Pregunta 1

Definir el concepto de autómata con pila, y explicar cuando una palabra es reconocida por un autómata con pila

Un autómata con pila es una estructura $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ donde:

(1) K es un conjunto finito y no vacío de estados,

(2) Σ es el alfabeto de entrada,

(3) Γ es el alfabeto de la pila,

(4) $q_0 \in K$ es el estado inicial,

(5) $F \subseteq K$ es el conjunto de estados aceptadores,

(6) Δ es un conjunto finito de elementos de la forma $((p, a, b), (q, x))$ donde $p, q \in K$, $a \in \Sigma \cup \{\lambda\}$, $b \in \Gamma \cup \{\lambda\}$ y $x \in \Gamma^*$.

Pregunta 1

Definir el concepto de autómata con pila, y explicar cuando una palabra es reconocida por un autómata con pila

Si $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ es un autómata con pila, decimos que una palabra $x \in \Sigma^*$ es reconocida por M, si existe un cómputo para la entrada x que termina en un estado aceptador, lee toda la palabra de entrada y deja la pila vacía al final del cómputo.

Pregunta 2

Definir el concepto de autómata con pila determinista

Sea $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ un autómata con pila.

Dos transiciones $((p_1, a_1, b_1), (q_1, \alpha_1))$ y $((p_2, a_2, b_2), (q_2, \alpha_2))$ de M son compatibles, si se cumplen las tres siguientes condiciones:

(1) $p_1 = p_2$.

(2) $a_1 = a_2$ ó $a_1 = \lambda$ ó $a_2 = \lambda$.

(3) $b_1 = b_2$ ó $b_1 = \lambda$ ó $b_2 = \lambda$.

El que dos transiciones sean compatibles significa que las dos se pueden aplicar en un mismo paso de cómputo del autómata

Decimos entonces que M es determinista, si no tiene dos transiciones compatibles distintas.

Pregunta 3

Definir el concepto de gramática incontextual y explicar la relación existente entre las gramáticas incontextuales y los autómatas con pila.

Una gramática incontextual es una estructura $G = (V, \Sigma, P, S)$ donde:

(1) V es un alfabeto, a cuyos elementos se les llama variables.

(2) Σ es un alfabeto disjunto de V, a cuyos elementos se les llama terminales.

(3) P es un subconjunto finito de $V \times (V \cup \Sigma)^*$, a cuyos elementos se les llama producciones (o reglas).

(4) $S \in V$ es la variable inicial.

Los autómatas con pila son estructuras equivalentes a las gramáticas incontextuales, es decir, se tiene que para todo lenguaje L, existe una autómata con pila M tal que $L = L(M)$ si y sólo si existe una gramática incontextual G tal que $L = L(G)$.

Pregunta 4

Definir el concepto de derivación en una gramática incontextual

Si $G = (V, \Sigma, P, S)$ es una gramática incontextual y $u, v \in (V \cup \Sigma)^*$, decimos que hay una derivación de un paso de la palabra u a la palabra v, en símbolos $u \Rightarrow_G v$, si obtenemos v a partir de u aplicando una producción de P. Y decimos que hay una derivación de la palabra u a la palabra v, en símbolos $u \Rightarrow_G^* v$, si obtenemos v a partir de u aplicando un número finito de veces las producciones de P.

Definimos entonces $L(G) = \{x \in \Sigma^* : S \Rightarrow_G^* x\}$.

Pregunta 5

Mostrar el algoritmo visto en clase para construir un autómata con pila equivalente a una gramática incontextual

Si $G = (V, \Sigma, P, S)$ es una gramática incontextual, definimos el autómata con pila $M = (\{q_0, f\}, \Sigma, V \cup \Sigma, \Delta, q_0, \{f\})$ donde Δ consta de las siguientes transiciones:

(1) $((q_0, \lambda, \lambda), (f, S))$.

(2) $((f, \lambda, A), (f, x))$ para cada producción $A \to x$ de P.

(3) $((f, a, a), (f, \lambda))$ para cada símbolo terminal a de Σ.

Se puede demostrar entonces que $L(G) = L(M)$, es decir, G y M son equivalentes.

Pregunta 6

Definir los conceptos de árbol de derivación y gramática ambigua, y explicar por qué las gramáticas ambiguas no deben utilizarse en el diseño de compiladores

Sea $G = (V, \Sigma, P, S)$ una gramática incontextual.

(a) A toda derivación $S \Rightarrow_G^* x$ en G le asociamos el siguiente árbol:

(1) La raíz del árbol es S.

(2) Cada nodo interno del árbol corresponde a una variable de la gramática que es sustituida en el proceso de derivación.

(3) La palabra formada por las hojas en sentido de izquierda a derecha es x.

A dicho árbol se le llama árbol de derivación de x.

(b) Decimos entonces que la gramática G es ambigua, si hay una palabra $x \in L(G)$ que tiene más de un árbol de derivación asociado.

Pregunta 6

Definir los conceptos de árbol de derivación y gramática ambigua, y explicar por qué las gramáticas ambiguas no deben utilizarse en el diseño de compiladores

Las gramáticas ambiguas no se deben utilizar en el diseño de compiladores, porque si se utilizaran habría instrucciones en los lenguajes de programación que se podrían interpretar de diferentes maneras y dar resultados de ejecución diferentes.

Pregunta 7

Demostrar que es ambigua la gramática incontextual definida por las dos siguientes producciones:

$I \longrightarrow \underline{if}\,(E)\,I\,\underline{else}\,I$

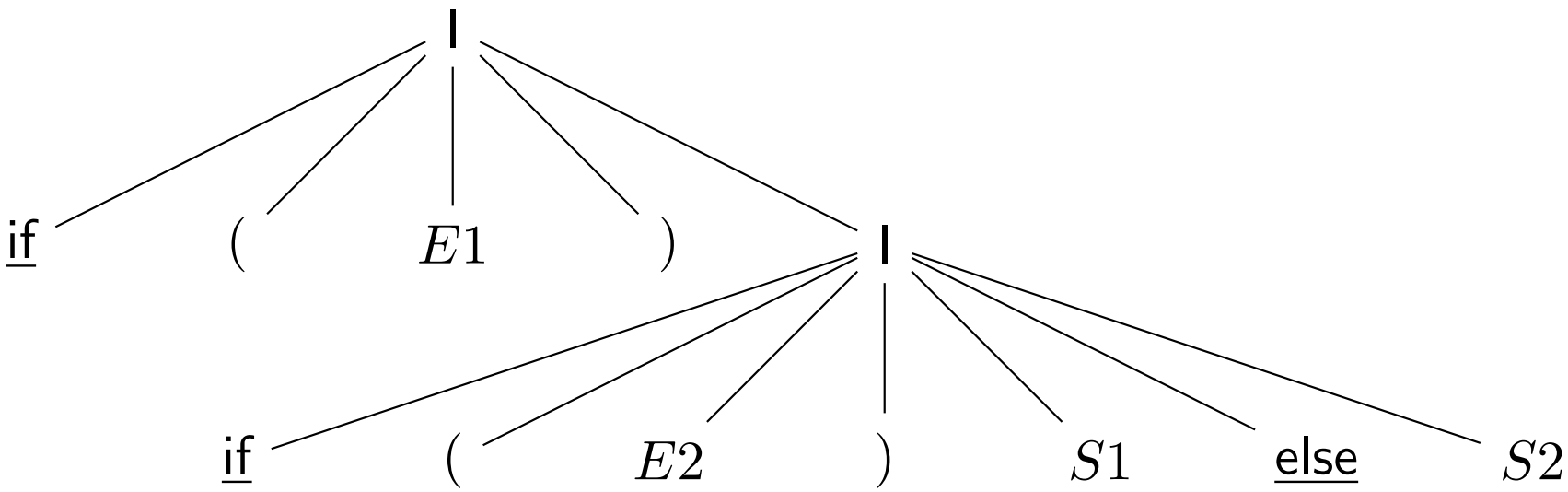
$I \longrightarrow \underline{if}\,(E)\,I$

La gramática es ambigua, porque una instrucción de la forma

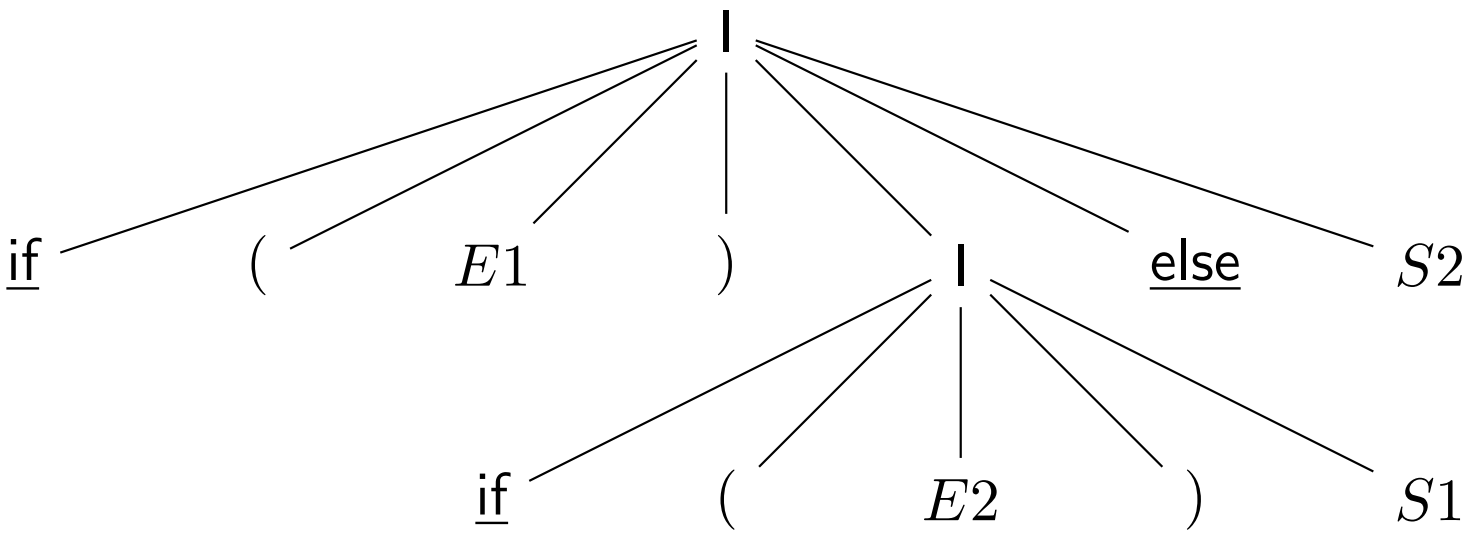
$\underline{if}\,(E1)\,\underline{if}\,(E2)\,S1\,\underline{else}\,S2$

(es decir, una instrucción con dos ifs y un solo else) tiene los dos siguientes árboles de derivación:

Pregunta 7



Pregunta 7



Pregunta 8

Explicar en qué consisten las tres fases del diseño de un compilador y en qué fases se utilizan los autómatas deterministas, los autómatas con pila y las gramáticas incontextuales

El diseño de un compilador consta de las siguientes fases:

(1) Análisis léxico.

(2) Análisis sintáctico.

(3) Análisis semántico.

En la fase del análisis léxico, el compilador agrupa los símbolos que va leyendo del programa fuente en categorías sintácticas, que son secuencias de caracteres que tienen un significado, y pasa dicha información al analizador sintáctico.

Pregunta 8

Explicar en qué consisten las tres fases del diseño de un compilador y en qué fases se utilizan los autómatas deterministas, los autómatas con pila y las gramáticas incontextuales

El analizador sintáctico determina entonces si el programa fuente está escrito correctamente con respecto a las reglas del lenguaje de programación. Si es así, proporciona como representación del proceso de análisis realizado el árbol de derivación de la palabra que recibe como entrada.

Y en la fase del análisis semántico, el compilador controla los aspectos semánticos, especialmente la compatibilidad de tipos en las asignaciones y en los parámetros de llamadas a funciones, y si no hay tales errores el compilador traduce el programa fuente a código máquina.

Pregunta 8

Explicar en qué consisten las tres fases del diseño de un compilador y en qué fases se utilizan los autómatas deterministas, los autómatas con pila y las gramáticas incontextuales

Para diseñar el analizador léxico, se utilizan los autómatas deterministas. Y para diseñar el analizador sintáctico y el analizador semántico, se utilizan las gramáticas incontextuales y los autómatas con pila.

Pregunta 9

Explicar el interés que tiene el diseño de compiladores para la programación

Sin los compiladores no sería posible programar en lenguajes de alto nivel, ya que un compilador transforma el fichero de entrada que contiene el programa fuente en un fichero ejecutable.

Además, los compiladores nos permiten corregir los errores de los programas que escribimos en lenguajes de alto nivel.

Asimismo, los compiladores nos permiten abordar problemas complejos, lo cual sería imposible utilizando el lenguaje ensamblador.

Pregunta 10

Mostrar el algoritmo visto en clase para diseñar un analizador sintáctico

El algoritmo consta de las siguientes tres etapas:

(1) Diseñar una gramática incontextual no ambigua que genere el lenguaje.

(2) Transformar la gramática del paso (1) en una gramática equivalente cuyo autómata con pila asociado sea determinista.

(3) Programar el autómata con pila asociado a la gramática del paso (2).

Pregunta 11

Explicar cómo se construye la tabla de análisis de una gramática incontextual

Sea $G = (V, \Sigma, P, S)$ una gramática incontextual.

(a) Para toda palabra $\alpha \in (V \cup \Sigma)^*$, definimos el conjunto Primeros(α) como el conjunto de los símbolos terminales que aparecen al comienzo de una palabra derivada a partir de α. Y si la palabra vacía λ se deriva a partir de α, incluimos asimismo λ en los Primeros(α).

(b) Para toda variable $A \in V$, definimos

Siguientes$(A) = \{a \in \Sigma : S \Rightarrow_G^* \alpha A a \beta$ donde $\alpha, \beta \in (V \cup \Sigma)^*\}$.

(c) Si $A \in V$ y $a \in \Sigma$, definimos entonces

TABLA$[A, a] = \{(A, \beta) \in P : a \in$ Primeros$(\beta \cdot$ Siguientes(A))$\}$.

Pregunta 12

Definir el concepto de gramática LL(1) y explicar el interés de las gramáticas LL(1) en el diseño de compiladores

Una gramática incontextual G es LL(1), si el algoritmo de construcción de la tabla de análisis no genera conflictos, es decir, si para todo $A \in V$ y para todo $a \in \Sigma$ se tiene que TABLA$[A, a]$ no contiene más de un elemento.

Las gramáticas LL(1) son interesantes para el diseño de compiladores, porque utilizando la tabla de análisis de la gramática LL(1) se puede programar directamente el autómata con pila asociado a la gramática.

Pregunta 13

Explicar el algoritmo visto en clase para programar el autómata con pila asociado a una gramática LL(1)

public boolean analisis_sintactico(String entrada)
{ Stack⟨Character⟩ pila = new Stack⟨Character⟩();
int q = 0, i = 0; boolean b = true; char c = entrada.charAt(0);
(1) Poner '$' en la pila
(2) Aplicar la transicion $((q_0, \lambda, \lambda), (f, S))$
(3) while ((tope de la pila ! = '$' || caracter de entrada ! = '$')
&& b)
{ if (tope de la pila == 'a' && caracter de entrada == 'a')
{aplicar la transicion $((f, a, a), (f, \lambda))$; leer siguiente caracter;}
else if (tope de la pila == 'A' && caracter de entrada == 'a'
&&TABLA$[A, a] == A \to X1 \dots X_n$)
aplicar la transicion $((f, \lambda, A), (f, X1 \dots X_n))$;
else b = false; }
(4) if (tope de la pila == '$' && caracter de entrada == '$')
return true; else return false; }

Pregunta 14

Explicar en qué consiste la regla de factorización

La regla de factorización, que se utiliza para poder transformar muchas gramáticas no ambiguas en gramáticas LL(1), afirma lo siguiente:

Si tenemos producciones en la gramática de la forma $A \longrightarrow \alpha\beta_1|\ldots|\alpha\beta_n$ donde $\alpha \neq \lambda$ y $n \geq 2$, reemplazar estas producciones por $A \longrightarrow \alpha A'$, $A' \longrightarrow \beta_1|\ldots|\beta_n$ donde A' es una nueva variable.

Pregunta 15

Explicar en qué consiste la regla de recursión

La regla de recursión, que se utiliza para poder transformar muchas gramáticas no ambiguas en gramáticas LL(1), afirma lo siguiente:

Si tenemos producciones en la gramática de la forma $A \longrightarrow A\alpha_1|\ldots|A\alpha_n|\beta_1|\ldots|\beta_m$ donde $n \geq 1$ y los β_i no comienzan por A, reemplazar estas producciones por $A \longrightarrow \beta_1 A'|\ldots|\beta_m A'$, $A' \longrightarrow \alpha_1 A'|\ldots|\alpha_n A'|\lambda$ donde A' es una nueva variable.