.

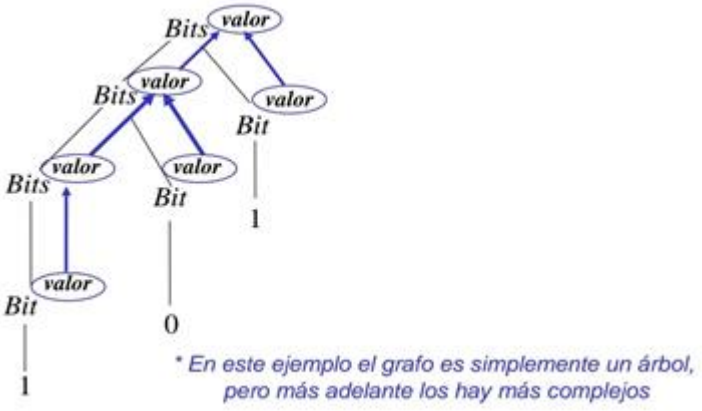
Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

```
Bits ::= Bits Bit
   Bits0.valor = Bits1.valor * 2 + Bit.valor
Bits ::= Bit
   Bits.valor = Bit.valor
Bit ::=  0
   Bit.valor = 0
Bit ::= 1
   Bit.valor = 1
```

Los nodos-padre de los árboles sintácticos, decorados por los atributos, se vuelven árboles sintácticos atribuidos.

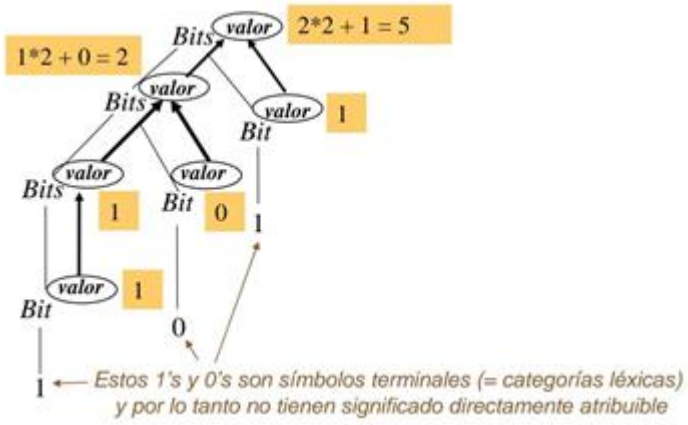


Las ecuaciones semánticas introducen dependencias que se muestran como grafos de dependencias de atributos Los nodos son los atributos y los arcos van de los atributos de la derecha de una ecuación al atributo de la izquierda.



El valor de un atributo se calcula evaluando la función semántica correspondiente sobre los otros atributos, según el orden del grafo de dependencias de atributos.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014



Hay varios tipos de atributos interesantes para definir la gramática de atributos de un lenguaje de programación.

Los atributos léxicos se asocian a todos los terminales de la gramática (= categorías léxicas) y contienen el valor concreto de los lexemas que han sido reconocidos categoria-léxicaX.lex categoria-léxicaY.lex …

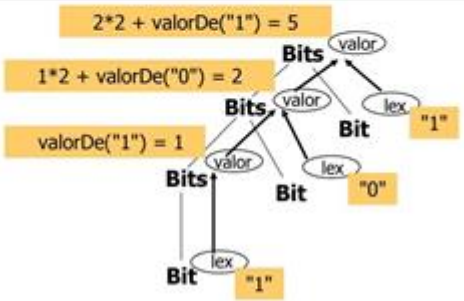
Como la gramática no captura la microsintaxis (nos quedamos en las categorías léxicas) hace falta alguna manera de acceder a la cadena concreta que hay por debajo.

Ej. El nombre de un Identificador, los dígitos de un Número, etc.

Asumiremos que vienen prefijados "de manera externa" a la gramática de atributos.

Ej. Lo hace el analizador léxico.

```
Bits ::= Bits Bit
   Bits0.valor = Bits1.valor * 2 + valorDe(Bit.lex)
Bits ::= Bit
   Bits.valor = valorDe(Bit.lex)
Bit ::= 0
   Bit.lex = "0"
Bit ::= 1
   Bit.lex = "1"
```



Todos los atributos que se calculan según el principio de composicionalidad se llaman atributos sintetizados.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Ej. Los atributos anteriores lex y valor Las gramáticas de atributos que únicamente poseen atributos sintetizados se llaman gramáticas s-atribuidas.

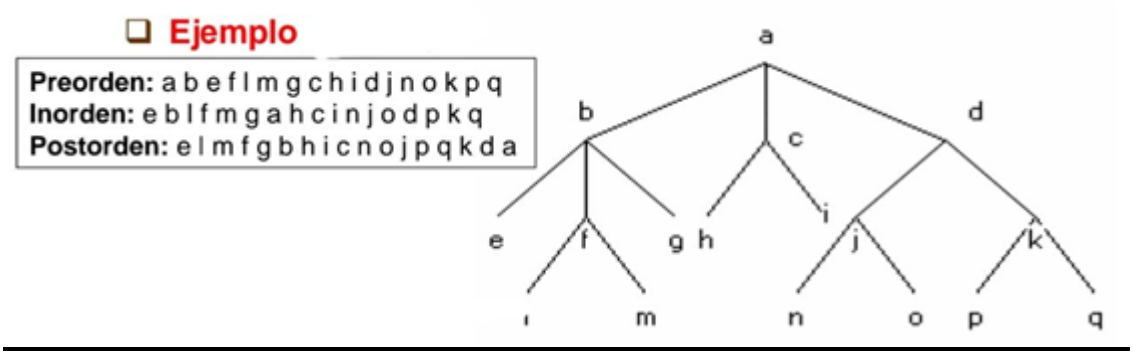
La evaluación en una gramática s-atribuida se puede hacer con un recorrido en postorden de los árboles sintácticos atribuidos.

Todos los atributos que se calculan según el principio decomposicionalidad se llaman atributos sintetizados.

Los atributos anteriores lex y valor.

Las gramáticas de atributos que únicamente poseen atributos sintetizados se llaman gramáticas s-atribuidas.

La evaluación en una gramática s-atribuida se puede hacer con un recorrido en postorden de los árboles sintácticos atribuidos.



Apuntes semana 9 (Nov 23-27, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

El analizador semántico utiliza el árbol sintáctico y la información en la tabla de símbolos para comprobar la consistencia semántica del programa fuente con la definición del lenguaje. También recopila información sobre el tipo y la guarda, ya sea en el árbol sintáctico o en la tabla de símbolos, para usarla más tarde durante la generación de código intermedio.

Una parte importante del análisis semántico es la comprobación (verificación) de tipos, en donde el compilador verifica que cada operador tiene operandos que coincidan. Por ejemplo, muchas definiciones de lenguajes de programación requieren que el índice de un arreglo sea entero; el compilador debe reportar un error si se utiliza un número de punto flotante para indexar el arreglo.

La especificación del lenguaje puede permitir ciertas conversiones de tipo conocidas como coerciones. Por ejemplo, puede aplicarse un operador binario aritmético a un par de enteros o a un par de números de punto flotante. Si el operador se aplica a un número de punto flotante y a un entero, el compilador puede convertir u obligar a que se convierta en un número de punto flotante.

Las gramáticas de atributos son un formalismo simple para la especificación de la semántica de lenguajes formales, como ser lenguajes de programación o de especificación. Integran la modularidad que brindan las gramáticas libres de contexto y la expresividad de un lenguaje funcional.

En una gramática de atributos, se relaciona con cada símbolo de una gramática libre de contexto un conjunto de atributos. Cada regla o producción tiene asociados un conjunto de reglas semánticas que toman la forma de asignación a atributos de valores denotados por la aplicación de una función, la cual puede tomar como argumentos instancias de atributos pertenecientes a los símbolos que aparecen en la producción.

Apuntes semana 10 (Nov 23-27, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Apuntes semana 11 (30 Nov- 4 Dic, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Esta segunda edición revisada de Engineering a Compiler está llena de actualizaciones técnicas y nuevo material que cubre los últimos desarrollos en tecnología de compiladores. En este texto completo aprenderá técnicas importantes para construir un compilador moderno. Los principales educadores e investigadores Keith Cooper y Linda Torczon combinan principios básicos con ideas pragmáticas de su experiencia en la creación de compiladores de última generación. técnicas importantes, como la compilación de lenguajes imperativos y orientados a objetos, la construcción de formularios estáticos de asignación única, la programación de instrucciones y la asignación de registros para colorear gráficos. Tratamiento en profundidad de algoritmos y técnicas utilizados en un compilador moderno. Céntrese en la optimización de código y la generación de código, las principales áreas de investigación y desarrollo recientes. Mejoras en la presentación, incluyendo resúmenes conceptuales para cada capítulo, resúmenes y preguntas de revisión para secciones, y colocación prominente de definiciones para nuevos términos.

Para estos sistemas de software, mejorando esta nueva edición con lecciones claras y los detalles que simplemente debe hacerlo bien, mientras se mantiene firmemente a la vista el panorama general. La ingeniería de un compilador es un compañero invaluable para cualquier persona nueva en el tema.

La segunda edición de Engineering a Compiler es una excelente introducción a la construcción de compiladores de optimización modernos. Los autores se basan en una gran experiencia en compiladores construcción con el fin de ayudar a los estudiantes a comprender el panorama general y al mismo tiempo guiar a través de muchos detalles importantes pero sutiles que deben abordarse para construir un compilador de optimización eficaz.

Keith D. Cooper es el profesor Doerr de Ingeniería Computacional en la Universidad de rice. Él ha trabajado en una amplia colección de problemas en la optimización del código compilado, incluido el análisis de flujo de datos entre procedimientos y sus aplicaciones, numeración de valores, asociación algebraica, asignación de registros y programación de instrucciones. Su trabajo reciente se ha centrado en un fundamental examen de la estructura y comportamiento de los compiladores tradicionales. Ha enseñado una variedad de cursos a nivel de pregrado, desde programación introductoria hasta optimización de código a nivel de posgrado. Es miembro de la ACM.

Linda Torczon, investigadora científica sénior, Departamento de Ciencias de la Computación de la Universidad de rice, es investigadora principal del proyecto de entorno de compilación compatible con plataformas

PACE: un proyecto patrocinado por DARPA que está desarrollando un entorno de compilación optimizado que ajusta automáticamente sus optimizaciones y estrategias a nuevas plataformas. Desde 1990 hasta 2000, el Dr. Torczon se desempeñó como director ejecutivo del Centro de Investigación sobre

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Computación Paralela CRPC, un Centro de Ciencia y Tecnología de la Fundación Nacional de Ciencias.

Los conocimientos y las mejores prácticas en este campo cambian constantemente. A medida que nuevas investigaciones y experiencias amplían nuestra comprensión, los cambios en métodos de investigación o prácticas profesionales puede resultar necesario. Los profesionales e investigadores deben confiar siempre en sus propios experiencia y conocimiento en la evaluación y uso de cualquier información o métodos descritos en este documento. Al usar dicha información o métodos. Deben ser conscientes de su propia seguridad y la seguridad de los demás, incluidas las partes por las que tienen una responsabilidad profesional.

Apuntes semana 12 (Dic 7-11, 2020)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

En la teoría del lenguaje formal , una gramática libre de contexto (CFG) es una gramática formal en la que cada regla de producción tiene la forma:

A --> a

Dónde **A** es un solo símbolo no terminal , y **α** es una cadena de terminales y / o no terminales (**α** puede estar vacío). Una gramática formal se considera "libre de contexto" cuando sus reglas de producción se pueden aplicar independientemente del contexto de un no terminal. Independientemente de los símbolos que lo rodeen, el no terminal único en el lado izquierdo siempre se puede reemplazar por el lado derecho. Esto es lo que lo distingue de una gramática sensible al contexto .

Una gramática formal es esencialmente un conjunto de reglas de producción que describen todas las cadenas posibles en un lenguaje formal dado. Las reglas de producción son reemplazos simples. Por ejemplo, la primera regla en la imagen,

(Stmt) --> (Id) – (Expr);

Reemplaza **(Stmt)** con **(Id) – (Expr)**. Puede haber varias reglas de reemplazo para un símbolo no terminal dado. El lenguaje generado por una gramática es el conjunto de todas las cadenas de símbolo terminales que pueden derivarse, mediante aplicaciones de reglas repetidas, de algún símbolo no terminal particular ("símbolo de inicio"). Los símbolos no terminales se utilizan durante el proceso de derivación, pero es posible que no aparezcan en su cadena de resultado final.

Los lenguajes generados por gramáticas libres de contexto se conocen como lenguajes libres de contexto (CFL). Diferentes gramáticas libres de contexto pueden generar el mismo lenguaje libre de contexto. Es importante distinguir las propiedades del lenguaje (propiedades intrínsecas) de las propiedades de una gramática particular (propiedades extrínsecas). La cuestión de la igualdad del lenguaje es indecidible .

Las gramáticas libres de contexto surgen en lingüística donde se utilizan para describir la estructura de oraciones y palabras en un lenguaje natural , y de hecho fueron inventadas por el lingüista Noam Chomsky para este propósito. Por el contrario, en informática , a medida que aumentaba el uso de conceptos definidos de forma recursiva, se utilizaban cada vez más. En una aplicación temprana, las gramáticas se utilizan para describir la estructura de los lenguajes de programación . En una aplicación más reciente, se utilizan en una parte esencial del lenguaje de marcado extensible (XML) denominada Definición de tipo de documento.

En lingüística , algunos autores utilizan el término gramática de estructura de frases para referirse a gramáticas libres de contexto, por lo que las gramáticas de estructura de frases son distintas de las gramáticas de dependencia . En informática , una notación popular para gramáticas libres de contexto es la forma Backus-Naur o BNF.

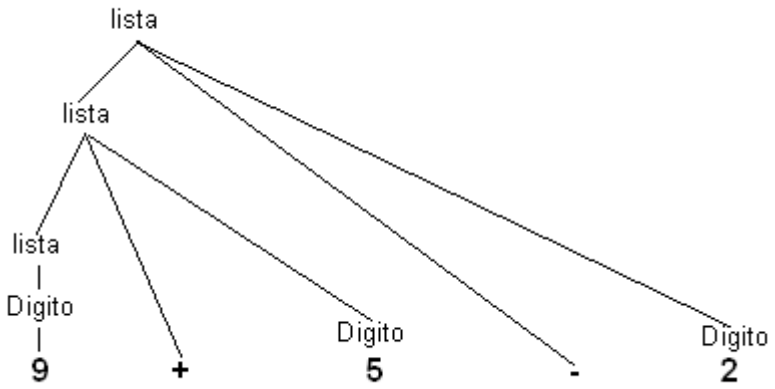
Una GIC está compuesta por 4 elementos:

1. Símbolos terminales (elementos que no generan nada)
2. No terminales (elementos del lado izquierdo de una producción, antes de la flecha "->")
3. Producciones (sentencias que se escriben en la gramática)
4. Símbolo inicial (primer elemento de la gramática)

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Ejemplo 1: Teniendo un lenguaje que genera expresiones de tipo:
9 + 5 - 2
Para determinar si una GIC está bien escrita se utilizan los árboles de análisis sintáctico , así:
Producciones:
lista -> lista + dígito
lista -> lista - dígito
lista -> dígito
dígito -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Arbol de analisis sintactico:



La gramática es correcta siempre y cuando el símbolo inicial esté al lado izquierdo de las producciones y sea la raíz del árbol.

Apuntes semana 13 (Dic 14-18, 2020)

La mayoría de los lenguajes de programación asocian una colección de propiedades con cada valor de datos. Llamamos a esta colección de propiedades el tipo de valor.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

El tipo especifica un conjunto de propiedades que tienen en común todos los valores de ese tipo. Los tipos se pueden especificar por membresía; por ejemplo, un entero podría ser cualquier número entero i en el rango -2^31 i <231, o el rojo podría ser un valor en un tipo enumerado de colores, definido como el conjunto fred, naranja, amarillo, verde, azul, marrón, negro, blanco g. Los tipos pueden ser especificado por reglas; por ejemplo, la declaración de una estructura en c define un tipo. En este caso, el tipo incluye cualquier objeto con los campos declarados en la orden declarada; los campos individuales tienen tipos que especifican lo permitido rangos de valores y su interpretación. (Representamos el tipo de estructura como el producto de los tipos de sus campos constituyentes, en orden). Los tipos están predefinidos por un lenguaje de programación; otros están construidos por el programador. El conjunto de tipos en un lenguaje de programación, junto con las reglas que usan tipos para especificar el comportamiento del programa, se denominan colectivamente un sistema de tipos.

El concepto de sistema todavía hoy en día se sigue redefiniendo, no porque sea una realidad cambiante sino porque el término posee diferentes connotaciones según el contexto que se esté considerando y es uno de los términos más utilizados con diferentes significados. Aun así, se tiene una idea general e intuitiva de qué es un sistema. Basta con mirar alrededor para darse cuenta de que el mundo está formado por ellos: colecciones complejas de elementos altamente relacionados, en los que todo va más allá de la suma de las partes que lo componen (Platero, 2012).

Desde el punto de vista ingenieril, el concepto de sistema queda muy bien reflejado por las siguientes definiciones, dadas por Aracil:

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

“Sistema es una entidad formada por un conjunto de elementos o componentes básicos del sistema, y por las relaciones existentes entre ellos, así como con el entorno. Estas relaciones se expresan formalmente empleando lenguaje matemático”.

“Sistema es un objeto en el que variables de distintos tipos interactúan y producen señales observables. Las señales observables que nos son de interés se suelen denominar salidas. El sistema está afectado también por estímulos externos. Las señales externas que pueden ser manipuladas por el observador se denominan entradas; las que no se pueden manipular se denominan perturbaciones y se dividen en aquellas que son directamente medibles y aquellas que son sólo observables por su influencia sobre la salida” (Figura 1.14).

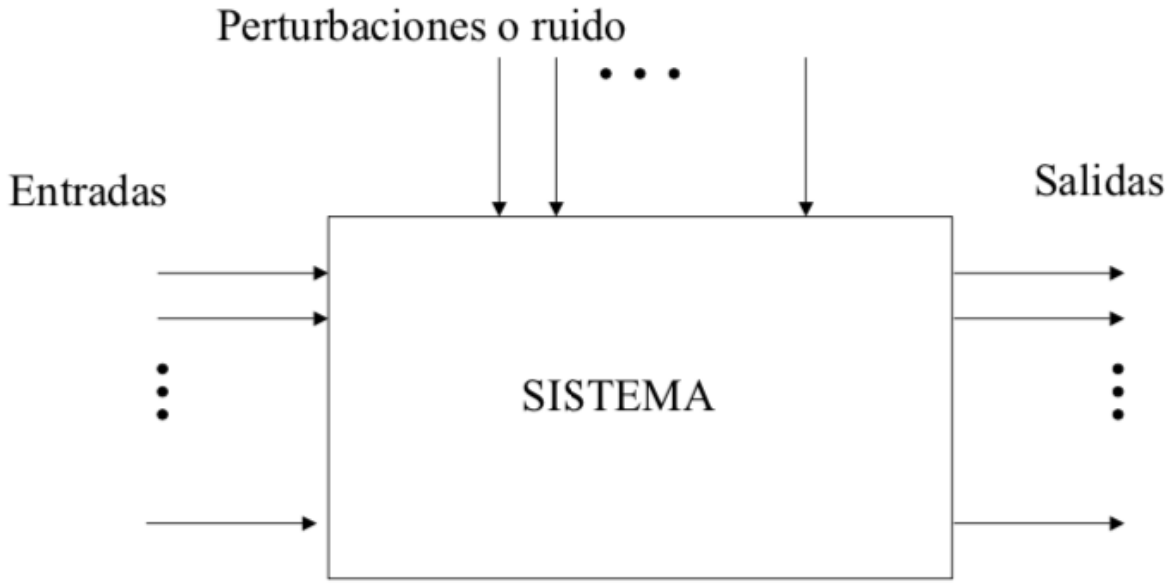


Figura 1.14: Flujo de la información en los sistema

Atendiendo a esta última definición se pueden distinguir cuatro tipos bien diferenciados de variables:

- variables de entrada
- variables de salida
- variables de estado
- perturbaciones

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Apuntes semana 14 (Ene 7-8, 2021)

El diseño del lenguaje intermedio difiere típicamente del lenguaje de máquina de tres maneras fundamentales:

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Cada instrucción representa exactamente una operación fundamental; por ejemplo, los modos de direccionamiento "shift-add" (desplazar y añadir) comunes en microprocesadores no están presentes.

La información de la estructura de control puede no estar incluida en el juego de instrucciones.

El número de registros disponibles puede ser grande, incluso ilimitado.

Un formato popular para lenguajes intermedios es el de código de tres direcciones.

El término también es usado para referirse a lenguajes usados como intermedios por algunos lenguajes de alto nivel que no crean código objeto o código máquina por sí mismos, sino que solamente el lenguaje intermedio.

Este lenguaje intermedio es enviado a un compilador para tal lenguaje, que genera el código objeto o código máquina finalizado. Esto se suele hacer para facilitar el proceso de optimización o para incrementar la portabilidad mediante el uso de un lenguaje intermedio que tenga compiladores para la mayoría de CPUs y sistemas operativos, como puede ser C.

Los lenguajes usados para esto tienen una complejidad que se sitúa entre los lenguajes de alto y bajo nivel, como el lenguaje ensamblador.

Una representación intermedia es una estructura de datos creada a partir de los datos de entrada de un programa informático y de la que parte o la totalidad de los datos de salida son construidos por turno, el uso del término suele implicar que la mayoría de la información presente en la entrada es guardada por la representación intermedia junto con más anotaciones o características de búsqueda rápida.

Un ejemplo canónico se encuentra en la mayoría de compiladores modernos, donde el texto linear entendible por los humanos es transformado en un grafo que permite el análisis del flujo de datos y recolocaciones antes de empezar a crear la lista de instrucciones del CPU que harán el trabajo.

El uso de una representación intermedia permite a sistemas de compiladores como GNU GCC y LLVM poder tener como destino diferentes códigos fuente y admiten su generación para diferentes tipos de arquitectura.

En el proceso de traducir un programa fuente a código destino, un compilador puede construir una o más representaciones intermedias, las cuales pueden tener una variedad de formas. Los árboles sintácticos son una forma de representación intermedia; por lo general, se utilizan durante el análisis sintáctico y semántico.

Después del análisis sintáctico y semántico del programa fuente, muchos compiladores generan un nivel bajo explícito, o una representación intermedia similar al código máquina, que podemos considerar como un programa para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir en la máquina destino.

Morelos Lira Jose Luis 171080082
Olivares Cordero Victor Hugo 151080014

Existe una forma intermedia llamada código de tres direcciones, que consiste en una secuencia de instrucciones similares a ensamblador, con tres operandos por instrucción. Cada operando puede actuar como un registro. La salida del generador de código intermedio en la figura, consiste en la secuencia de código de tres direcciones.

t1 = inttofloat(60)

t2 = id3 * t1

t3 = id2 + t2

id1 = t3

Hay varios puntos que vale la pena mencionar sobre las instrucciones de tres direcciones. En primer lugar, cada instrucción de asignación de tres direcciones tiene, por lo menos, un operador del lado derecho. Por ende, estas instrucciones corrigen el orden en el que se van a realizar las operaciones; la multiplicación va antes que la suma en el programa fuente. En segundo lugar, el compilador debe generar un nombre temporal para guardar el valor calculado por una instrucción de tres direcciones. En tercer lugar, algunas "instrucciones de tres direcciones" como la primera y la última en la secuencia anterior, tienen menos de tres operandos.