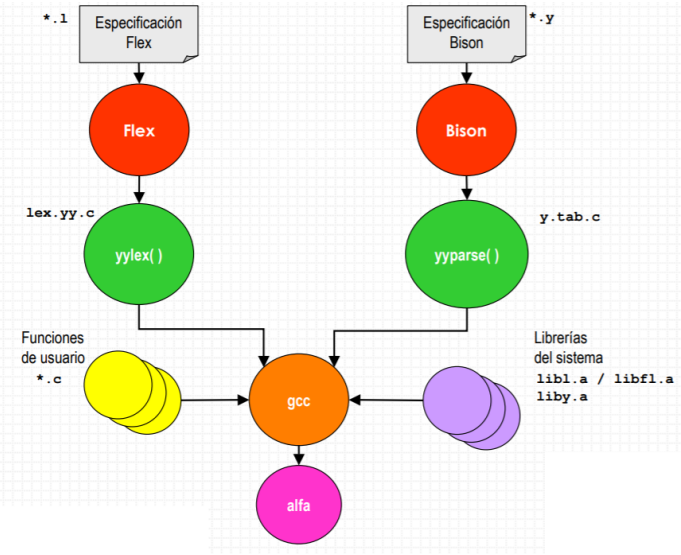
RESUMEN SEGUNDO PARCIAL

Bison

Parser: Encargado de realizar un análisis sintáctico mediante un análisis secuencial de tokens (categorías léxicas) identificando categorías sintácticas válidas.

Bison: Herramienta que genera parsers de forma automática a través de un archivo de especificación que le permita saber las estructuras del programa que se necesita reconocer (categorías sintácticas). Utiliza gramáticas independientes de contexto (GIC). La integración con el analizador léxico ocurre de la siguiente manera:

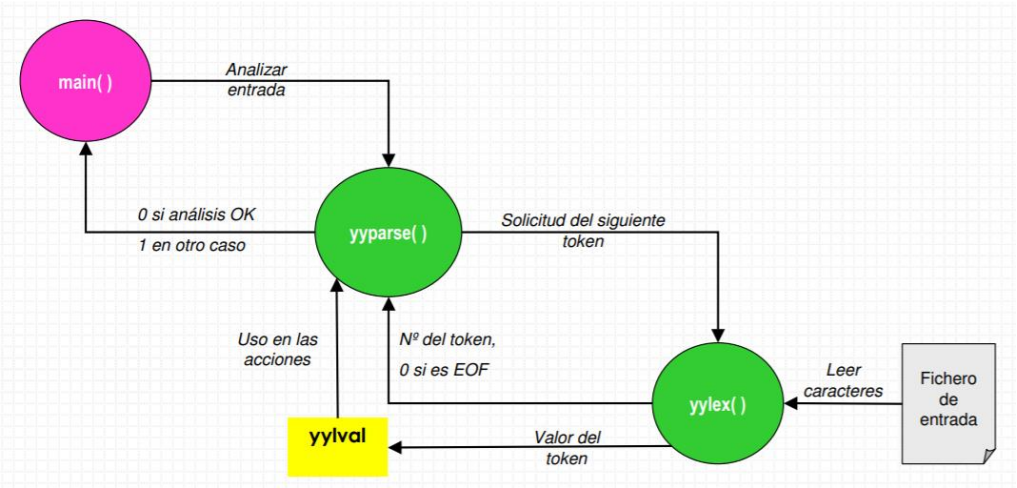


La **comunicación** entre los analizadores ocurre de la siguiente manera:

Desde la función main() se invoca la función yyparse() (analizador sintáctico) quien a su vez invoca varias veces durante el análisis a la función yylex() (analizador léxico) para solicitar los tokens que van siendo reconocidos de un archivo de entrada (secuencia de caracteres).

Cada token se identifica con un número entero y se almacena su valor semántico asociado en la variable global compartida *yylval* que será utilizada posteriormente en las acciones de las producciones de la gramática. A medida que yyparse() solicita los tokens comprueba si forman una construcción válida de acuerdo con las reglas de gramática independiente del contexto descrita en el archivo de especificación.

El yyparse() devuelve un 0 si termina el análisis con éxito, es decir, si la entrada es sintácticamente correcta. Cuando detecta un error sintáctico invoca a la función yyerror() y termina el proceso de análisis devolviendo un 1 a la función principal main().



Secciones del archivo de Bison:

Definiciones: Dentro de esta sección podemos encontrar:

- **%union:** Por defecto la variable yylval es de tipo int. Define una estructura union de C con un campo para cada tipo de valor semántico que se requiera asociar a los tokens. Solo se puede utilizar un campo de este. Su tamaño es el de su miembro de mayor tamaño.
- **%token:** Se utiliza para definir los símbolos terminales de la gramática.
- **%type:** Permite declarar el tipo de los símbolos no terminales. Los no terminales a los que no se les asigna ningún valor a través de $$ no es necesario declararlos.
- **%left/%right:** Permiten declarar la asociatividad de los operadores de la gramática.
- **%start:** Declara el axioma de la gramática. Si se omite, se asume que el axioma es el primer no terminal de la sección de reglas.

Reglas: Contiene las reglas de la gramática escritas en un formato concreto y opcionalmente con acciones asociadas a las reglas. La mayoría de las acciones trabajan con los valores semánticos de los símbolos de la parte derecha, accediendo a ellos mediante pseudo-variables del tipo $N, donde N representa la posición del símbolo. El valor semántico del símbolo no terminal de la parte izquierda de la regla se referencia con $$.

Código de usuario: Funciones diseñadas por el usuario para ser utilizadas en la sección de reglas. En este sector es conveniente definir la función yyerror() para que cuando yyparse() detecte un error sintáctico, la invoque.

Análisis Sintáctico

- Llevado a cabo por el **Parser**, que verifica si los tokens forman secuencias o construcciones validas.
- Convierte el flujo de tokens en un árbol de análisis sintáctico. Representa la secuencia analizada, en donde los tokens que forman la construcción son las hojas del árbol.

Tipos de Análisis Sintácticos y de GICs:

Al derivar una secuencia de tokens, si existe más de un noterminal debemos elegir cuál es el próximo noterminal que se va a expandir. Existen dos tipos de derivación: a IZQUIERDA o a DERECHA. Gracias a estas, podemos "obtener" dos tipos de análisis sintácticos:

- Análisis Sintáctico Descendente: produce una derivación por Izquierda que comienza por el axioma y finaliza con los terminales que forman la construcción analizada.
- Análisis Sintáctico Ascendente: Utiliza la derivación a derecha, pero en un orden inverso, es decir, que la última producción aplicada en la derivación a derecha es la primera producción utilizada y la que involucra al axioma es la última producción en ser descubierta.

Hay reglas sintácticas que no pueden ser expresadas empleando únicamente GICs, es por eso, que son considerados parte de la semántica estática y son chequeados por rutinas semánticas.

Gramáticas LL y LR

Análisis Sintáctico LL: Consiste en analizar el flujo de tokens de izquierda a derecha por medio de una derivación a izquierda.

Gramática LL: Aquella utilizada en el análisis sintáctico LL.

LL(1) o Parser Predictivo:

- Gramática especial que puede ser utilizada por una Parser LL con un solo símbolo de preanálisis. Es decir, que, si un noterminal tiene varias producciones, puede decidir cual lado derecho debe aplicar con solo conocer el próximo token.
- Los análisis sintácticos descendentes deben basarse en estas mismas.

- **NO** deben ser recursivas a izquierda y no pueden tener un noterminal con dos o más producciones cuyos lados derechos comiencen con el mismo terminal.

Análisis Sintáctico LR: Consiste en analizar el flujo de tokens de izquierda a derecha por medio de una derivación por derecha, aunque utilizada a la inversa, ya que la última producción expandida en la derivación será la primera en ser reducida.

LR(1): Solo requiere conocer el próximo token para hacer un análisis sintáctico correcto.

Factorización a Izquierda: Se utiliza cuando hay un prefijo común. Se deben modificar las producciones de este noterminal de tal forma que el prefijo común quede aislado en una sola producción.

```
A -> αβ1 |
     . . .
     αβn

A -> αB
B -> β1 |
     ...
     βn
```

Eliminación de la recursividad a izquierda: Se debe eliminar la recursividad a izquierda de la siguiente manera:

```
X -> Xα | β

X -> βZ          toda secuencia comienza con β
Z -> αZ | ε      seguida de cero o más α
```

Símbolos de Preanálisis, el conjunto Primero y el conjunto Siguiente

Para poder realizar un análisis sintáctico descendente LL(1), es necesario que, para cada noterminal de la GIC, el Parser puede determinar la producción aplicar con solo conocer cuál es el **símbolo de preanálisis** (próximo token). Es por esto, que es importante conocer el conjunto de todos los posibles símbolos de preanálisis.

Conjunto Primero: Sirve para distinguir como se expande un noterminal si este mismo tiene dos o más producciones. Sea la producción A → X1...Xm, entonces PRIMERO es el conjunto de terminales que pueden iniciar cualquier cadena de derivación. Si X1...Xm puede derivar en ε, entonces también lo contiene.

Algoritmo para obtener el conjunto Primero:

1. Si el primer símbolo, X1, es un terminal, entonces Primero(X1...Xm) = {X1}.
2. Si X1 es un noterminal, entonces se calculan los conjuntos Primero para cada lado derecho de las producciones que tenga X1.
3. Si X1 puede generar ε, entonces X1 puede ser eliminada y, en consecuencia, Primero(X1...Xm) depende de X2.
4. Si X2 es un terminal, entonces es incluido en Primero(X1...Xm).
5. Si X2 es un noterminal, entonces se calculan los conjuntos Primero para cada lado derecho de las producciones que tenga X2.
6. En forma similar, si tanto X1 como X2 pueden producir ε, consideramos X3, luego X4, etc.

Conjunto Siguiente: Es un conjunto de terminales formado por aquellos que siguen inmediatamente al noterminal A en las cadenas de derivación que contengan al noterminal A. Define el contexto derecho para este noterminal.

Para obtenerlo debemos:

- Si el noterminal A esta seguido por el terminal x, entonces x pertenece a Siguiente(A).
- Si el noterminal A esta seguido por el noterminal B, entonces Siguiente(A) incluye Primero(B).
- Si el noterminal A es el último símbolo del lado derecho de cierta producción de un noterminal T, entonces Siguiente(A) incluye a Siguiente(T).

Algoritmo de Thompson

Consiste en:

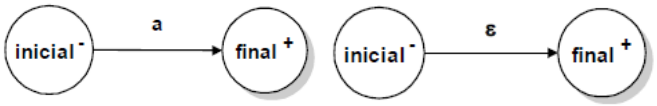
1. Desmembrar la ER en sus componentes básico, es decir, caracteres, operadores y ε.
2. Generar un AF básico por cada carácter o símbolo ε.
3. Componer estos autómatas básicos según los operadores existentes en la ER.

Trabaja UNICAMENTE con los tres operadores oficiales de las ERs: Unión, Concatenación y Clausura de Kleene.

Sus características generales son:

- Al estado inicial no llegan transiciones.
- El estado final debe ser único.
- Del estado final no parten transiciones.
- De cualquier estado no final pueden partir:
  - Una sola transición etiquetada con un carácter,
  - Una sola transición-ε,
  - Dos transiciones-ε.

Autómata para cada carácter y para el símbolo ε: El AF genérico que reconoce un símbolo del alfabeto o el símbolo ε es:



Autómata para la unión:

Se construye de la siguiente forma:

1. Se construyen los dos autómatas básicos.
2. Los estados iniciales y finales dejan de serlo.
3. Se agrega un nuevo estado inicial.
4. Se incorporan dos transiciones- ε que relacionarán al nuevo estado inicial con los dos ex estados iniciales.
5. Se añade un nuevo estado final.
6. Se trazan dos transiciones- ε para unir a los dos ex estados finales con este nuevo estado final.

Autómata para la concatenación:

Se construye de la siguiente forma:

1. Se construyen los autómatas básicos.
2. El estado final del autómata izquierdo deja de serlo y el estado inicial del autómata derecho también deja de serlo.
3. Se agrega una transición-ε que vincule al ex estado final del autómata izquierdo con el ex estado inicial del autómata derecho.
4. Queda como estado inicial el del autómata izquierdo y como estado final el del autómata derecho.

Autómata para la Clausura de Kleene:

Se construye de la siguiente forma:

1. Se construye el autómata básico.
2. El estado inicial y final dejan de serlo.
3. Se incorporan un nuevo estado inicial y un nuevo estado final.
4. Se agrega una transición-ε desde el nuevo estado inicial hasta el ex estado inicial.
5. Se agrega una transición-ε desde el ex estado final al nuevo estado final.
6. Se incorpora una transición-ε desde el nuevo estado inicial al nuevo estado final.
7. Se agrega una transición-ε desde el ex estado final al ex estado inicial.

Algoritmo de Clausuras-ε

PARA TODO AFN EXISTE UN AFD EQUIVALENTE

Clausura-ε de un estado: Sea q un estado, entonces la Clausura-ε(q) es el conjunto de estados, formado por q y por todos aquellos estados a los cuales se llega, desde q, utilizando solo transiciones-ε. Este mismo *nunca puede ser vacío*.

Clausura-ε de un conjunto de estados: Sea R un conjunto de estados, entonces la Clausura-ε (R) es la unión de las clausuras-ε de los estados que componen el conjunto R.

Conjunto "Hacia": Sea R un conjunto de estados y sea x un símbolo del alfabeto. Entonces, hacia(R,x) es el conjunto de estados a los cuales se transita por el símbolo x. Es decir, es el conjunto de estados de llegada para (R,x).

Algoritmo de Clausuras- ε: Sirve para construir un AFD a partir de un AFN. Para realizarlo debemos determinar tres cosas:

- El estado inicial del AFD.
- Sus estados finales.
- Su tabla de transiciones.

El estado inicial del AFD es la clausura-ε del estado inicial del AFN dado, mientras que los estados finales del AFD son todos aquellos conjuntos de estados del AFN que contienen, por lo menos, un estado final.

La **TT** se obtiene siguiendo los siguientes pasos:

1. Se obtiene el estado inicial del AFD, que es la clausura-ε del estado inicial del AFN.
2. Se agrega este estado a la primera columna de la tabla.
3. Para cada símbolo, se calcula el conjunto hacia del estado.
4. Se determinan nuevos estados del AFD por medio de la clausura- ε de cada conjunto hacia y se los agrega a la tabla.
5. Si un nuevo estado obtenido no existe, se lo agrega.
6. Se repiten los pasos (3) al (5) hasta que no surjan nuevos estados.

Algoritmo de Clases

AFD Mínimo: Es aquel que posee la mínima cantidad de estados. Se lo considera el único autómata óptimo. La obtención de este mismo tiene tres aplicaciones importantes:

- Determinar si dos AFDs son equivalentes.
- Probar la equivalencia de dos o más Expresiones Regulares.
- Al tener la menor cantidad de filas, beneficia la implementación computacional del mismo.

Algoritmo de Clases:

1. **Partición del conjunto de estados:** Se divide el conjunto de estados en dos clases, la clase de los estados no finales y la clase de los estados finales.

2. **Detección de estados equivalentes inmediatos:** Si dos o más estados son equivalentes, significa que solo uno de ellos es necesario, por lo que se puede reducir el AFD dejando solo uno de ellos.

   Comportamiento: Es la sucesión de estados de llegada de las transiciones que parten de ese estado, es decir, la n-upla formada por los estados de su fila en la TT.

   Estados equivalentes inmediatos: Si pertenecen a la misma clase y tienen el mismo comportamiento.

3. **Eliminación de los estados equivalentes inmediatos.**
4. **Construcción de la Tabla de Transiciones por Clases (TTC):** Se forma a partir de la última TT obtenida y las transiciones indicadas en la tabla original, reemplazando cada estado de llegada por la clase a la que pertenece.

5. **Búsqueda de estados equivalentes por clase:** Provoca una partición de la clase en subclases que se caracterizan por contener estados que son equivalentes por clase. El proceso continúa hasta que se llega a que cada clase este formada por un solo estado, o bien algunas clases están constituidas por mas de un estado, dejando solo un representante de esta.

   Comportamiento por clases: Sucesión de clases de llegada de las transiciones que parten de ese estado.

   Estados equivalentes por clase: Son aquellos que están en la misma clase y que tienen el mismo comportamiento por clases.

Máquina de Turing

Es un autómata determinístico con la capacidad de reconocer cualquier lenguaje formal. Está formada por los siguientes elementos:

- Un alfabeto A de símbolos del lenguaje.
- Una cinta infinita dividida en una secuencia de celdas, cada una con un carácter o en blanco. En esta misma se coloca la cadena a analizar.
- Una cabeza de cinta que puede leer el contenido, reemplazarlo con otro carácter o dejar el mismo y reposicionarse.
- Un alfabeto A' de símbolos que pueden ser escritos en la cinta por la cabeza de cinta.
- Un conjunto finito de estados que incluye un estado inicial y un conjunto de estados finales que producen la terminación de la ejecución.
- Un programa, que es un conjunto de reglas que nos dicen, en función del estado y del carácter leído, qué carácter escribir en la cinta en la misma posición, en qué dirección moverse y a que estado debe realizar la transición.