

AUTÓMATAS SEMANA 4

MIS APUNTES (CRISTOPHER HERNAN MARTINEZ CORTES)

un tipo de compilador es HTML. HTML se basa en tecnologías de un compilador, es un script de java y se pueden ejecutar dentro de un navegador.

las computadoras tiene ya predeterminado el compilador para poder correr c o c++ o cualquier otro idioma de alto nivel. si es necesario se deben de ejecutar para ver el rendimiento del hardware, entonces a esto se le va a conocer como compilador de desarrollo de hardware de bucle. esto es muy útil para las empresas para la crear chips.

VHDL o VDL es una descripción muy alta de lenguajes.

la simulación es un programa de computadora que se genera para un programa en general.

la complejidad de un compilador surge del hecho de que se requiere mapear un programa como requisitos en lenguaje de alto nivel, entonces estamos hablando de un programa de CA.

Opini3n Isaac Rosales JD.

La verdad me costo mucho entender los videos ya que estaban en otro idioma pero a grandes rasgos un compilador es un programa que lee un programa escrito es un lenguaje, el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto. Como parte importante de este proceso de traducci3n, el compilador informa a su usuario de la presencia de errores en el programa fuente.

Los lenguajes objeto son igualmente variados, un lenguaje objeto puede ser otro lenguaje de programaci3n o el lenguaje de m1quina de cualquier computador entre un microprocesador y un supercomputador. A pesar de existir una aparente complejidad por la clasificaci3n de los compiladores, como se vio en el tema anterior, las tareas b1sicas que debe realizar cualquier compilador son esencialmente las mismas. Al comprender tales tareas, se pueden construir compiladores para una gran diversidad de lenguajes fuente y m1quinas objeto utilizando las mismas t1cnicas b1sicas.

Nuestro conocimiento sobre c3mo organizar y escribir compiladores ha aumentado mucho desde que comenzaron a aparecer y es mundo que debemos de entender.

Opinion Zendejas Mendez Edwin

El an1lisis l1xico es la primer parte de un compilador, este opera a petici3n del analizador sint1ctico que es un componente l1xico.

Sabemos que un compilador en la vida de un programador es una parte fundamental ya que la mayor1a de veces en inform1tica lo utilizamos ya sea para una base de datos, una p1gina

de internet, algún programa, etc.

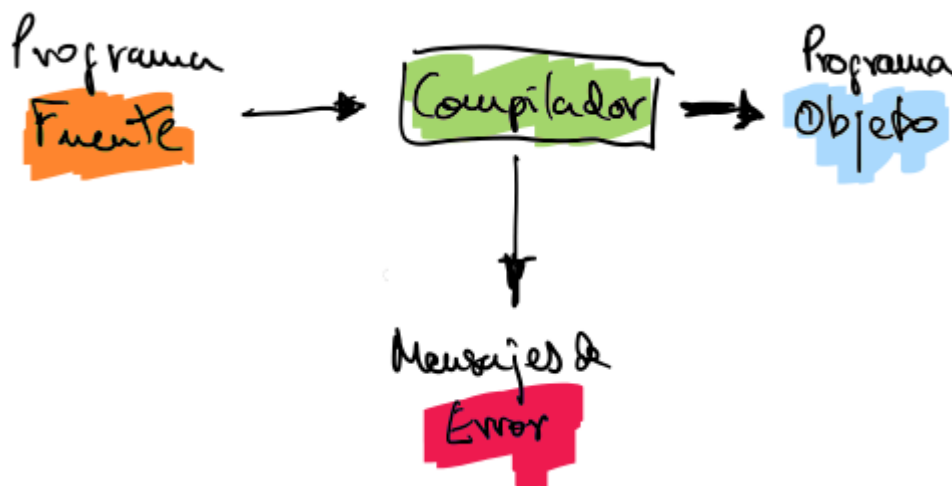
Los más comunes son Java, JavaScript, C#, C++, por mencionar algunos.

Estos compiladores han ido incrementando en cuanto a su tecnología ya que los primeros compiladores eran robustos en cuanto a memoria y en cuanto a velocidad de trabajar, pero actualmente han estado implementando varias opciones para poder facilitar más su uso y que sea de más beneficio para la sociedad informática y para todos aquellos que tengan la necesidad de usar dichos compiladores.

OZALDE ALDAY VIRIDIANA ELIZABETH

Los compiladores en la actualidad están siendo importantes en la conquista del mundo en cada una de las cosas que están presentes en nuestras vidas lo podemos encontrar en el trabajo por ejemplo en la parte de los ratones y mouses de la computadora que funciona con programas que facilitan la interacción entre los humanos y las máquinas.

En un comienzo las máquinas solo se encargaban de ejecutar instrucciones que solo se basan en códigos numéricos que señalaban cada una de las operaciones que se realizaban por lo cual el primer lenguaje se llamaba máquina a través de este se ha ido evolucionando en microprogramas que tiene un mejor objetivo y pueden ser más amigables con el usuario.



Resumen equipo

como equipo sabemos los principales compiladores y empezamos a entender que es un compilador, comprendemos la importancia de los compiladores y como cambian constantemente, ya que estos cada vez son más avanzados y en un futuro podrán hacer casi cualquier cosa. Para nosotros es necesario el saber para que se utilizan y como los implementamos en la vida cotidiana.

También hemos podido identificar sus principales características como por ejemplo que para cada lenguaje de programación se requiere un compilador separado ya que El compilador traduce todo el programa antes de ejecutarlo.

Los programas compilados se ejecutan más rápido que los interpretados, debido a que han sido completamente traducidos a lenguaje máquina.

Informa al usuario de la presencia de errores en el programa fuente.

Poseen un editor integrado con un sistema de coloreado para los comandos, funciones, variables y demás partes de un programa.

Capítulos "Engineering a Compiler".

En resumen de lo que leímos este primer capítulo el papel de la computadora en la vida diaria crece cada año. Internet, las computadoras y el software que se ejecuta en ellas brindan comunicaciones, noticias, entretenimiento y seguridad. Todas estas aplicaciones informáticas se basan en programas informáticos de software que construyen herramientas virtuales sobre las abstracciones de bajo nivel proporcionadas por el hardware subyacente. Casi todo ese software es traducido por una herramienta llamada compilador.

Un compilador es simplemente un programa de computadora que trans- Compilador un programa de computadora que traduce otros programas de computadora a otros programas de computadora para prepararlos para su ejecución. Ingeniería de un compilador.

Visión de conjunto

Los lenguajes de programación están diseñados para ofrecer expresividad, concisión y claridad. Los lenguajes de programación están, en general, diseñados para permitir que los humanos expresen cálculos como secuencias de operaciones. Los procesadores de computadora, en lo sucesivo denominados procesadores, microprocesadores o máquinas, están diseñados para ejecutar secuencias de operaciones. Las operaciones que implementa un procesador son, en su mayor parte, a un nivel de abstracción mucho más bajo que las especificadas en un lenguaje de programación.

Por ejemplo, un lenguaje de programación normalmente incluye una forma concisa de imprimir algún número en un archivo. Esa declaración única del lenguaje de programación debe traducirse literalmente en cientos de operaciones de la máquina antes de que pueda ejecutarse.

La tarea del escáner es transformar un flujo de caracteres en un flujo de palabras en el idioma de entrada. Cada palabra debe clasificarse en una categoría sintáctica o «parte del discurso». El escáner es la única pasada en el compilador para tocar cada carácter en el programa de entrada. Los escritores de compiladores dan mucha importancia a la velocidad en el escaneo, en parte porque la entrada del escáner es mayor, en cierta medida, que la de cualquier otro paso, y, en parte, porque las técnicas altamente eficientes son fáciles de entender e implementar. El escáner es la única pasada en el compilador que manipula cada carácter del programa de entrada.

El escáner de un compilador lee un flujo de entrada que consta de caracteres y produce un flujo de salida que contiene palabras, cada una etiquetada con su. Para que una

clasificación de palabras de acuerdo con su uso gramatical logre esta agregación y clasificación, el escáner aplica un conjunto de reglas que describen la estructura léxica del lenguaje de programación de entrada, Microsyntax a veces llamado microsyntax.

SEMANA 5

OZALDE ALDAY VIRIDIANA ELIZABETH

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática. Los componentes léxicos son los símbolos terminales de la gramática. Suele implementarse como una subrutina del analizador sintáctico.

Otras funciones secundarias:

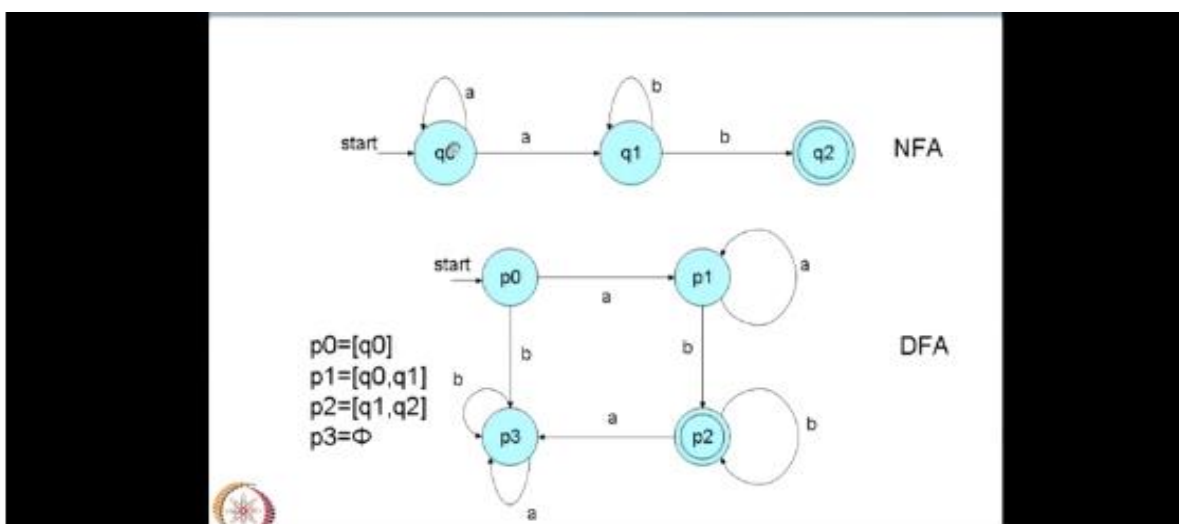
- Manejo del fichero de entrada del programa fuente: abrirlo, leer sus caracteres, cerrarlo y gestionar posibles errores de lectura.
- Eliminar comentarios, espacios en blanco, tabuladores y saltos de línea (caracteres no válidos para formar un token).
- Inclusión de ficheros: `# include ...`
- La expansión de macros y funciones inline: `# define ...`
- Contabilizar el número de líneas y columnas para emitir mensajes de error.
- Reconocimiento y ejecución de las directivas de compilación (por ejemplo, para depurar u optimizar el código fuente).

MARTINEZ CORTES CRISTOPHER HERNAN

un autómata determinista tiene exactamente transiciones en cada símbolo del alfabeto en cada uno de sus estados.

puede haber muchas transiciones en un autómata finito no determinista.

la diferencia entre un autómata determinista de estado finito, un autómata finito de estado no determinista y el teorema básico de la teoría de los autómatas es que NFA se puede convertir en un DFA equivalente que acepta el mismo idioma que la NFA.



ejemplos de un autómata determinista finito y un autómata finito no determinista

hay otra variedad de autómatas de estado finito no determinista que se llaman NFA epsilon, que es una cadena vacía.

las expresiones regulares son específicas y se usan para especificar analizadores léxicos. epsilon también se maneja como una expresión regular, el lenguaje del lenguaje generado por la cadena vacía es un conjunto que contiene cadena vacía.

Los diagramas de transición son DFA generalizados con las siguientes diferencias:

- Los bordes pueden estar etiquetados con un símbolo, un conjunto de símbolos o una definición regular.
- algunos estados de aceptación pueden indicarse como estados de retracción, lo que indica que el lexema no incluye el símbolo que nos llevó al estado de recepción.
- cada estado de aceptación tiene una acción adjunta, que se ejecuta cuando se alcanza ese estado. Normalmente, tal acción devuelve un token y su valor de atributo.

Isaac Rosales Juan de Dios

Lo poco que entendi es que una gramática libre de contexto tiene cuatro elementos básicos:

- Un conjunto de no terminales (V). Los no terminales son variables sintácticas que denotan conjuntos de cadenas de texto que ayudan a definir el lenguaje generado por la gramática.
- Un conjunto de tokens, conocidos como símbolos terminales (?). Los símbolos terminales son los símbolos básicos con los cuales las cadenas de texto son formadas.
- Un conjunto de producciones (P). Las producciones de una gramática especifican la forma en la cual los no terminales pueden ser combinados para

formar cadenas. Cada producción consiste de un no terminal, a los que llamamos lado izquierdo de la producción, luego una flecha, seguida de una secuencia de tokens y/o terminales a los que llamamos lado derecho de la producción.


- Uno de los no terminales es designado como el símbolo de inicio (S) desde el cual comienza la producción.


Las cadenas son derivadas del símbolo de inicio repitiendo y reemplazando no terminales en el lado derecho de la producción por otro no terminal o un terminal.

Context-free Grammars

- A CFG is denoted as $G = (N, T, P, S)$
 - N : Finite set of non-terminals
 - T : Finite set of terminals
 - $S \in N$: The start symbol
 - P : Finite set of productions, each of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$
- Usually, only P is specified and the first production corresponds to that of the start symbol
- Examples

(1)	(2)	(3)	(4)
$E \rightarrow E + E$	$S \rightarrow 0S0$	$S \rightarrow aSb$	$S \rightarrow aB$
$E \rightarrow E * E$	$S \rightarrow 1S1$	$S \rightarrow \epsilon$	$A \rightarrow a$
$E \rightarrow (E)$	$S \rightarrow 0$		$B \rightarrow b$
$E \rightarrow id$	$S \rightarrow 1$		
	$S \rightarrow \epsilon$		





Edwin Antonio Zendejas Mendez

Y con estos videos entendemos que el análisis léxico-sintáctico tiene por objeto reconocer la forma de las sentencias de un lenguaje. Reconocer la forma de una sentencia implica reconocer sus lexemas y estructuras sintácticas.

El resultado del análisis léxico-sintáctico puede ser un error de reconocimiento o una versión de la sentencia reconocida en forma de árbol de sintaxis abstracta (asa).

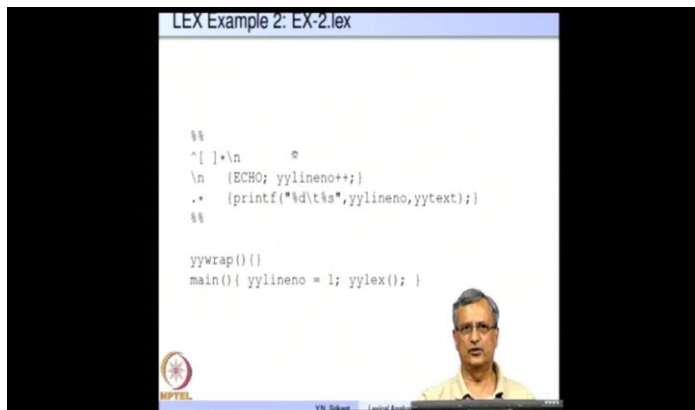
Para reconocer los lexemas de un lenguaje usaremos expresiones regulares y para reconocer estructuras sintácticas se usaran gramáticas independientes de contexto

Una gramática es un conjunto de reglas.

Cada regla es de la forma genérica: cabeza :

cuerpo1 | cuerpo2 | ... | cuerpoN siendo $N \geq 1$

La cabeza de la regla es un símbolo llamado no terminal que representa una estructura sintáctica. El cuerpo de la regla está compuesto por símbolos terminales (lexemas) y no terminales. La composición de estos símbolos se consigue haciendo uso de alternativas, iteraciones y yuxtaposiciones.



Semana 6

MARTINEZ CORTES CRISTOPHER HERNAN

un autómata de empuje hacia abajo M tiene un conjunto finito de afirma Q , tiene un alfabeto σ de entrada, tiene un alfabeto de pila γ , hay un estado inicial Q_0 y un símbolo de la pila inicial Z y varios más componentes.

los autómatas de empuje hacia abajo son tan iguales como en el caso del estado finito no determinista

El análisis sintáctico es el proceso de construir un árbol de parcelas para una oración generada por una gramática determinada.

si no hay restricciones en el idioma y la forma de gramática utilizada analiza para requisitos libres de contexto.

subconjuntos de lenguajes libres de contexto típicamente requieren $O(n^3)$ tiempo.

El análisis de arriba hacia abajo mediante el análisis predictivo rastrea la derivación más a la izquierda de la cadena mientras se construye el árbol de análisis.

comienza desde el símbolo de inicio de la gramática y predice la siguiente producción utilizada en la derivación

Ozalde Alday Viridiana Elizabeth

Una de las definiciones equivalentes de lenguaje libre de contexto emplea autómatas no deterministas: que se dice un lenguaje es libre de contexto si puede ser aceptado por un autómata.

Un lenguaje puede ser modelado como un conjunto de todas las secuencias de terminales aceptadas por la gramática. Este modelo ayuda a entender las operaciones de conjuntos sobre lenguajes.

- Ejemplo: Considere $L_{pal} = \{w \in \Sigma^* : w = w^R\}$. Por ejemplo, $oso \in L_{pal}$, $anitalavalatina \in L_{pal}$.
- Sea $\Sigma = \{0, 1\}$ y supongamos que L_{pal} es regular.
- Sea n dada por el pumping lemma. Entonces $0^n 1 0^n \in L_{pal}$. Al leer 0^n el FA debe de entrar a un ciclo. Si quitamos el ciclo entonces llegamos a una contradicción.

Rosales Juan de Dios Isaac

Los autómatas pushdown son no deterministas, lo que significa que en una descripción instantánea dada puede haber varios pasos posibles. Cualquiera de estos pasos se puede elegir en un cálculo. Con la definición anterior en cada paso, siempre aparece un solo símbolo (parte superior de la pila), reemplazándolo con tantos símbolos como sea necesario. Como consecuencia, no se define ningún paso cuando la pila está vacía.

Zendejas Mendez Edwin

Los cálculos del autómata de empuje son secuencias de pasos. El cálculo comienza en el estado inicial con el símbolo de pila inicial en la pila y una cadena en la cinta de entrada, por lo tanto con la descripción inicial. Hay dos modos de aceptar. El autómata pushdown acepta por estado final, lo que significa que después de leer su entrada, el autómata alcanza un estado de aceptación (in), o acepta por pila vacía (), lo que significa que después de leer su entrada, el autómata vacía su pila.

SEMANA 7

MARTINEZ CORTES CRISTOPHER HERNAN

Los analizadores LR también se generan automáticamente utilizando generadores de analizadores.

Las gramáticas LR son un subconjunto de CFGs para el que se pueden construir analizadores LR.

Las gramáticas LR(1) se pueden escribir con bastante facilidad para prácticamente todas las construcciones de lenguaje de programación para las que se pueden escribir CFG.

la tabla de análisis sintáctico LR y la tabla del analizador se ajustan a otra caja que contiene una pila y una rutina del conductor, así que la rutina del controlador del ciervo y la tabla de análisis juntos hacen que el analizador tome el programa como entrada y entrega como salida posiblemente una sintaxis árbol o algo mas.

iniciando la configuración de los analizadores: (s0, a1a2...ans). donde, s0 es el estado inicial del analizador, y a1a2...an es la cadena que se va a analizar.

ROSALES JUAN DE DIOS ISAAC

Un analizador LR detectará un error cuando consulte la tabla de acciones de análisis y encuentre una entrada en blanco o de error.

Los errores nunca se detectan consultando la tabla goto. Un analizador LR detectará un error tan pronto como haya no hay continuación válida para la parte de la entrada escaneada hasta ahora. Un analizador LR canónico no hará incluso una única reducción antes de anunciar el error. Los analizadores SLR y LALR pueden realizar varias reducciones antes de detectar un error, pero nunca cambiarán un símbolo de entrada erróneo a la pila.

Error Recovery in LR Parsers - Parser Operation

- When the parser encounters an error, it scans the stack to find the topmost state containing an *error item* of the form $A \rightarrow .error \alpha$
- The parser then shifts a token *error* as though it occurred in the input
- If $\alpha = \epsilon$, reduces by $A \rightarrow \epsilon$ and invokes the error message routine associated with it
- If $\alpha \neq \epsilon$, discards input symbols until it finds a symbol with which the parser can proceed
- Reduction by $A \rightarrow .error \alpha$ happens at the appropriate time
Example: If the error production is $A \rightarrow .error ;$, then the parser skips input symbols until ';' is found, performs reduction by $A \rightarrow .error ;$, and proceeds as above
- Error recovery is not perfect and parser may abort on end of input

Zendejas Mendez Edwin

He entendido que el elemento LR(1) es un par formado por un elemento LR(0) y el símbolo de anticipación. Este elemento tiene la forma general $[A \rightarrow \alpha.B\beta, a]$, donde $A \rightarrow \alpha B \beta$ es una producción y a es un terminal o el $\$$. Precisamente el 1 de LR(1) representa la longitud de a . Es de destacar que todos los analizadores sintácticos ascendentes reconocen las sentencias de la gramática de la misma forma y solo cambia la forma de obtener la tabla de análisis, pero una vez obtenida, el tratamiento de la pila y la entrada para realizar el reconocimiento de una sentencia es el mismo.

TELCEL 8:55

LR(1) Grammar - Example 2

Grammar $S' \rightarrow S$ $S \rightarrow L=R \mid R$ $L \rightarrow *R \mid id$ $R \rightarrow L$	State 2 $S \rightarrow L.=R, \$$ $R \rightarrow L., \$$	State 6 $S \rightarrow L.=R, \$$ $R \rightarrow .L, \$$ $L \rightarrow .*R, \$$ $L \rightarrow .id, \$$	State 10 $R \rightarrow L., \$$
State 0 $S' \rightarrow .S, \$$ $S \rightarrow .L=R, \$$ $S \rightarrow .R, \$$ $L \rightarrow .*R, =$ $L \rightarrow .id, =$ $R \rightarrow .L, \$$ $L \rightarrow .*R, \$$ $L \rightarrow .id, \$$	State 3 $S \rightarrow R., \$$	State 7 $L \rightarrow *R., =/\$$	State 11 $L \rightarrow *.R, \$$ $R \rightarrow .L, \$$ $L \rightarrow .*R, \$$ $L \rightarrow .id, \$$
State 1 $S' \rightarrow S., \$$	State 4 $L \rightarrow *.R, =/\$$ $R \rightarrow .L, =/\$$ $L \rightarrow ^{(*)}.R, =/\$$ $L \rightarrow .id, =/\$$	State 8 $R \rightarrow L., =/\$$	State 12 $L \rightarrow id., \$$
	State 5 $L \rightarrow id., =/\$$	State 9 $S \rightarrow L=R., \$$	State 13 $L \rightarrow *R., \$$

Grammar is not SLR(1), but is LR(1)

OZALDE ALDAY VIRIDIANA ELIZABETH

ANÁLISIS SINTÁCTICO ASCENDENTE LR

- **L:** examen de la entrada de izquierda a derecha
- **R:** "rightmost derivation", corresponde al orden inverso
- **(k):** número de símbolos de entrada leídos por vez
- Es un método de desplazamiento y reducción sin retroceso
- Puede detectar un error tan pronto como sea posible en un examen de izquierda a derecha

La tabla de análisis sintáctico tiene 2 partes:

Acción: indica una acción del analizador

Ir_a: indica transiciones de estado

El programa que maneja el analizador LR se comporta de la siguiente manera.

Determina el estado al tope de la pila y el símbolo en la entrada.

Consulta la tabla Acción para el estado del tope de la pila y la entrada. La tabla tiene 4 valores: desplazar, reducir, aceptar, error.

La función `ir_a` toma un estado y un símbolo gramatical como argumentos y produce estados

"La función `ir_a` de una tabla de análisis sintáctico construida a partir de una gramática G es la función de transiciones de un autómata finito determinista que reconoce los prefijos viables de G ."

El estado inicial de este AFD es el estado puesto inicialmente en el tope de la pila. Las posibles acciones de la tabla acción son:

Desplazar: el analizador ejecuta un movimiento de desplazamiento donde desplaza a la pila el símbolo de entrada

Reducir

Aceptar: finaliza el análisis sintáctico

Error: llama a una función de recuperación de errores

SEMANA 8

MARTINEZ CORTES CRISTOPHER HERNAN

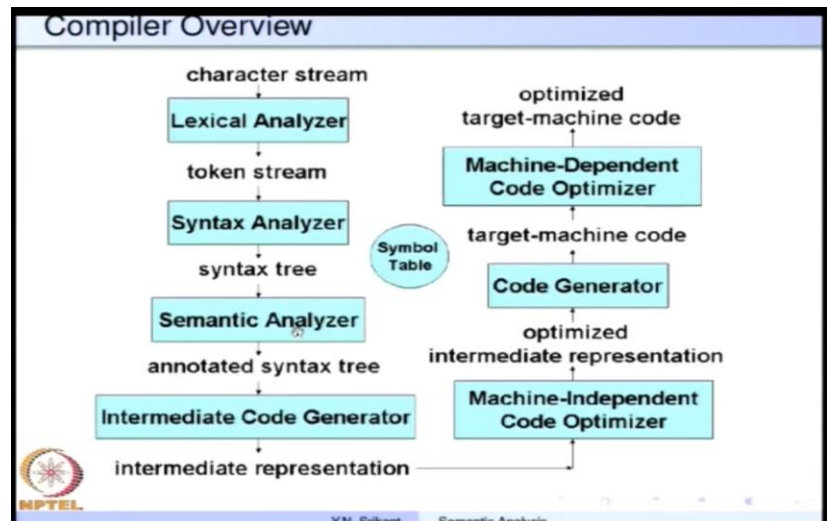
La consistencia semántica que no se puede manejar en la etapa de análisis se maneja aquí.

los analizadores no pueden manejar características sensibles al contexto de los lenguajes de programación

Estos son semánticos estáticos de lenguajes de programación y pueden ser verificados por el analizador semántico:

- las variables se declaran antes de usarse
- los tipos coinciden en ambos lados de las asignaciones
- tipos de parámetros y coincidencia de números en la declaración y el uso

Los compiladores solo pueden generar código para verificar lenguajes de semántica dinámica y tiempo de ejecución



ROSALES JUAN DE DIOS
ISAAC

El análisis semántico puede realizarse cuando está hecho

todo el análisis sintáctico y por tanto se ha construido un árbol de sintaxis abstracta, entonces esta tarea de análisis semántico sería mucho más fácil, puesto que consistiría en la especificación del orden para el recorrido de dicho árbol, junto con los cálculos a realizar en cada nodo del recorrido.

Cada atributo es una variable que representa una propiedad de un símbolo (terminal o no terminal). Ejemplos de atributos que podemos necesitar son tipo (entero, real, etc.) o valor (25, 25.4, etc.).

ZENDEJAS MENDEZ EDWIN

Debemos ser conscientes de que el objetivo, por razones de eficiencia, es realizar todas las operaciones en una sola pasada (todas las fases, incluida la generación de código, suceden en una única lectura del código fuente). Para ello vamos a realizar las comprobaciones semánticas a la vez que se valida la estructura sintáctica, de tal forma que se generará un nodo si este es correcto tanto sintácticamente como semánticamente. Es por ello que se añaden a las reglas gramaticales el cálculo necesario para establecer estos controles o comprobaciones semánticas

OZALDE ALDAY VIRIDIANA ELIZABETH

una parte importante del análisis es la comprobación pues así se puede corroborar que las operaciones se estén ejecutando correctamente. 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.

SEMANA 11 ACTUALIZACIÓN

OZALDE ALDAY VIRIDIANA ELIZABETH

Una gramática libre de contexto, básicamente, consiste en un conjunto finito de reglas gramaticales. Con el fin de definir reglas de la gramática (producciones), asumimos que tenemos dos tipos de símbolos: los terminales, que son los símbolos del alfabeto, y los no terminales que se comportan como las variables.

Una gramática libre de contexto es una cuádrupla $G = (V, \Sigma, P, S)$, donde

- V es un conjunto finito de símbolos denominado vocabulario (o conjunto de símbolos gramaticales).
- $\Sigma \subseteq V$ es el conjunto de símbolos terminales.
- $S \in (V - \Sigma)$ es el símbolo que denota el inicio.
- $P \subseteq (V - \Sigma) \times V$ es un conjunto finito de producciones.

El lenguaje definido por una gramática independiente de contexto se denomina lenguaje independiente del contexto.

Las producciones de una gramática se utilizan para derivar cadenas. En este proceso, las producciones se utilizan como reglas de reescritura.

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:

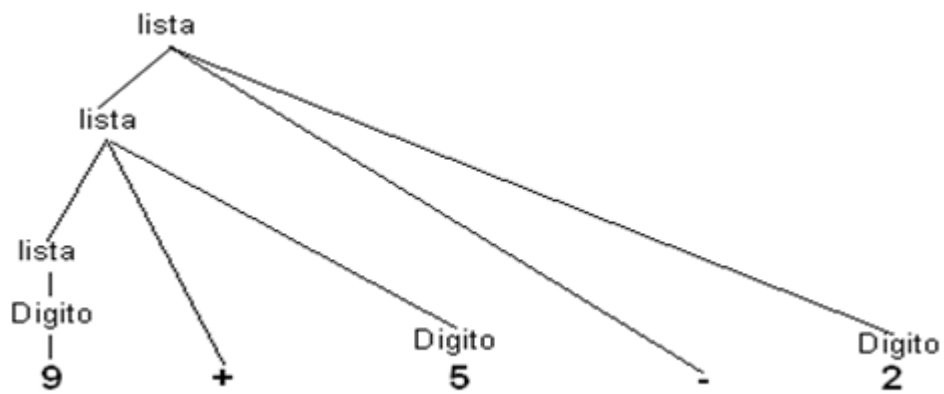
lista \rightarrow lista + dígito

lista \rightarrow lista - dígito

lista \rightarrow dígito

dígito \rightarrow 0|1|2|3|4|5|6|7|8|9

Arbol de analisis sintactico:



Isaac Rosales Juan de Dios

Se hizo un GIC que genere un número binario

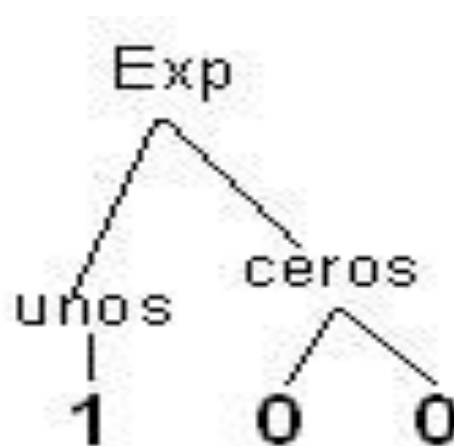
Exp \rightarrow Exp bin | bin

bin \rightarrow 0 | 1

Ó con una sola producción:

Exp \rightarrow Exp 0 | Exp 1 | 0 | 1

Prueba:



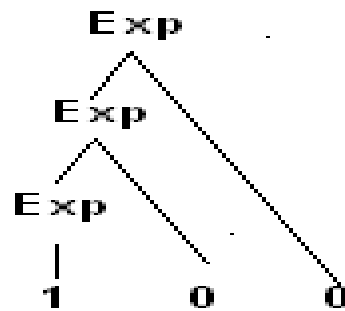
Aunque esta producción puede generar expresiones como 0000, para evitar errores como este:

dig \rightarrow 1 Exp | 1

$\text{Exp} \rightarrow \text{Exp } 0 \mid \text{Exp } 1 \mid 0 \mid 1$

Nota 1: El símbolo inicial siempre debe estar en la primera producción de la gramática.

Nota 2: Expresiones de tipo, $\text{Exp} \rightarrow \text{Exp } 0 \mid 1$, genera potencias de 10, ejemplo



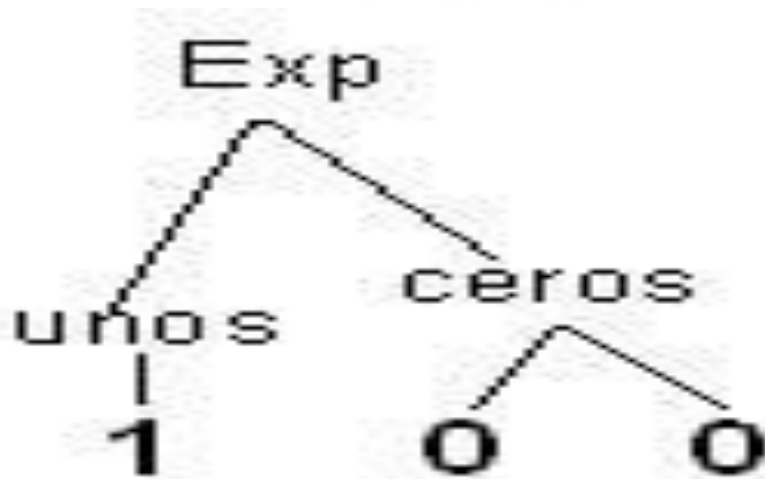
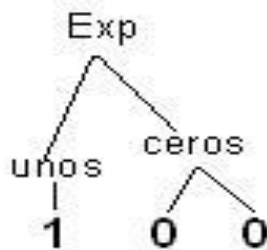
[Zendejas Mendez Edwin Antonio](#)

en este ejemplo se creó una gramática que genere un conjunto de 1 seguido de un conjunto de 0, donde el número 1 debe ser impar y el número de 0 debe ser par.

$\text{Exp} \rightarrow \text{unos } \text{ceros}$

$\text{ceros} \rightarrow \text{ceros } 00 \mid 00$

$\text{unos} \rightarrow \text{unos } 11 \mid 1$



Ejemplo 8: ¿Cuál es el lenguaje de la siguiente producción?

Pal \rightarrow Pal letras | letras

Letras \rightarrow a | b | c | d | e | f | g | ... | z

R/ Es una GIC que genera palabras escritas en minúsculas.

Ejemplo 9: Hacer una GIC que genere una frase cuya letra inicial de cada palabra sea mayúscula.

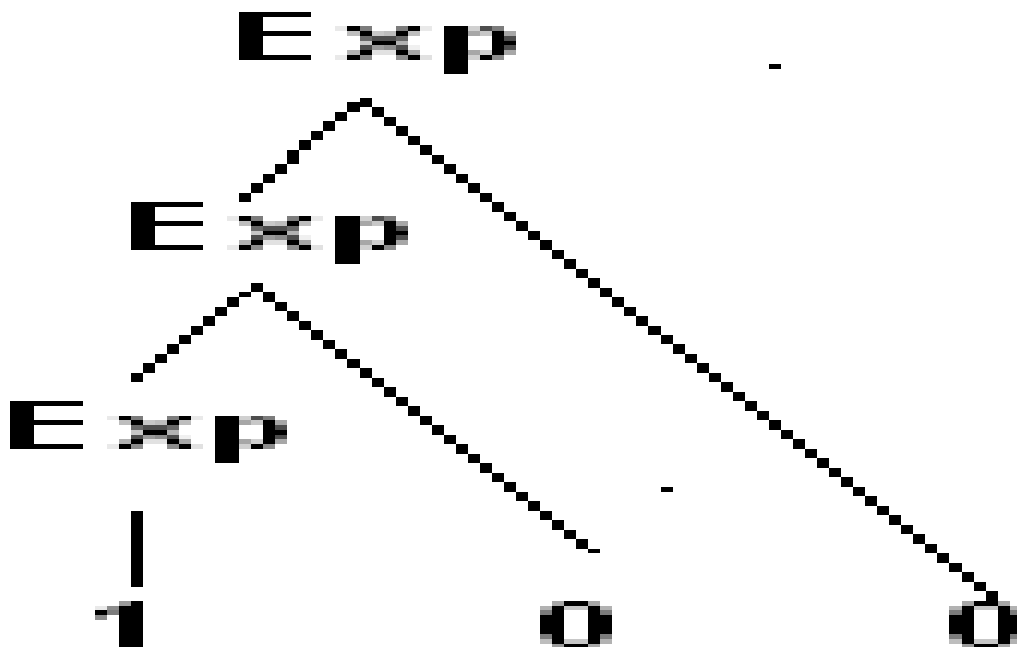
Frase \rightarrow Frase Exp pal | pal

Exp \rightarrow " "

Pal \rightarrow may | pal min

min \rightarrow a | b | c | d | ... | z

may \rightarrow A | B | C | D | ... | Z



SEMANA 12

Rosales Juan de Dios Isaac

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 Gramática libre de contexto

$$A \rightarrow \alpha$$

donde A es un solo símbolo no terminal y α es una cadena de terminales y / o no terminales (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 del lado izquierdo siempre se puede reemplazar por el lado derecho. Esto es lo que lo distingue de una gramática sensible al contexto.

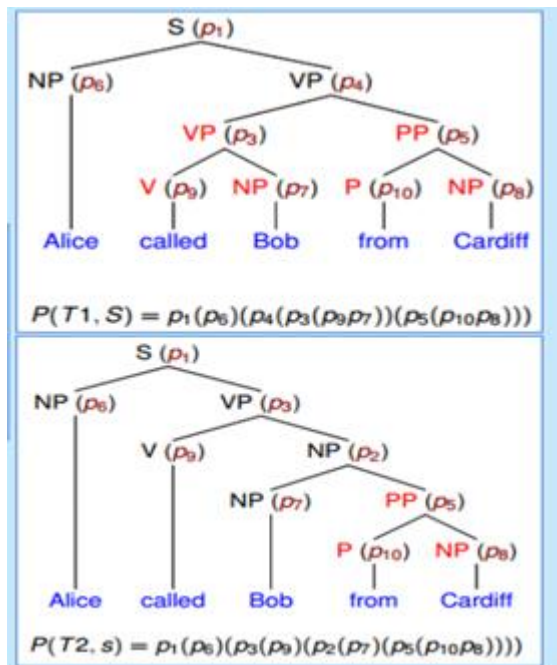
$S \rightarrow NP VP$ (p_1)	$NP \rightarrow \text{Alice}$ (p_6)
$NP \rightarrow NP PP$ (p_2)	$NP \rightarrow \text{Bob}$ (p_7)
$VP \rightarrow V NP$ (p_3)	$NP \rightarrow \text{Cardiff}$ (p_8)
$VP \rightarrow VP PP$ (p_4)	$V \rightarrow \text{called}$ (p_9)
$PP \rightarrow P NP$ (p_5)	$P \rightarrow \text{from}$ (p_{10})

OZALDE ALDAY VIRIDIANA ELIZABETH

Argumenta cada regla con una probabilidad condicionada

- $A \rightarrow \alpha$ (p) $P(A \rightarrow \alpha)$
- p representa la probabilidad de que dado un no terminal A pueda ser expandido con la secuencia α .
- La probabilidad del árbol de derivación es el producto de las probabilidades de las reglas usadas en su construcción.

A continuación, se continúa con el ejemplo de mi compañero:



ZENDEJAS MENDEZ EDWIN ANTONIO

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 de la imagen,

reemplaza con. 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 símbolos terminales que pueden derivarse, mediante aplicaciones de reglas repetidas, de algún símbolo no terminal particular ("símbolo de inicio").

El ejemplo canónico de una gramática libre de contexto es la coincidencia de paréntesis, que es representativa del caso general. Hay dos símbolos terminales "(" y ")" y un símbolo no terminal S. Las reglas de producción son

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow ()$$

La primera regla permite que el símbolo S se multiplique; la segunda regla permite que el símbolo S quede entre paréntesis coincidentes; y la tercera regla termina la recursividad.

MARTINEZ CORTES CRISTOPHER HERNAN

Gramáticas independientes del contexto (context-free grammars): usan reglas que predicen las palabras que pueden posiblemente seguir a la última palabra reconocida, reduciendo el número de palabras candidatas a evaluar para reconocer la siguiente pronunciación del speaker (hablante o fichero de voz). Las reglas están formadas por dos tipos de símbolos: palabras y directrices.

Ejemplo: =ALT(SEQ("Jesús", "Moreno"), "Miguel Alonso")

que significa algo como "lo que en la categoría que se puede dar como válido al reconocer el habla es una de las siguientes alternativas: la secuencia Jesús y detrás Moreno o Miguel Alonso".

SEMANA 13

OZALDE ALDAY VIRIDIANA ELIZABETH

Los diseñadores de lenguajes de programación introducen tipos de sistemas para que puedan especificar el comportamiento del programa a un nivel más preciso de lo que es posible en una gramática libre de contexto. El sistema de tipos crea un segundo vocabulario para describir tanto la forma como el comportamiento de programas válidos.

Por ejemplo, un número entero podría ser cualquier número entero

i en el rango de $-2^{31} \leq i < 2^{31}$

MARTINEZ CORTES CRISTOPHER HERNAN

Analizando un programa desde la perspectiva de su tipo de sistema produce información que no se puede obtener utilizando las técnicas de escaneo y análisis. En un compilador, esta información se utiliza normalmente para tres propósitos distintos: seguridad, expresividad y eficiencia en tiempo de ejecución.

Type of			Code
a	b	a+b	
integer	integer	integer	iADD $r_a, r_b \Rightarrow r_{a+b}$
integer	real	real	i2f $r_a \Rightarrow r_{af}$ fADD $r_{af}, r_b \Rightarrow r_{af+b}$
integer	double	double	i2d $r_a \Rightarrow r_{ad}$ dADD $r_{ad}, r_b \Rightarrow r_{ad+b}$
real	real	real	fADD $r_a, r_b \Rightarrow r_{a+b}$
real	double	double	r2d $r_a \Rightarrow r_{ad}$ dADD $r_{ad}, r_b \Rightarrow r_{ad+b}$
double	double	double	dADD $r_a, r_b \Rightarrow r_{a+b}$

Rosales Juan de Dios Isaac

Un sistema de tipos para un lenguaje moderno típico tiene cuatro componentes principales:

conjunto de tipos base o tipos integrados; reglas para construir nuevos tipos a partir de los tipos existentes; un método para determinar si dos tipos son equivalentes o compatibles; y reglas para inferir el tipo de cada expresión del idioma de origen.

Muchos lenguajes también incluyen reglas para la conversión implícita de valores de un tipo a otro según el contexto. Esta sección describe cada uno de estos en

más detalles, con ejemplos de lenguajes de programación populares.

ZENDEJAS MENDEZ EDWIN ANTONIO

Lo que dice en el libro es que en la mayoría de los lenguajes de programación incluyen tipos base para algunos, si no todos, los siguientes tipos de datos: números, caracteres y valores booleanos. Estos tipos son soportados directamente por la mayoría de procesadores. Los números suelen venir en varias formas, como enteros y números de coma flotante. Agregar idiomas individuales otros tipos de base. Lisp incluye tanto un tipo de número racional como un recursivo tipo contras.

Los números racionales son, esencialmente, pares de enteros interpretados como ratios. Un contra se define como el valor designado cero o (contras primero resto) donde primero es un objeto, el resto es una desventaja y contras crea una lista a partir de sus argumentos.

SEMANA 14

ZENDEJAS MENDEZ EDWIN ANTONIO

Durante largo tiempo se han utilizado muchas estructuras de datos para implementar ir lineales. Las elecciones que hace un escritor de compiladores

afectan los costos de varias operaciones en su código. Dado que un compilador pasa la mayor parte de su tiempo manipulando la forma del código, estos costos merecen cierta atención. Si bien esta discusión se centra en los códigos de tres direcciones, la mayoría de los puntos se aplican igualmente al código de máquina de pila (o cualquier otra forma lineal).

Los códigos de tres direcciones a menudo se implementan como un conjunto de cuádruples. Cada cuádruple se representa con cuatro campos: un operador, dos operandos (o fuentes) y un destino.

```

t1 ← 2
t2 ← b
t3 ← t1 × t2
t4 ← a
t5 ← t4 - t3

```

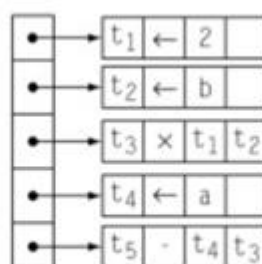
ROSALES JUAN DE DIOS ISAAC

En el ejemplo de mi amigo yo lo voy a explicar ahí muestra tres esquemas diferentes para implementar el código de tres direcciones para $a - 2 \times b$, repetido en el margen. El esquema más simple, en la imagen, usa una matriz corta para representar cada bloque básico.

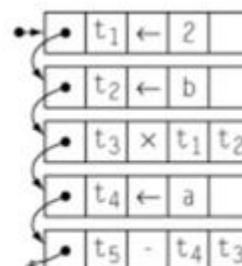
A menudo, el escritor del compilador coloca la matriz dentro de un nodo en el archivo cfg. (Esta puede ser la forma más común de ir híbrido). El esquema de la imagen b usa una matriz de punteros para agrupar cuádruplicados en un bloque; la matriz de punteros puede estar contenida en un nodo cfg. El esquema final, en la imagen c, vincula los cuádruples para formar una lista. Requiere menos almacenamiento en el nodo cfg, a costa de restringir los accesos a recorridos secuenciales.

Target	Op	Arg ₁	Arg ₂
t ₁	←	2	
t ₂	←	b	
t ₃	×	t ₁	t ₂
t ₄	←	a	
t ₅	-	t ₄	t ₃

(a) Simple Array



(b) Array of Pointers



(c) Linked List

MARTINEZ CORTES CRISTOPHER HERNAN

Como dice ahí en el libro un IR es una instrucción virtual puede calcular un valor, que debe mantenerse en un registro y luego ser utilizado por otras instrucciones como operandos.² En una implementación simplista, se podría asignar un registro distinto para contener cada valor. Esto conduce a un uso excesivo de los recursos de almacenamiento. Por el contrario, se podría aprovechar el hecho de que los valores no necesitan mantenerse todo el tiempo, porque tienen una vida útil limitada. Los valores que tienen vidas útiles que no se superponen pueden compartir el mismo registro para ahorrar área de silicio. La tarea de mapear valores a registros para maximizar el intercambio se denomina vinculación de registros.

Basic Block Liveness Analysis

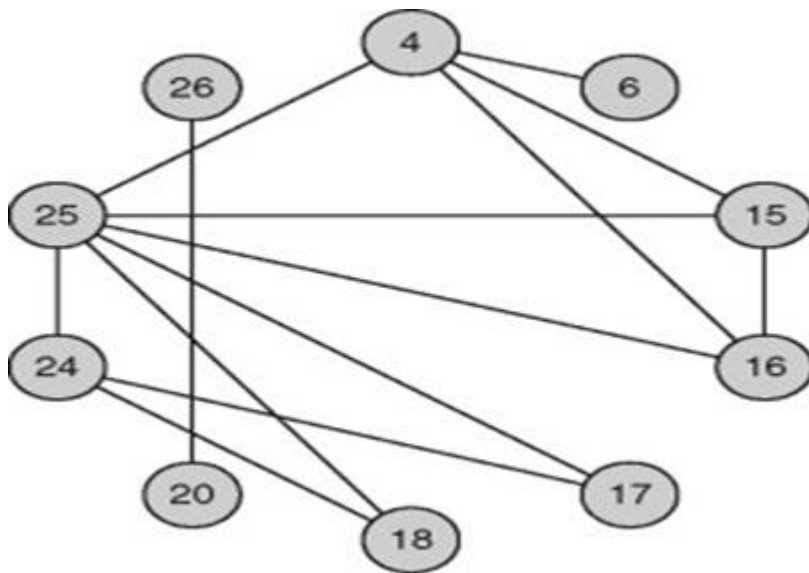
```
algorithm liveBB ( $S : V \mapsto \mathbf{Z}$ , liveOut :  $V[]$ ) returns  $\mathbf{Z} \mapsto V[]$ 
1. var    Live:  $\mathbf{Z} \mapsto V[]$ ;
2. var    l :  $\mathbf{Z}$ ;
3. l = | range S |; Live(l) = liveOut;
4. foreach ( $s \in [l-1..0]$ ) begin
5.   Live(s) = Live(s + 1) –  $S^{-1}(s)$ ;
6.   foreach ( $v \in S^{-1}(s)$ )
7.     Live(s) = Live(s)  $\cup \{v.src1\} \cup \{v.src2\}$ ;
8. end foreach
9. return Live;
```

OZALDE ALDAY VIRIDIANA ELIZABETH

En un gráfico de interferencia, el problema de vinculación de registros se reduce a la asignación de cada nodo a un número de registro, mientras se asegura que a dos nodos conectados por un borde se les asignen números de registro diferentes. Esto es equivalente al problema clásico de coloración de gráficos, si cada número de registro se trata como un color.

Un ejemplo.

La imagen de abajo muestra el gráfico de interferencia construido a partir de la información de vida. Por ejemplo, para la instrucción 20, está programada en el paso de control 3. El conjunto en vivo para el paso 4 es {20, 26}. Por lo tanto, se crea un borde del gráfico de interferencia entre 20 y 26. Este proceso se repite para cada instrucción.

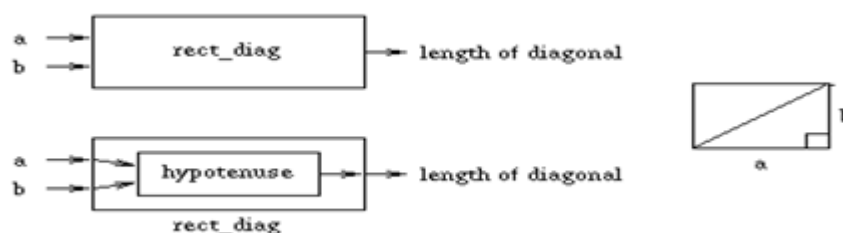


SEMANA 15

Rosales Juan de Dios Isaac

La abstracción procedimental es cuando escribimos secciones de código (llamadas "procedimientos" o en Java, "métodos estáticos") que se generalizan al tener parámetros variables. La idea es que tengamos código que pueda hacer frente a una variedad de situaciones diferentes, dependiendo de cómo se establezcan sus parámetros cuando se llama. Hace que un programa sea más corto y más fácil de entender, y ayuda a depurar, ya que cuando corregimos un procedimiento, corregimos el programa para todos los casos en los que se llama al procedimiento.

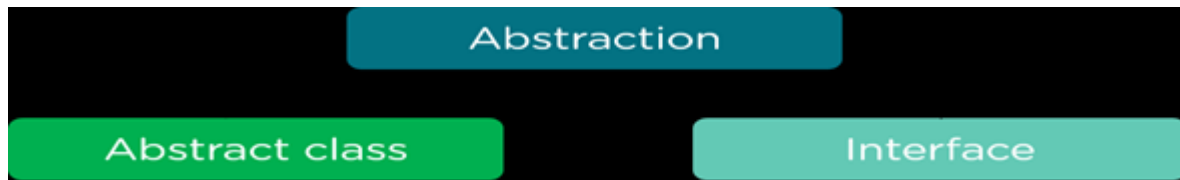
Un enfoque "de abajo hacia arriba" para introducir la abstracción procedimental es observar los casos en los que tenemos un código que es similar a otras piezas, o piezas que siguen un patrón común, y ver si hay una manera de convertirlas en llamadas a un procedimiento. Un enfoque "de arriba hacia abajo" es pensar en la operación generalizada que queremos hacer, escribir código que use llamadas al procedimiento para esta operación y luego escribir código para el procedimiento.



OZALDE ALDAY VIRIDIANA ELIZABETH

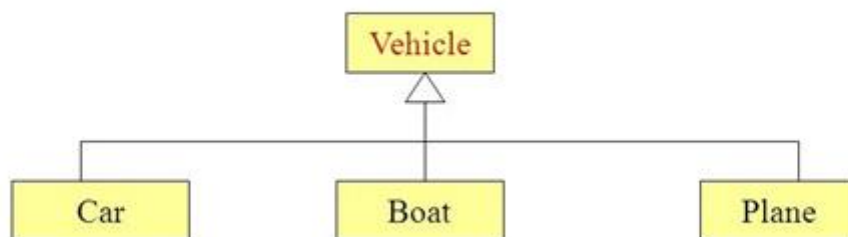
El enfoque de arriba hacia abajo conduce a la idea de escribir código según especificaciones. Decimos precisamente lo que queremos que haga un

procedimiento en términos de sus argumentos, y esperamos que haga precisamente eso, nada más y nada menos. Entonces, no necesitamos tener conocimiento de lo que hay dentro del procedimiento para hacer un uso exitoso de él. Cuando escribimos el procedimiento, no necesitamos tener conocimiento del código que va a hacer uso del procedimiento. Esto nos ayuda a dividir los programas en pequeñas partes manejables, quizás manejadas por diferentes programadores.



ZENDEJAS MENDEZ EDWIN ANTONIO

La abstracción procedimental se trata de generalizar haciendo alguna acción. Se llama a un procedimiento, realiza la acción y luego finaliza. El procedimiento puede devolver algún valor, o puede cambiar el estado de algunos de sus argumentos, o puede causar que suceda algún evento externo (entrada / salida de algún tipo), o puede causar alguna combinación de estos.



MARTINEZ CORTES CRISTOPHER HERNAN

Entonces, se sabe que la única forma en que diferentes partes de un programa pueden interactuar es a través de llamadas a procedimientos que tienen un efecto cuidadosamente definido, es más fácil para nosotros depurar y modificar el programa con menos posibilidades de que cambiar una parte tenga un efecto inesperado en otra parte.

