Vista general de compilación

Compiladores son programas que traducen un programa escrito en un lenguaje en otro programa en otro lenguaje

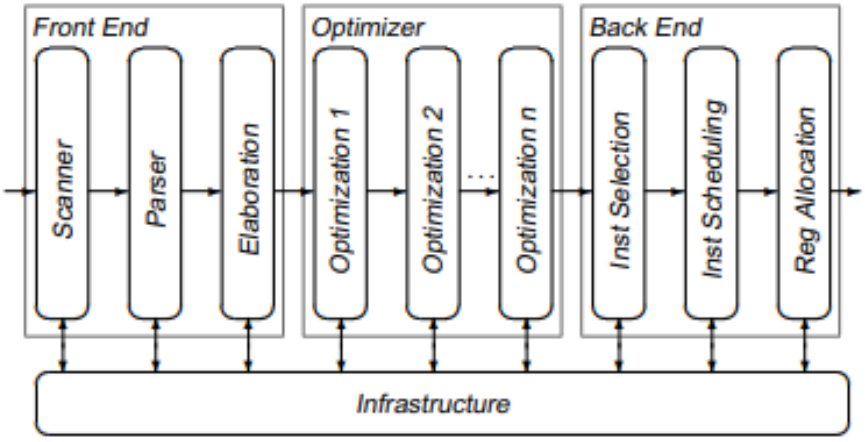
Un compilador traduce de un lenguaje a otro por medio de objetivos

Mientras que aquel que entrega resultados de un programa se le conoce como interprete

Los compiladores tienen reglas como

- El compilador debe preservar el significado del programa a compilar
- El compilador debe de distinguir un camino para la salida

Un compilador puede dividirse en dos partes principales el front end y el back end aunque algunos programadores ponen una tercera parte que es el optimizador



Los compiladores combinan ideas de la teoría de lenguaje y algoritmos de estudio de la inteligencia artificial y mucho de los problemas que se enfrentan en los compiladores son muy difíciles de resolver

Capitulo 2

La tarea de un scanner es transformar la cadena de caracteres que ingresa a otra cadena de caracteres entendible para el lenguaje del compilador

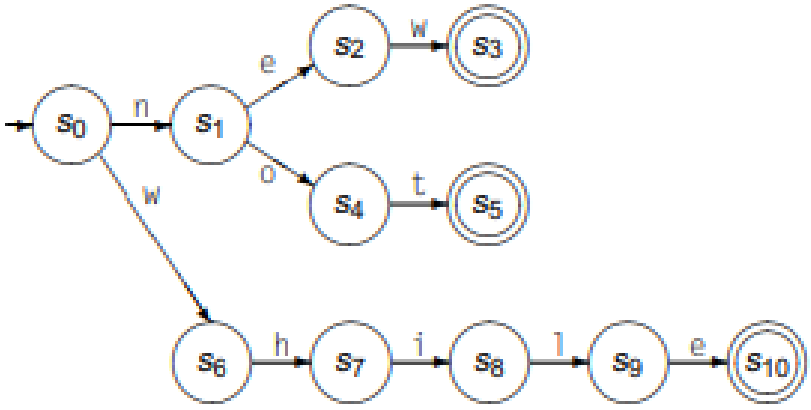
El scanner es la primera de tres partes del proceso de un compilador que usa para entender la entrada del programa, se le conoce como scanner o analizador léxico, este lee una cadena de caracteres que compone una palabra si las palabras son correctas el compilador lo asigna a la categoría sintáctica

Existen problemas con la identificación de los caracteres que componen una palabra un ejemplo es el siguiente

```
c ← NextChar();
if (c = 'n')
   then begin;
      c ← NextChar();
      if (c = 'e')
          then begin;
            c ← NextChar();
            if (c = 'w')
                then report success;
                else try something else;
          end;
          else try something else;
   end;
   else try something else;
```

Donde se identifica con éxito una palabra, pero únicamente una y no más, ese cambio es el que de palabras es el que hace difícil la identificación y genera este problema

Para poder ir solucionando los problemas de identificación de palabras es que se crean autómatas de la siguiente manera



Los cuales abren paso de manera mas amplia a la identificación de palabras para el compilador

Estos autómatas se pueden representar por expresiones regulares que denota la forma del lenguaje que permite una forma intuitiva para los humanos su

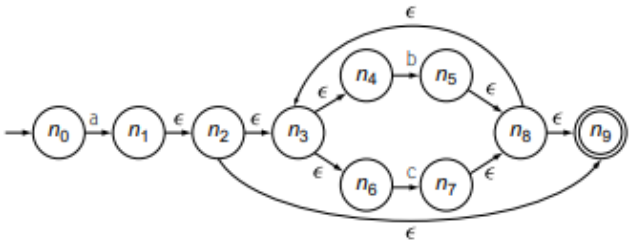
entendimiento de manera informal seria escribir digito por digito, pero existe una forma mejor llamada forma formal y existen tres maneras la alternación, la concentración y la clausura

Para la eliminación de cualquier ambigüedad se tiene el uso de ordenadores como son los paréntesis

Existen dos tipos de autómatas, los deterministas y no deterministas. Un FA debe de realizar una transición antes de examinar el siguiente carácter para garantizar que este no viole las normas definidas. Si en el FA se tienen múltiples transiciones en un estado es un NFA, pero si es de una sola transición es un DFA

Existen equivalencias entre los NFA y los DFA, pero con una construcción mas compleja

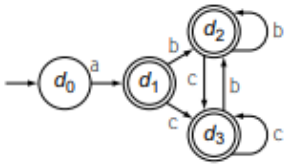
Un ejemplo de esta transformación es el siguiente



(a) NFA for "$a(b \mid c)^*$" (With States Renumbered)

Set Name	DFA States	NFA States	ϵ-closure(Delta(q,*))		
			a	b	c
q_0	d_0	n_0	$\{n_1, n_2, n_3, n_4, n_6, n_9\}$	– none –	– none –
q_1	d_1	$\{n_1, n_2, n_3, n_4, n_6, n_9\}$	– none –	$\{n_5, n_8, n_9, n_3, n_4, n_6\}$	$\{n_7, n_8, n_9, n_3, n_4, n_6\}$
q_2	d_2	$\{n_5, n_8, n_9, n_3, n_4, n_6\}$	– none –	q_2	q_3
q_3	d_3	$\{n_7, n_8, n_9, n_3, n_4, n_6\}$	– none –	q_2	q_3

(b) Iterations of the Subset Construction



Al último se le llama al algoritmo de minimización de numero de estados en el DFA y para este es que se utiliza la técnica de detección de estados iguales

En el nivel de reconocimiento se puede cambiar la implementación de un DFA y su noción de aceptación y al final de la compilación se puede aceptar el original de un DFA

La construcción de un escáner es donde la implementación de un DFA ha producido herramientas a través de la teoría de lenguajes para generar expresiones regulares

El scanner tiene una representación de segmento explicito para las entradas del DFA a través del código directo, el generador emite un código similar al correspondiente al que será interpretado por el compilador

En la practica el compilador escribe la información en barra para el uso simplificado del buffer incrementando el punto del modulo

Posteriormente se generan los lexemas por medio de una tabla de direcciones acumulado en los puntos de entrada de los caracteres de la cadena del lexema

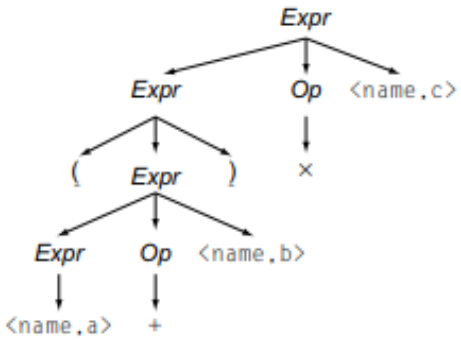
Capítulo 3

El análisis es la segunda estación de un compilador osea es el front end, como uno de sus problemas que tiene este es el de su semejanza con el escáner

La principal función del parser o del analizador es la de comprender e identificar las fallas sintácticamente

Existe un contexto libre de gramática donde la l no es precisamente el símbolo de este y este abre el mundo a una colección de gramáticas, un ejemplo es

Rule	Sentential Form
	Expr
2	*Expr Op* name
6	*Expr* × name
1	(*Expr*) × name
2	(*Expr Op* name) × name
4	(*Expr* + name) × name
3	(name + name) × name



Con estas derivaciones se pueden escribir en forma de G en contraste con el compilador deben tener una cadena de entrada determinista con no muchas derivaciones

Existen algunos problemas con la combinación de expresiones gramaticales y su estructura uno es desplazamiento a la izquierda y este es porque el símbolo se encuentra a la derecha, pero lo requerido a la izquierda

Rule	Sentential Form	Input
	Expr	↑ name + name x name
1	*Term Expr'*	↑ name + name x name
5	*Factor Term' Expr'*	↑ name + name x name
11	name *Term' Expr'*	↑ name + name x name
→	name *Term' Expr'*	name ↑ + name x name
8	name *Expr'*	name ↑ + name x name
2	name + *Term Expr'*	name ↑ + name x name
→	name + *Term Expr'*	name + ↑ name x name
5	name + *Factor Term' Expr'*	name + ↑ name x name
11	name + name *Term' Expr'*	name + ↑ name x name
→	name + name *Term' Expr'*	name + name ↑ x name
6	name + name x *Factor Term' Expr'*	name + name ↑ x name
→	name + name x *Factor Term' Expr'*	name + name x ↑ name
11	name + name x name *Term' Expr'*	name + name x ↑ name
→	name + name x name *Term' Expr'*	name + name x name ↑
8	name + name x name *Expr'*	name + name x name ↑
4	name + name x name	name + name x name ↑