Apuntes de lenguajes formales y compiladores

Juan Sebastián Díaz Osorio

Universidad EAFIT

3 de mayo de 2020

Disclaimer

Esta presentación sirve como una **guía rápida** de la materia de lenguajes formales y compiladores.

Disclaimer

Esta presentación sirve como una guía rápida de la materia de lenguajes formales y compiladores.

No se analiza el origen o practicidad de los procedimientos abordados en compiladores, pero sí se ejemplifican y describen las nociones más importantes de cada método.

Conversión de gramáticas lineales por izquierda

Método de conversión

Si tenemos una gramática G_1 que es lineal por izquierda, vamos a cambiar su orientación así:

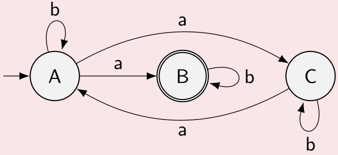
$$\begin{aligned} G_1 : A &\to Ab \mid Ba \mid Ca \\ B &\to Bb \mid \epsilon \\ C &\to Cb \mid Aa \end{aligned} \quad \Rightarrow \quad \begin{aligned} G_1^R : A &\to bA \mid aB \mid aC \\ B &\to bB \mid \epsilon \\ C &\to bC \mid aA \end{aligned}$$

Método de conversión

Dibujaremos el autómata correspondiente:

$$G_1^R : A \to bA \mid aB \mid aC$$
$$B \to bB \mid \epsilon$$
$$C \to bC \mid aA$$

$\Rightarrow$

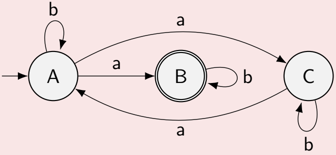


Conversión de gramáticas lineales por izquierda

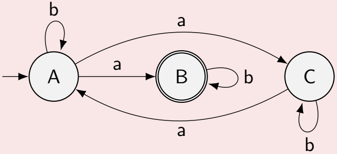
Método de conversión

Dibujaremos el autómata correspondiente:

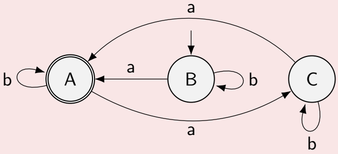
$$
\begin{aligned}
G_1^R : A &\to bA \mid aB \mid aC \\
B &\to bB \mid \epsilon \\
C &\to bC \mid aA
\end{aligned}
\quad \Rightarrow
$$



Invertiremos sus flechas y sus iniciales y finales:



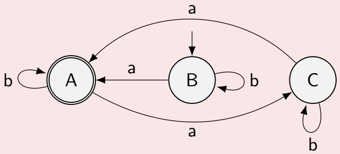
$\Rightarrow$



Conversión de gramáticas lineales por izquierda

Método de conversión

Ya este último autómata lo traduciremos a una gramática:

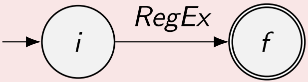


$\Rightarrow$

$$\begin{aligned} G_1' : B &\to bB \mid aA \\ A &\to bA \mid aC \mid \epsilon \\ C &\to bC \mid aA \end{aligned}$$

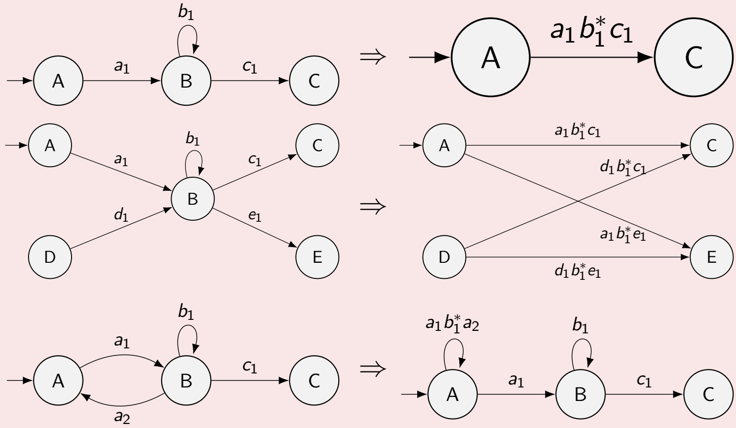
Preparando BMC

Crearemos un nuevo estado inicial **i** y un nuevo estado final **f**. Haremos que este estado i apunte a los iniciales con movimientos ϵ, y que al estado f le apunten los finales con movimientos ϵ.
El objetivo será reducir todo con ciertas transformaciones hasta que la transición de **i** a **f** nos describa nuestra expresión regular.



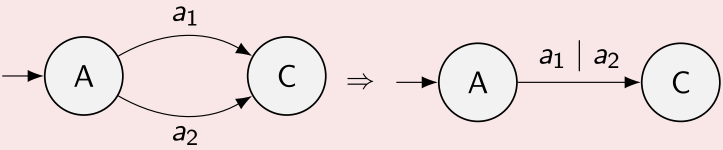
Aplicando BMC

Reglas de transformación básicas:



Aplicando BMC

Reglas de transformación básicas:



Eliminación del no determinismo

Método general

Algunos métodos que siguen a este tema nos ahorrarán total o parcialmente esta tarea, pero vale la pena mencionar la idea general.

Dado un autómata finito no determinista (AFND) que **está limpio** y tiene un **único estado inicial**, haremos un proceso de dos fases para volverlo determinista.

Eliminación del no determinismo

Fase 1: Eliminación de movimientos nulos

Traduciremos el autómata como una gramática y eliminaremos las **reglas de copia**. Luego de esto, debemos limpiarlo y eliminar reglas circulares ($S \Rightarrow A \Rightarrow S$). Ya aquí podemos transcribir otra vez el autómata.

Eliminación del no determinismo

Fase 1: Eliminación de movimientos nulos

Traduciremos el autómata como una gramática y eliminaremos las **reglas de copia**. Luego de esto, debemos limpiarlo y eliminar reglas circulares ($S \Rightarrow A \Rightarrow S$). Ya aquí podemos transcribir otra vez el autómata.

Fase 2: Conjunto potencia

Aplicaremos el método del conjunto potencia que, por su complejidad, se describe en la siguiente diapositiva.

Eliminación del no determinismo

Fase 1: Eliminación de movimientos nulos

Traduciremos el autómata como una gramática y eliminaremos las **reglas de copia**. Luego de esto, debemos limpiarlo y eliminar reglas circulares ($S \Rightarrow A \Rightarrow S$). Ya aquí podemos transcribir otra vez el autómata.

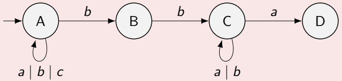
Fase 2: Conjunto potencia

Aplicaremos el método del conjunto potencia que, por su complejidad, se describe en la siguiente diapositiva.

Puede que el autómata solo requiera la primera fase o la segunda para volverse determinista. Debemos estar atentos a su forma.

Definiciones

El estado inicial del autómata (llamémoslo **A**) formará una clase de equivalencia **[A]**.
Hay una función $\boldsymbol{\delta}$ que recibe una clase de equivalencia $[\mathbf{A_1},\ \mathbf{A_2},\ \ldots,\ \mathbf{A_n}]$ y un carácter del alfabeto $\boldsymbol{\Sigma}$ (digamos **b**).
Esta función devuelve otra clase de equivalencia con todos los estados a los que se puede llegar con la transición $\mathbf{A_i} \xrightarrow{\boldsymbol{b}}$.
Por ejemplo:



$$\Rightarrow \quad \begin{array}{c} \delta([A],\ b) = \{A,\ B\} = [A, B] \\ \delta([A,\ B],\ b) = \{A,\ B\} \cup \{C\} = [A,\ B,\ C] \end{array}$$

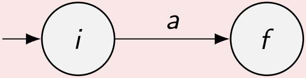
El método general se basa en aplicar δ al caso base **[A]** y seguir haciéndolo con todos los estados que vayan resultando. Se debe probar con todos los carácteres.

Método estructural de Thompson

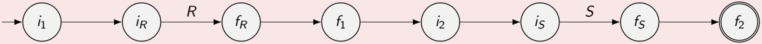
Estructuras de Thompson

Aquí las estructuras básicas.

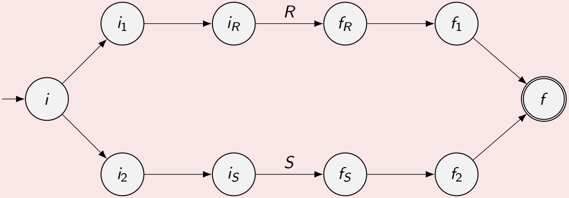
a:



RS:



$R \mid S$:

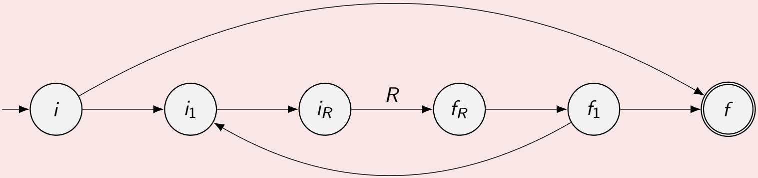


Método estructural de Thompson

Estructuras de Thompson

Aquí las estructuras básicas.

$R*$:



Método Glushkov, McNaughton y Yamada

Definiciones

Sea L un lenguaje.

Ini(L), conjunto con todos los símbolos con los que puede **iniciar** el lenguaje.

Fin(L), conjunto con todos los símbolos con los que puede **terminar** el lenguaje.

Dig(L), conjunto con todas las **palabras de dos letras** que se pueden encontrar dentro de las palabras del lenguaje.

Método Glushkov, McNaughton y Yamada

Definiciones

Sea L un lenguaje.
Ini(L), conjunto con todos los símbolos con los que puede **iniciar** el lenguaje.
Fin(L), conjunto con todos los símbolos con los que puede **terminar** el lenguaje.
Dig(L), conjunto con todas las **palabras de dos letras** que se pueden encontrar dentro de las palabras del lenguaje.

Ejemplo

$$(ab \mid c)^* d$$

Método Glushkov, McNaughton y Yamada

Definiciones

Sea L un lenguaje.
Ini(L), conjunto con todos los símbolos con los que puede **iniciar** el lenguaje.
Fin(L), conjunto con todos los símbolos con los que puede **terminar** el lenguaje.
Dig(L), conjunto con todas las **palabras de dos letras** que se pueden encontrar dentro de las palabras del lenguaje.

Ejemplo

$$(ab \mid c)^* d$$

Ini(L) $= \{$a, c, d$\}$
Fin(L) $= \{$d$\}$ Dig(L) $= \{$ab, ba, bc, bd, cd, ca$\}$

Método Glushkov, McNaughton y Yamada

Construcción del autómata

- Numerar, en caso de símbolos repetidos, cada letra de la expresión.
- Cada letra o símbolo de la expresión regular serán un **estado**.
- Crearemos un estado inicial $\mathbf{q_0}$ que apuntará a los estados en **Ini** con sus mismas letras ($q_0 \xrightarrow{a} a,\ q_0 \xrightarrow{b} b, \ldots$).
- Los estados en **Fin**, serán estados de aceptación. El estado $\mathbf{q_0}$ también lo será si el lenguaje acepta la cadena vacía.
- El conjunto **Dig** definirá las transiciones adicionales. Por ejemplo, si $ab \in Dig$, entonces haremos $a \xrightarrow{b} b$.

Definición de local

Una expresión regular e es local si los conjuntos **Ini, Fin y Dig** solo describen sus derivaciones.

Definición de local

Una expresión regular e es local si los conjuntos **Ini, Fin y Dig** solo describen sus derivaciones.

Ejemplo: $b(aa)^+b$ no es local; $a^*(bc)^*d$ si es local.

Definición de local

Una expresión regular e es local si los conjuntos **Ini, Fin y Dig** solo describen sus derivaciones.

Ejemplo: $b(aa)^+b$ no es local; $a^*(bc)^*d$ si es local.

Una característica que las hace locales inmediatamente es que **todos sus símbolos sean diferentes** (a esto se le llama expresión regular lineal).

Definición de local

Una expresión regular e es local si los conjuntos **Ini, Fin y Dig** solo describen sus derivaciones.

Ejemplo: $b(aa)^+b$ no es local; $a^*(bc)^*d$ si es local.

Una característica que las hace locales inmediatamente es que **todos sus símbolos sean diferentes** (a esto se le llama expresión regular lineal).

El hecho de que una expresión sea local hará que al aplicar GMY obtengamos un **autómata determinista**.

Tabla de computaciones

Iniciales	Finales
$Ini(\emptyset) = \emptyset$	$Fin(\emptyset) = \emptyset$
$Ini(\epsilon) = \emptyset$	$Fin(\epsilon) = \emptyset$
$Ini(a) = \{a\}$ para cada terminal a	$Fin(a) = \{a\}$ para cada terminal a
$Ini(e \cup e') = Ini(e) \cup Ini(e')$	$Fin(e \cup e') = Fin(e) \cup Fin(e')$
$Ini(e.e') = Ini(e) \cup Null(e)Ini(e')$	$Fin(e.e') = Fin(e') \cup Fin(e)Null(e')$
$Ini(e^*) = Ini(e^+) = Ini(e)$	$Fin(e^*) = Fin(e^+) = Fin(e)$

Digrams
$Dig(\emptyset) = \emptyset$
$Dig(\epsilon) = \emptyset$
$Dig(a) = \emptyset$, para cada terminal a
$Dig(e \cup e') = Dig(e) \cup Dig(e')$
$Dig(e.e') = Dig(e) \cup Dig(e') \cup Fin(e)Ini(e')$
$Dig(e^*) = Dig(e^+) = Dig(e) \cup Fin(e)Ini(e)$

Método Glushkov, McNaughton y Yamada

Tabla de computaciones

$Null(\epsilon) = \{\epsilon\}$	$Null(\emptyset) = \emptyset$
$Null(a) = \emptyset$ para cada terminal a	$Null(e \cup e') = Null(e) \cup Null(e')$
$Null(e.e') = Null(e) \cap Null(e')$	$Null(e^*) = \{\epsilon\}$
$Null(e^+) = Null(e)$	

Método Berry Sethi

Definiciones

Sea a un carácter del alfabeto.

Fol(a), conjunto con todos los símbolos que van después de a en los digrafos. Su notación convencional es una tabla dispuesta así:

Fol	Siguientes
a	a, c, d
b	b, d
c	d, $\dashv$

Recordemos que $\dashv$ significa **fin de cadena**.

Método Berry Sethi

Construcción del autómata (1/2)

Cada estado de este autómata tendrá por nombre un **conjunto**.

De la expresión regular e:

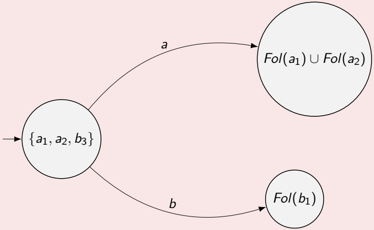
- **Numeramos** cada símbolo de la expresión.
- **Calculamos** los conjuntos Ini(e⊣), Fin(e), Dig(e).
- Hacemos la tabla de **Fol** con ayuda de los Dig. Tener en cuenta que si $a \in Fin(e)$, entonces a tendrá también ⊣ en la lista de siguientes.
- El conjunto **Ini** será el estado inicial.

El siguiente proceso se aplicará al estado inicial y, posteriormente, a los que se vayan creando. En la implementación marcaremos como **visitados** algunos estados para no repetir innecesariamente.

Método Berry Sethi

Construcción del autómata (2/2)

- Reconocemos los carácteres iguales de un estado $(\{a_1, a_2, b_3\} \Rightarrow (a_1, a_2)\ y\ (b_3))$ y creamos una transición con esa letra hacia un estado con los Fol de cada elemento común. Por ejemplo, para un estado con tres elementos: a_1, a_2 y b_3, crearemos una transición para $Fol(a_1) \cup Fol(a_2)$ con **a**, y otra para $Fol(b_3)$ con **b**.



Reconocedores en autómatas

Reglas del complemento

Dado un autómata determinista, podemos definir su complemento:

- Crearemos el estado sumidero p.
- Los estados finales dejan de serlo, y los estados normales se vuelven finales (aquí entra el estado sumidero también).

El autómata dado ya lee el complemento y por BMC se puede obtener su respectiva expresión regular.

Reconocedores en autómatas

Reglas de la unión

Dados dos autómatas deterministas, podemos definir su unión:

- Crearemos el estado inicial principal i.
- i se conecta con movimientos ϵ a todos los estados iniciales de los dos autómatas.
- Se debe convertir el autómata en determinista (sea por conjunto potencia o BS).

El autómata dado ya lee la union y por BMC se puede obtener su respectiva expresión regular.

Reconocedores en autómatas

Otros reconocedores

Por simplificación, podemos interpretar la intersección y la diferencia como:

$$L_1 \cap L_2 \equiv \neg(\neg L_1 \cup \neg L_2)$$

$$L_1 - L_2 \equiv L_1 \cap \neg L_2 \equiv \neg(\neg L_1 \cup L_2)$$

Autómatas de pila

Definición

Un autómata de pila podemos verlo como un autómata con memoria de su recorrido. Sus transiciones se definen así:

$$\frac{(cc)(pop)}{(push)}$$

Donde (cc) es el carácter que se lee, (pop) es el tope de la pila (que al mismo tiempo quitamos de ella) y $(push)$ es lo que agregamos a la pila.

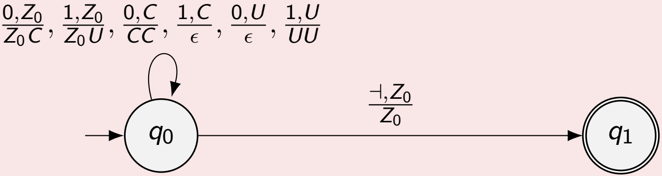
Por lo general, se trabaja con una pila con el primer elemento Z_0 y por esto no es usual ver al (pop) como ϵ. Igualmente, esto depende de como lo definamos y no es ninguna limitación para hacerlo.

Reglas de transición comunes

Regla	Transición
$A \rightarrow \epsilon$	$\dfrac{\epsilon, A}{\epsilon}$
$A \rightarrow a$	$\dfrac{a, A}{\epsilon}$
$A \rightarrow aC_1C_2...C_n$	$\dfrac{a, A}{C_n...C_2C_1}$
$A \rightarrow BC_1C_2...C_n$	$\dfrac{\epsilon, A}{C_n...C_2C_1B}$
$cc = b, \ top = b$	$\dfrac{bb}{\epsilon}$

Autómatas de pila

Formato de derivación



Probemos la cadena $x = 01001$:

$$
\begin{aligned}
< q0, x, Z_0 > &\mapsto < q0, 01001 \dashv, Z_0 > \\
&\mapsto < q0, 1001 \dashv, Z_0C > \\
&\mapsto < q0, 001 \dashv, Z_0 > \\
&\mapsto < q0, 01 \dashv, Z_0C > \\
&\mapsto < q0, 1 \dashv, Z_0CC > \\
&\mapsto < q0, \dashv, Z_0C >
\end{aligned}
$$

De autómatas de pila a gramáticas

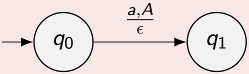
Generalidades del método

Definiremos los no terminales de una forma extraña. Entiéndase como una transición del estado q_a al q_b cuando el símbolo A está en el tope de la pila. Esto se escribirá $< q_a,\ A,\ q_b >$
Nuestro objetivo con este método será encontrar el axioma $< q_i,\ Z_0,\ q_f >$, siendo q_i el estado inicial y q_f el estado final.

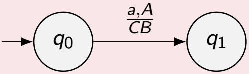
De autómatas de pila a gramáticas

Creación de la gramática

Usaremos estas transformaciones para definir cada transición existente:



$$< q_0, A, q_1 > \rightarrow \ a$$



$$< q_0, A, q_y > \rightarrow \ a < q_1, B, q_x >< q_x, C, q_y >$$

Para el último, q_x y q_y representan todos los estados que tenga el autómata. Deben probarse **todos** los casos posibles.

De autómatas de pila a gramáticas

Procesos adicionales

El trabajo final, luego de definir todas esas reglas, será limpiar la gramática, quitar la ambigüedad que pueda tener (reglas circulares) y, por supuesto, cambiar los nombres de los no terminales.