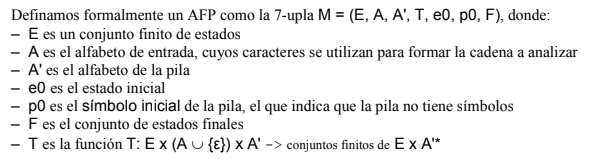
Autómatas finitos con pila (APD)

* Pueden reconocer LR y LIC’s (expresiones aritméticas y sentencias de un lenguaje de programación)
* A diferencia de un autómata normal, este cuenta con una pila (stack) donde se puede almacenar y retirar información a la hora de leer un lenguaje.

El APD puede reconocer o por estado final o por una pila vacía.

Autómatas finitos con pila determinístico (AFPD)

Esta conformado por:  
-Un alfabeto A (símbolos con los que formara la palabra)  
-Un flujo de entrada infinito en una dirección   
-Un alfabeto A’ de símbolos de la pila. (se encuentra el símbolo $, que indica pila vacía)  
- Una pila infinita en una dirección. Inicialmente esta vacía  
- Un conjunto finito y no vacío de estados E.  
- Un estado inicial (único).  
- Un conjunto de estados finales o de aceptación.  
- Una función de transiciones entre los estados que definimos de la siguiente forma:  
  
T: E x (A U {ε}) x A' -> E x A'\*  
  
Para que sea determinístico debe cumplir que solo puede haber un movimiento para cada (e,R) siendo e un estado y R el símbolo en el tope de la pila

Tabla de movimientos

Es similar a la tabla de transiciones, pero también se tiene en cuenta el estado de la pila   
  
Expresiones aritméticas: aquellas que utilicen paréntesis para indicar orden de operación, conforman un LIC.   
  
  
Proceso de compilación  
Un compilador es un complejo programa que lee un programa escrito en un lenguaje fuente, habitualmente un lenguaje de alto nivel y lo traduce a un lenguaje mas cercano a la maquina (bajo nivel), el programa resultante se llama programa objeto.  
  
La compilación esta formada por 3 etapas de análisis:

**Análisis léxico**: Se ocupa de detectar los elementos que conforman un programa fuente, como son los identificadores, operadores, palabras reservadas, constantes, carácters de puntuación.   
  
Se ocupa de los LR del lenguaje de programación, las palabras detectadas se denominan **Lexemas**, y a la categoría a la cual pertenecen se les llama **Categoría léxica** o **Tokens**.

**Análisis sintáctico**: Este tiene como entrada los tokens del análisis léxico. Conoce la sintaxis del programa y se encarga de corroborar que las construcciones son sintácticamente correctas. No puede reconocer errores semánticos, como por ejemplo que se use el mismo identificador para 2 datos distintos

**Análisis semántico:** El análisis semántico realiza la verificación de tipos, viendo que cada operador este con su operando asociado correspondiente.   
  
Las rutinas semánticas realizan 2 funciones:  
- chequean la semántica estática de cada construcción, es decir si tiene sentido y están bien armadas, las variables utilizadas definidas, los tipos correctos, etc.  
- hacen la traducción, generan el código para la maquina virtual para que esta implemente la construcción analizada.   
  
\*No todo el programa puede ser descripto por GICs, la compatibilidad de tipos y reglas de alcance de variables, son sensibles al contexto.   
  
La semántica estática: define las restricciones sensible al contexto, para que el programa sea correcto ej: los identificadores deben estar declarados; los operadores y operandos deben ser de tipos compatibles; las funciones deben ser llamadas con el numero correcto de argumentos.

**La tabla de símbolos(TS):** es una estructura de datos utilizada para almacenar las palabras reservadas; todos los identificadores a compilar; los literales-constantes. Cada elemento de la TS**,** esta formado por una cadena y sus atributos.  
\*Los atributos se representan en números enteros.  
Ejemplo de un elemento en la TS:  
  
“ int XX (double a) ”

El identificador XX debe poseer atributos que indiquen que es una función, que retorna un valor de tipo int, que tiene un parámetro y que ese parámetro es de tipo double. (XX, función, int, 1, double).  
  
**El SCANNER**  
  
\*Un **carácter centinela**: es aquel carácter que nos indica que ya no se esta leyendo el mismo lexema, ejemplo en la expresión edad+20, el centinela seria el operador suma (+), ya que un identificador no puede estar formado por este. Entonces a partir de ese punto, se busca el lexema con el cual encaja.   
  
Se lee y debe ser retornado (función ungetc), para que pueda ser leído nuevamente y detectar su lexema. **\***El Scanner: ignora los espacios, tabulaciones y saltos de líneas.Si el scanner al principio del lexema lee un carácter no valido, (error léxico), reinicia el análisis léxico ignorando este carácter y comenzando con el siguiente. El scanner, estará basado en la construcción de un AFD que reconoce los LRs que forman los tokens.  
Una forma de detectar las palabras reservadas es agregarla a la TS con el atributo “reservada”.  **El PARSER**Debe saber cuando el scanner le pasa el ultimo token, esto lo logra cuando le pasa el token FDT (fin de texto).

El Parser invoca al Scanner cada vez que necesita un token. El Scanner analiza el flujo de entrada a partir de la posición que corresponde, detecta el próximo lexema en el Programa Fuente y retorna el token correspondiente, o tal vez encuentra un error léxico y le informa al Parser de esta situación anómala.  
  
**Funciones útiles:  
La función ungetc de ANSI C.**

- El vector de caracteres externo que llamamos buffer, utilizado para almacenar los caracteres de los identificadores y los dígitos de las constantes en la medida que son reconocidos por el AFD.

- La función con prototipo void AgregarCaracter(int); que añade un carácter al buffer.

– La función con prototipo TOKEN EsReservada(void); que, dado el identificador reconocido por el AFD y almacenado en el buffer, retorna el token que le corresponde

– La función con prototipo void LimpiarBuffer(void); que “vacía” el buffer para el próximo uso.

– La función feof de ANSI C, para detectar el centinela del texto, el FDT, y así saber que todos los caracteres del Programa Fuente fueron procesados.

– La función fgetc de ANSI C, para leer cada carácter del flujo de entrada.

**VOLUMEN 3**

Un autómata finito es un mecanismo que reconoce a un LR.   
  
Autómata finito determinístico: Son aquellos que no presentan ambigüedades para la lectura de ningún carácter en ningún estado y no existe transiciones por Lambda.

Se compone de 3 conjuntos  
- Conjunto finito de estados, se designa estado inicial (único) y estados finales o aceptación (varios)

- Un alfabeto

- Un conjunto finito de transacciones.

Texto

Descripción generada automáticamente

* Un AFD es completo cuando cada estado tiene exactamente una transición para cada carácter del alfabeto
* Es completo cuando su tabla de transiciones no tiene huecos.
* Para completar los huecos se agregan estados de rechazo/ no aceptación.

Dos AFDs son equivalentes cuando reconocen el mismo LR.  
  
**Autómata finito no determinístico (AFN)**

Texto

Descripción generada automáticamenteDefinición formal, es igual salvo que cambia la función de transiciones T  
**Autómata finito con transiciones-e**Se caracteriza por contar con 1 o mas transiciones de tipo “e”, donde no se lee ningún carácter de la cadena a analizar y se cambia a otro estado.  
La función de transiciones de este es:

* La tabla de transiciones de un AFN con T e, debe tener una columna adicionar por el símbolo e.

**Pasar de una ER a un Autómata**

* Siguiendo el siguiente algoritmo  
  ER -> Algoritmo de Thompson -> AFN -> Algoritmo de clausura-e -> AFD -> Algoritmo de clases -> AFD mínimo

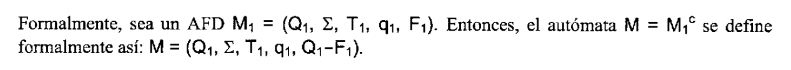
**Algoritmo de Thompson**Consiste en :

- Separar la ER de partida en sus componentes básicos: caracteres, operadores y e  
- Generar un AF básico por cada carácter de la ER  
- Componer estos autómatas hasta lograr el AF que reconoce la ER dada. Se obtiene un AFN con transiciones-e   
  
\* Este algoritmo trabaja solo con unión, concatenación y clausura de kleene.

\* Características de un AFN generado por Thompson:   
Al estado inicial no llegan transiciones  
Estado final es único  
Del estado final no hay transiciones  
De cualquier estado no final pueden salir una única transición con un carácter del alfabeto, 1 o 2 transiciones-e

**Del AFN al AFD**

\*Para todo AFN existe un AFD que reconoce el mismo lenguaje.  
 **Clausura-e de un estado**: la clausura-e (q) es el conjunto de estados formados por q y todos aquellos estados a los cuales se llega, desde q, utilizando solo transiciones-e (el conjunto clausura-e nunca puede ser vacío, como mínimo contiene a su propio estado)  
  
**Clausura-e de un conjunto de estados**= sea R un conjunto de estados, la clausura-e (R) es la unión de las clausura-e de los estados que componen a R  
  
**Conjunto HACIA:** es el conjunto de estados a los cuales se transita por el símbolo X, hacia(R,x) ejemplo 1=>A=>4 2=>A=>5 3=>B=>6  
R= {1,2,3} hacia(R,a)={4,5} Hacia(R,b)={6}

**Operaciones fundamentales con Autómatas Finitos  
  
Complemento de un AFD**El complemento de un AFD es un AFD, se obtiene invirtiendo los estados finales y no finales. Todos sus estados finales serán no finales y viceversa. Lo mismo con los estados de rechazo, se convertirán en finales.  
  
  
  
**Intersección de dos AFDs**La intersección de dos AFDs también es un AFD, reconoce las palabras comunes que ambos AFDs reconocen. Los estados del AFD, son pares ordenados de estados, uno de cada AFD.  
Texto

Descripción generada automáticamente  
  
**Del autómata finito a la ER**Algunos programas solicitan que el dato sea un AFD, mientras que otros como el LEX una ER.   
  
**Depuración del autómata finito**  
\*Hay dos tipos de estados erróneos:  
-Estados a los que no se puede llegar desde un estado inicial (estado inalcanzable)

-Estados que no conducen a un estado final (estado de rechazo)La depuración consiste en eliminar estos estados erróneos.  
**Resolución de sistemas de ecuaciones**Consiste en establecer una S.E con tantas ecuaciones como estados haya en el AF depurado. Cuando este S.E sea resuelto se encontrará la ER buscada.

Plantear ecuaciones:

Diagrama

Descripción generada automáticamente

**Reducciones**  
Se realiza en la ecuación que tiene uno o mas ciclos. (Ecuaciones recursivas)  
Texto, Carta

Descripción generada automáticamente  
Texto

Descripción generada automáticamente

La ecuación que corresponde al estado inicial debe ser la ultima en resolverse, porque a su lado derecho quedara la ER buscada. **Obtención del AFD MINIMO**El afd mínimo es el AFD con la cantidad mínima de estados que reconoce a un LR.   
Obtener el afd mínimo permite 3 aplicaciones:  
-Determinar si dos o mas afd son equivalentes  
- Probar la equivalencia de 2 o mas ER  
- El AFD mínimo es el que tiene la TT mas reducida- Dos AFDs son equivalentes si el AFD mínimo es el mismo.  
-Dos ER son equivalentes si son reconocidas por el mismo AFD mínimo

**El algoritmo**(es muy largo) **Maquina de TURING   
Texto, Carta

Descripción generada automáticamente**