>>> IF013 - Fundamentos Teóricos de Informática
>>> Licenciatura de Sistemas - UNPSJB - Sede Trelew

Name: Celia Cintas[†], Pablo Navarro[‡], Samuel Almonacid[§]
Date: August 16, 2017

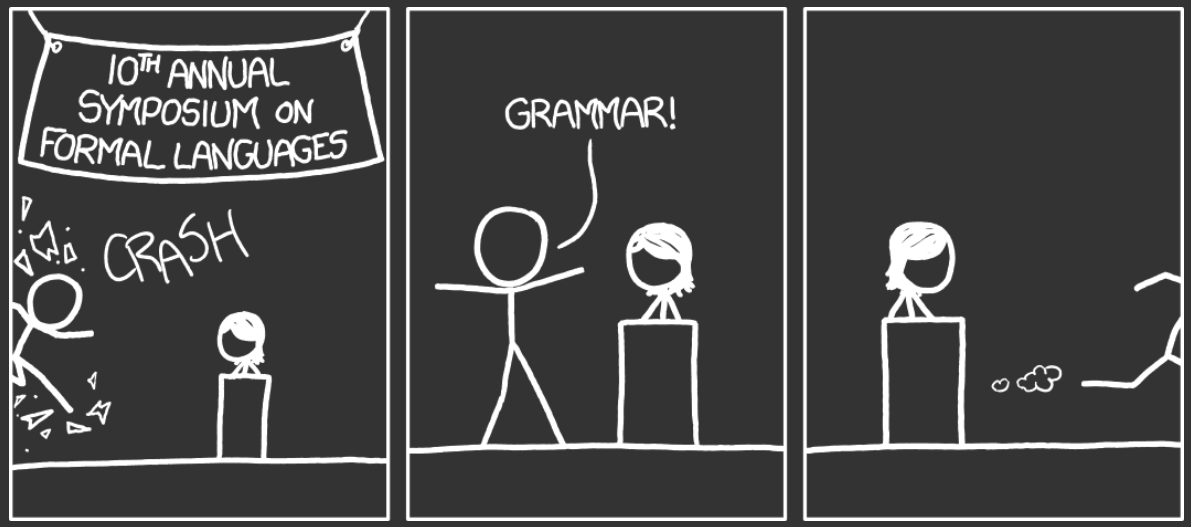
[†]cintas@cenpat-conicet.gob.ar, cintas.celia@gmail.com, @RTFMCelia
[‡]pnavarro@cenpat-conicet.gob.ar, pablo1n7@gmail.com
[§]almonacid@cenpat-conicet.gob.ar, almonacid.samuel.tw@gmail.com

>>> Unidad 2

1. Gramáticas libres de contexto. Clasificación de Chomsky.
2. Relación entre gramáticas regulares y libres de contexto.
3. Árboles de derivación. Ambigüedad. Formas normales de Chomsky, Greibach y Backus-Naur.
4. Autómata a pila. No determinismo. Relación entre los autómatas a pila y las gramáticas libres de contexto.
5. Propiedades de los lenguajes libres de contexto.
6. Sustituciones y Homomorfismos.
7. Usos y aplicaciones de los lenguajes libres de contexto.

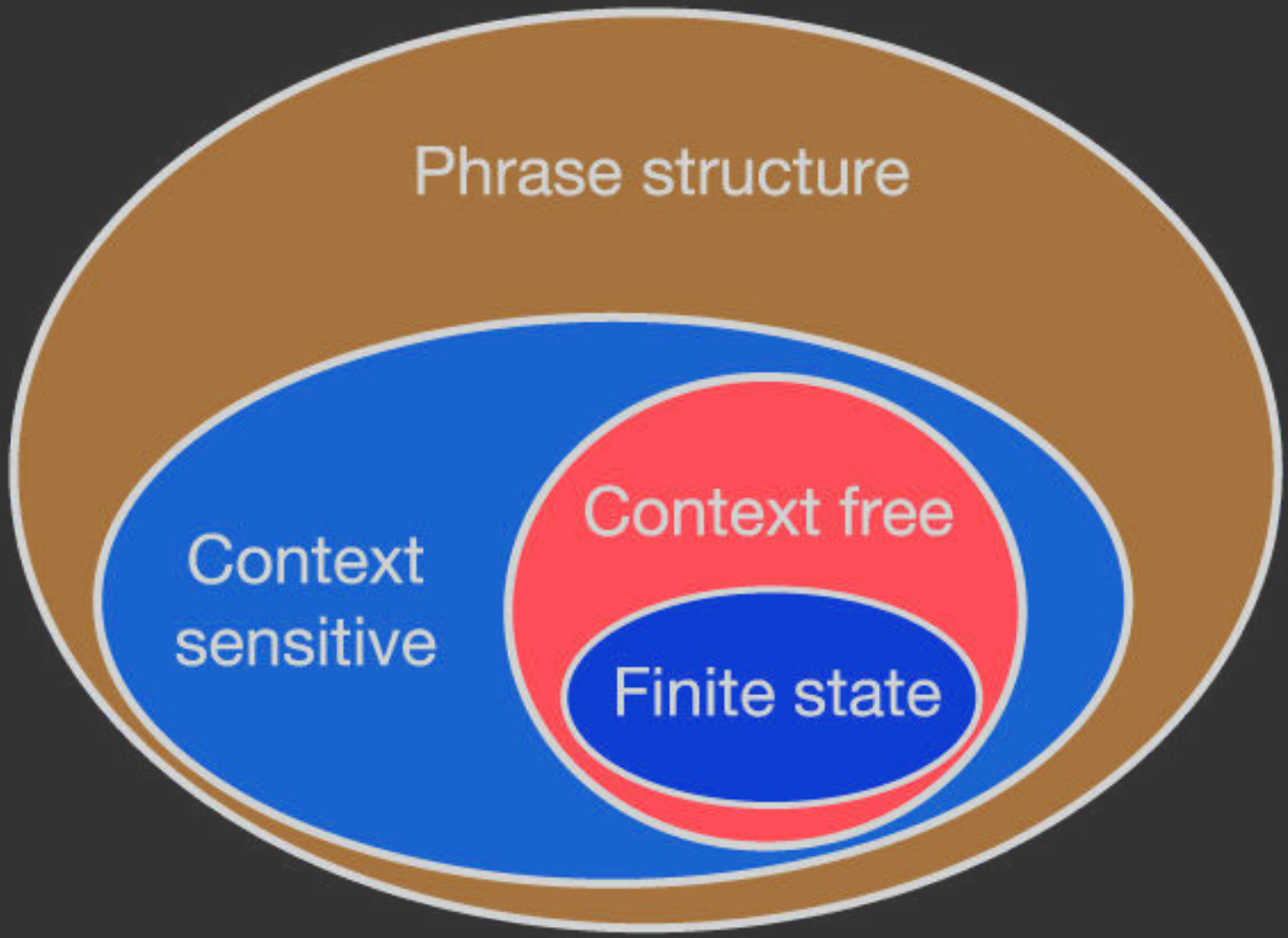
>>> Gramáticas Libres de Contexto

Los lenguajes libres del contexto (LLC) son importantes porque sirven como mecanismo formal para expresar la gramática de lenguajes de programación. Los DTDs usados para indicar el formato permitido en documentos XML. Se usan en biología computacional para modelar las propiedades que se buscan en secuencias de ADN o proteínas[1]. La construcción semiautomática de parsers eficientes, los cuales son esenciales en la construcción de compiladores e intérpretes.



[1] https://www.youtube.com/watch?v=pnHD8gvccpI

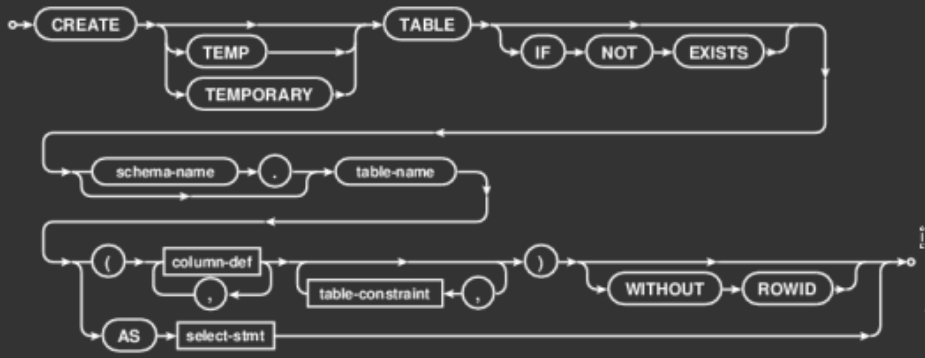
>>> Clasificación de Chomsky

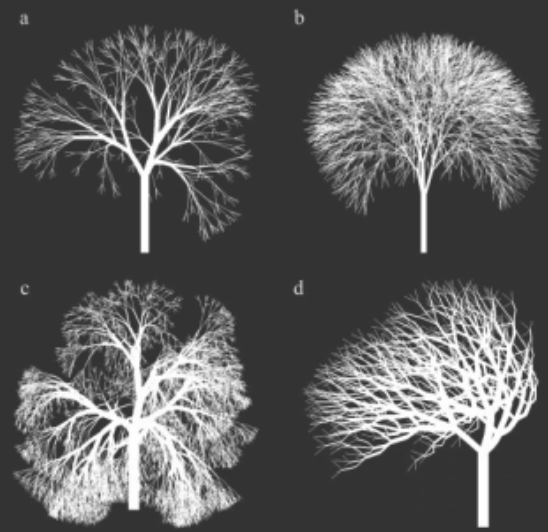


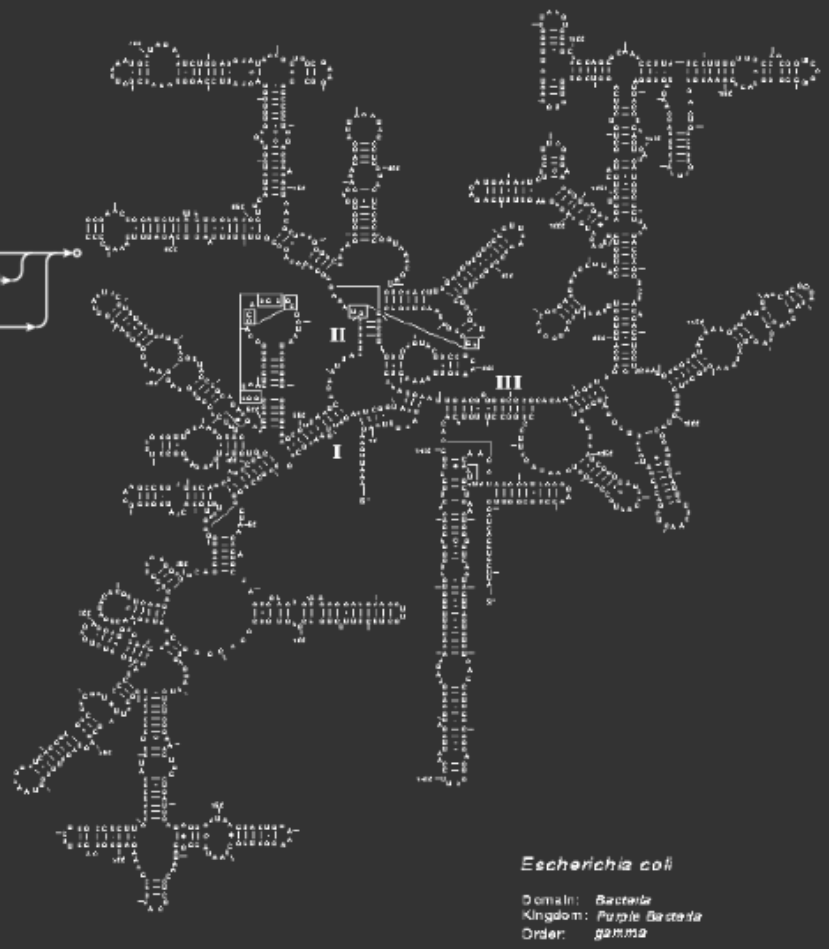
Computational and volutionary aspects of language. Martin A. Nowak, Natalia L. Komarova and Partha Niyogi, 2002. Nature.

>>> Gramáticas Libres de Contexto Tipo 2(Cont.)

```
import_stmt: import_name | import_from
import_name: 'import' dotted_as_names
# note below: the ('.' | '...') is necessary because '...' is tokenized as ELLIPSIS
import_from: ('from' (('.' | '...')* dotted_name | ('.' | '...')+)
              'import' ('*' | '(' import_as_names ')' | import_as_names))
import_as_name: NAME ['as' NAME]
dotted_as_name: dotted_name ['as' NAME]
import_as_names: import_as_name (',' import_as_name)* [',']
dotted_as_names: dotted_as_name (',' dotted_as_name)*
dotted_name: NAME ('.' NAME)*
```







Definición

Una gramática libre de contexto es una tupla $G = (V_n, V_t, P, S)$ donde:

1. V_n es un conjunto finito de símbolos no terminales.
2. V_t es un conjunto finito de símbolos terminales $(V_t \cap V_n = \emptyset)$.
3. $S \in V_n$, es el símbolo Start.
4. P es el conjunto finito de reglas de producción de la forma $\alpha \to \beta$ donde $\alpha \in V_n$ y $\beta \in (V_n \cup V_t)^+$.

Consideremos una gramática para generar el lenguaje $L = \{a^n b^n | n \geq 0\}$:

$$G = (V_n, V_t, S, P)$$
$$V_n = \{S, R\}$$
$$V_t = \{a, b\}$$
$$P = \{S \to \lambda, S \to R, R \to aRb, R \to ab\}$$

Definición

Un árbol D es un árbol de derivación para una gramática $G = (V_n, V_t, S, P)$ si (a) La raíz del árbol es rotulada con S (b) Cada nodo es rotulado con un símbolo de $V_n \cup V_t \cup \{\lambda\}$ (c) Si un nodo interior es rotulado A y sus descendientes directos son rotulados $x_1 x_2 \cdots x_n$, implica que existe una producción en G de la forma $A \to x_1 x_2 \cdots x_n$.

Definición

A los nodos del árbol, que no tienen descendientes se los denomina hojas. La frontera de un árbol de derivación es la cadena obtenida de la concatenación de los rótulos de las hojas de izquierda a derecha.

Veamos un ejemplo de árbol de derivación con el siguiente lenguaje:

$$G = (V_n, V_t, S, P)$$
$$V_n = \{S\}$$
$$V_t = \{(,)\}$$
$$P = \{S \to (), S \to SS, S \to (S)\}$$

Teorema

Sea $G = (V_n, V_t, S, P)$ una GLC, entonces para toda cadena $w \in L(G)$, $S \xrightarrow{G+} w$, si y solo si existe un árbol de derivación de la gramática cuya frontera es w.

$G = (V_n, V_t, S, P)$

$V_n = \{S\}$

$V_t = \{a\}$

$P = \{S \to aS, S \to Sa, S \to a\}$

Definición

Se dice que una gramática G libre de contexto es ambigua si existe al menos una cadena $w \in L(G)$ para la cual existe más de un árbol de derivación con frontera w.

Definición

Sea L un Lenguaje Libre del Contexto, si toda gramática G que genera L, es ambigua, entonces L es inherentemente ambiguo.

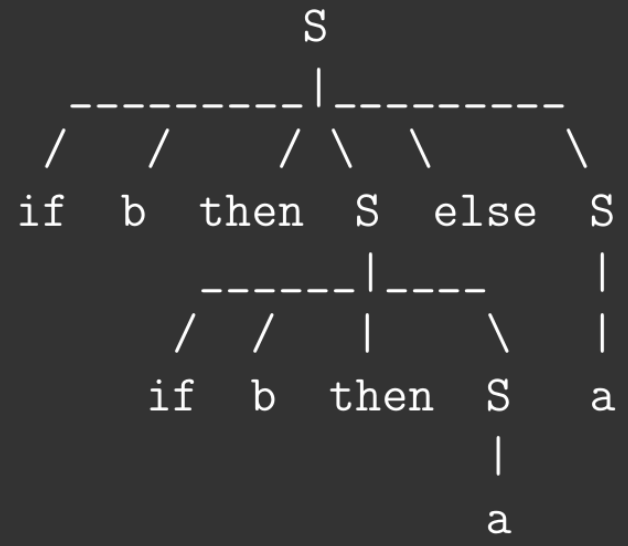
Veamos un ejemplo!

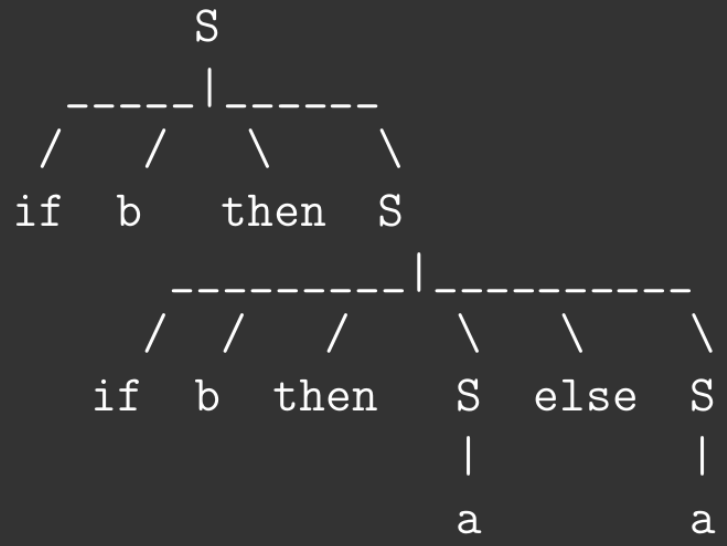
$G = (V_n, V_t, S, P)$

$V_t = \{if, then, else, a, b\}$

$V_n = \{S\}$

$P = \{S \rightarrow$ `if b then S else S`, $S \rightarrow$ `if b then S`, $S \rightarrow$ `a`$\}$





>>> Demo Time!

Gramáticas Libres de Contexto en Python - Ver ipynb

Definición

Un autómata a pila determinista (APD) P es una 6-upla $P = (S, \Sigma, \Gamma, \delta, s_0, F)$ donde:

S es el conjunto de estados, $S \neq \emptyset$.

Σ es el alfabeto de entrada.

Γ es el alfabeto de la pila.

$\delta : S \times (\Sigma \cup \{\lambda\}) \times \Gamma^* \to S \times \Gamma^*$.

$s_0 \in S$ y es el estado inicial del Autómata a Pila.

$F \subseteq S$ y es el conjunto de estados finales o aceptadores.

Definición

Sea un autómata a pila $P = (S, \Sigma, \Gamma, \delta, s_0, F)$, llamaremos configuración a cada terna $(s, w, \gamma) \in S \times \Sigma^* \times \Gamma^*$.

$$\delta(\underset{\text{estado corriente}}{s_0}, \underset{\text{símbolo leído}}{0}, \underset{\text{símbolo en pila, tope}}{\lambda}) = (\underset{\text{nuevo estado}}{s_1}, \underset{\text{acción sobre la pila}}{0})$$

Definición

Lenguaje reconocido por un autómata de pila (por estado final): $L(M) = \{\alpha \in \Sigma^* | (q_0, \alpha, Z_0) \vdash^* (p, \lambda, \gamma)\ y\ p \in F\}$

Definición

Aceptación por pila vacía. Sea M un autómata pila, definimos $L_\lambda(M)$ al lenguaje reconocido por el autómata de la siguiente forma:

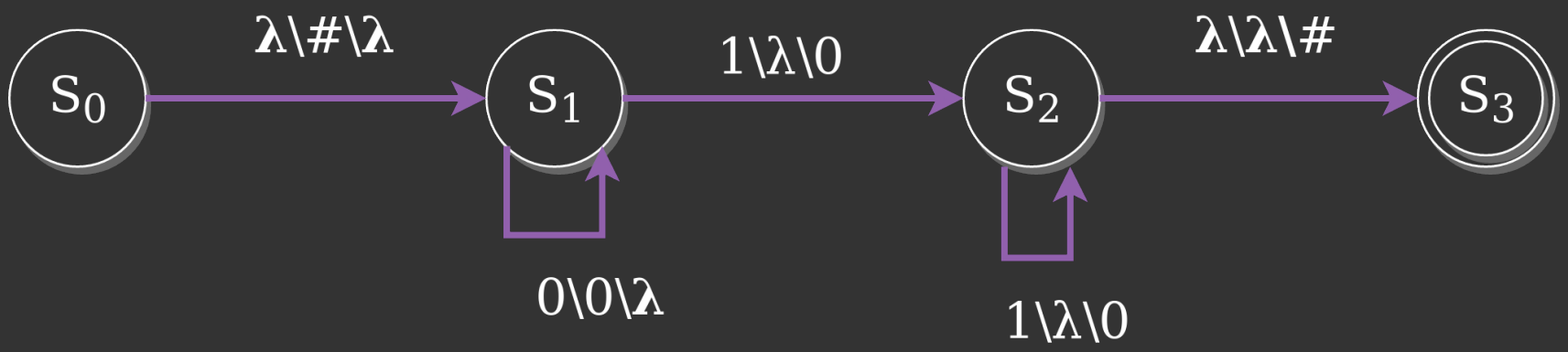
$L_\lambda(M) = \{\alpha \in \Sigma^* | (q_0, \alpha, Z_0) \vdash^* (r, \lambda, \lambda)\}$.

Es decir, que una cadena es aceptada si y solo si al terminar de procesarla, la pila está vacía.

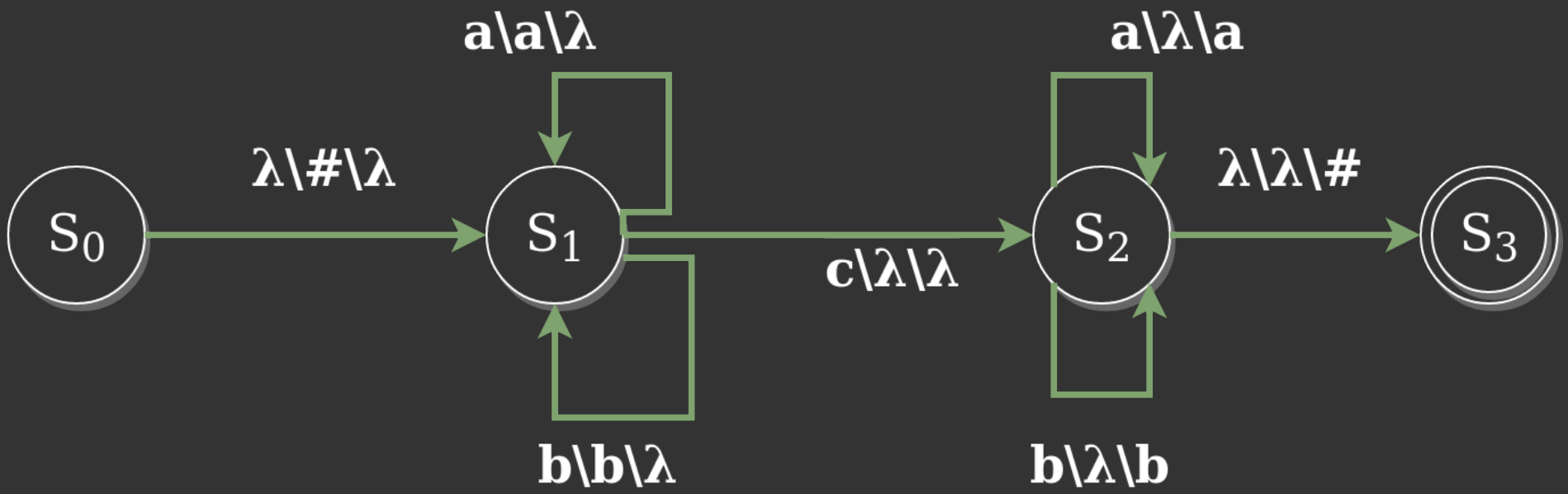
Definición

Sea P un autómata a Pila, el lenguaje aceptado por $L(P) = w|w\ es\ aceptada\ por\ P$.

Desarrollar un Autómata Pila para L, $L = \{0^n 1^n | n \geq 0\}$.
Sea $P = (S, \Sigma, \Gamma, \delta, s_0, F)$, $\Sigma = \{0, 1\}$, $S = \{s_0, s_1, s_2\}$, $\Gamma = \{0\}$, $F = \{s_0\}$:



$$L = \{w | wc\overleftarrow{w}, w \in \{a, b\}^*\}$$



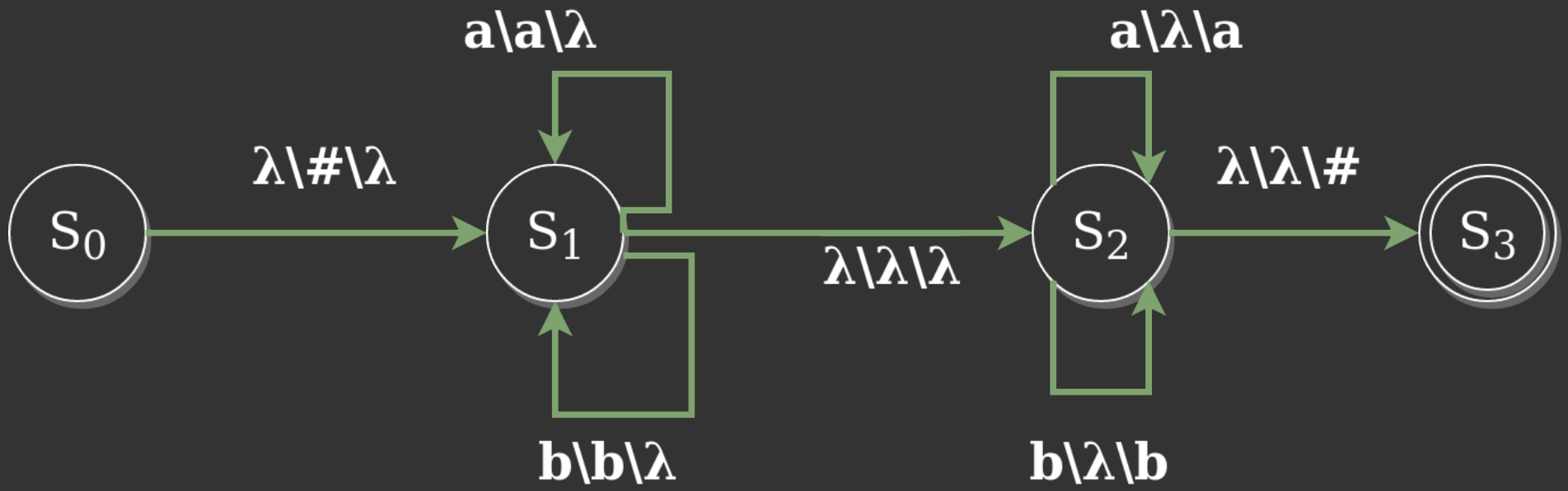
Definición

Un autómata a pila no determinista (APND) P es una 6-upla $P = (S, \Sigma, \Gamma, \delta, s_0, F)$ donde:

* S es el conjunto de estados, $S \neq \emptyset$.
* Σ es el alfabeto de entrada.
* Γ es el alfabeto de la pila.
* $\delta : (\Sigma \cup \{\lambda\}) \times \Gamma^* \to P(S \times \Gamma^*)$.
* $s_0 \in S$ y es el estado inicial del Autómata a Pila.
* $F \subseteq S$ y es el conjunto de estados finales o aceptadores.

$$L = \{w | w \overleftarrow{w}, w \in \{a, b\}^*\}$$

$$L = \{w | w \overleftarrow{w}, w \in \{a, b\}^*\}$$



Teorema

Los autómatas a pila no deterministas tienen mayor poder de reconocimiento de lenguajes que los autómatas a pila deterministas.

Demostración

Ver contraejemplo anterior analizado en pagina anterior.

Definición

Sea una gramática $G = (V_n, V_t, S, P)$ libre de contexto entonces $w \in L(G)$ si y solo si $w \in V_t$ y existe una derivación $S \to \omega_1 \to \omega_2 \to \omega_3 \cdots \to w$ *con cadenas* $\omega_i \in (V_n \cup V_t)^+$.

Definición

Diremos que una derivación es una derivación a izquierda si el símbolo no terminal reemplazado en cada paso es el no terminal ubicado más a la izquierda. $S \to SS \to ()S \to ()()$

Teorema

La clase de lenguajes aceptados por autómatas a pila no deterministas es exactamente la clase de lenguajes libres del contexto.

Definición

Para una gramática independiente de contexto G, puede definirse un autómata pila M que acepta el lenguaje generado por dicha gramática.
Sea $G = (V_n, V_t, P, S)$ una gramática libre de contexto tal que $L = L(G)$, definimos:
$M = (\{q_0\}, V_t, V_n \cup V_t, \delta, q_0, S, \emptyset)$, con

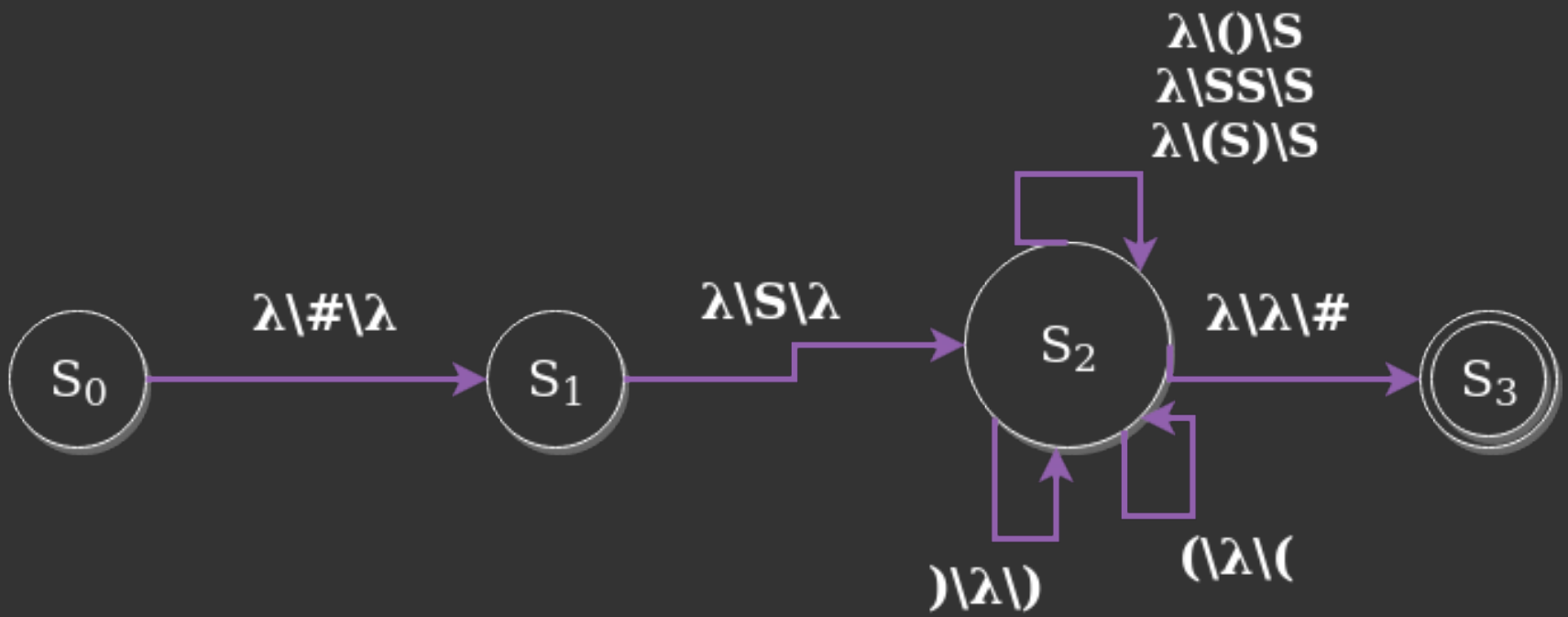
$$\delta(q_0, a, t) = \begin{cases} \{(q_0, \alpha) | t \to \alpha \in P\} \; si \; t \in V_n \; y \; a = \lambda \\ \{(q_0, \lambda)\} \; si \; t \in V_t \; y \; a = t \neq \lambda \end{cases}$$

Si en el tope de la pila hay un símbolo no terminal t, el autómata lo reemplazará por el lado derecho α de alguna producción del mismo. Si el tope de la pila hay un símbolo terminal t, el autómata chequeará que es igual al próximo símbolo en la cadena de entrada y lo desapilará.

Veamos un Ejemplo:

$$G = (V_n, V_t, S, P)$$
$$V_n = \{S\}$$
$$V_t = \{(,)\}$$
$$P = \{S \to (), S \to SS, S \to (S)\}$$

lee\apila\desapila



>>> Formas Normales

¿Cómo se puede parsear eficientemente un lenguaje? El hecho de que el no determinismo de los APs no sea superficial, indica que parsear eficientemente un lenguaje LC no es algo tan inmediato como para un lenguaje regular. Lo primero es un resultado que indica que es posible parsear cualquier lenguaje LC en tiempo polinomial.

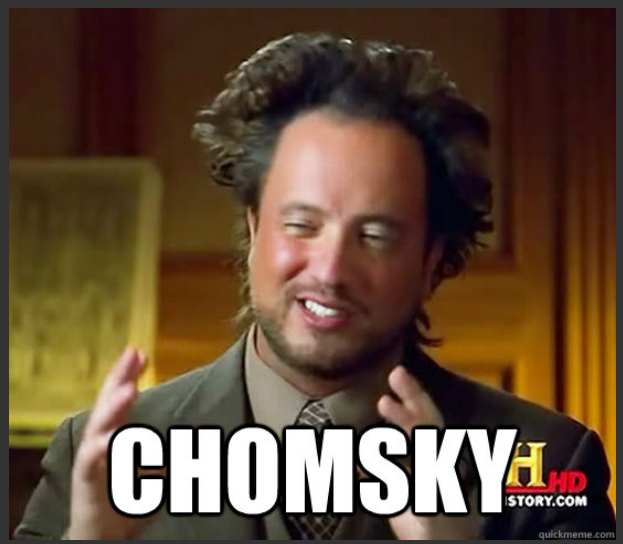
>>> Forma Normal de Chomsky

Definición

$SeaG = (V_n, V_t, S, P)$ una gramática libre del contexto, diremos que G respeta la Forma Normal de Chomsky si toda regla de P es de la forma:

$A \to BC$ o $A \to a$ o $S \to \lambda$, donde $A, B, C, S, \in V_n$ y $a \in V_t$.

La FNC estandarizan la forma de las GLC, generando únicamente árboles binarios y toda cadena de longitud n es derivable en $2n - 1$ pasos. Esta forma es utilizada en el algoritmo de CYK utilizado en análisis sintáctico de una cadena.



>>> Forma Normal de Chomsky (Cont.)

Teorema

Cualquier Lenguaje Libre del Contexto puede ser generado por una gramática que respeta la forma normal de Chomsky.

Demostración

La demostración es sugerida a traves del siguiente algoritmo que provee un método de transformar una gramática libre de contexto que respeta la notación de la Forma Normal de Chomsky.

>>> Forma Normal de Chomsky (Cont.)

Entrada: una GLC, $G = (V_n, V_t, S, P)$.
Salida: una GLC, $G = (V_n, V_t, S, P)$, en Forma Normal de Chomsky.

Mientras existan producciones que no respeta la FNC:

1. Reemplazar los pares de producciones de la forma $A \to B$ y $B \to w_1|\cdots|w_n$ por $A \to w_1|\cdots|w_n$.
2. Sea una producción del estilo $A \to ab$, crear una nueva producción $N \to a$ y $M \to b, N, M \notin V_n, V_n' = V_n \cup \{M, N\}$ y $P = P \cup \{N \to a, M \to b\}$.
3. Sea una producción $B \to C_1C_2\cdots C_n$ donde $n > 2$, reemplazar por nuevo no terminal : $B \to C_1D, D \to C_2\cdots C_n$

Sea la producción de una gramática libre de contexto G:

$$S \to A|aCaB|Ab|DCC$$
$$A \to aaA|\lambda$$
$$B \to Bbb|CBC$$
$$C \to CB|bB$$
$$D \to \lambda$$

1. Eliminar λ de prod. que no sean *Start*.
2. Eliminar producciones unitarias.
3. Renombrar producciones con más de dos términos no terminales.
4. Crear nuevas producciones para elementos terminales.

1. Eliminemos λ de producciones A y D.
   $S \to A|aCaB|Ab|CC|\lambda|b$
   $A \to aaA|aa$
   $B \to Bbb|CBC$
   $C \to CB|bB$
2. Eliminemos producciones unitarias ($S \to A$).
   $S \to aaA|aa|aCaB|Ab|CC|\lambda|b$
   $A \to aaA|aa$
   $B \to Bbb|CBC$
   $C \to CB|bB$
3. Generemos nuevas producciones para elementos terminales.
   $S \to A'A'A|A'A'|A'CA'B|AB'|CC|\lambda|b$
   $A \to A'A'A|A'A'$
   $B \to BB'B'|CBC$
   $C \to CB|B'B$
   $A' \to a$
   $B' \to b$

4. Renombrar no terminales que excedan el $n > 2$.

$S \to A''A|A'A'|DE|AB'|CC|\lambda|b$

$A \to A''A|A'A'$

$B \to BB''|CF$

$C \to CB|B'B$

$A'' \to A'A'$

$B'' \to B'B'$

$D \to A'C$

$E \to A'B$

$F \to BC$

$A' \to a$

$B' \to b$

Definición

Sea $G = (V_n, V_t, S, P)$ una gramática libre del contexto, diremos que G respeta la Forma Normal de Greibach si toda regla de P es de la forma:

$A \to aZ, A \in V_n, a \in V_t, Z \in V_n^*$

Definiciones menos estrictas permiten producciones del estilo $S \to \lambda$.

Teorema

Cualquier Lenguaje Libre del Contexto puede ser generado por una gramática que respeta la forma normal de Greibach.

http://www.cs.ucla.edu/sheila-greibach/

Entre las aplicaciones de la Forma Normal de Greibach se cuenta la ventaja de conocer que cada cadena de longitud n es derivable en n pasos.

Cómo transformar una G en FNC a FNG:
Partimos desde la suposición que ya pasamos a FNC. A partir de una GLC podemos descomponer el problema en dos pasos básicos.

1. Eliminar recursión a izquierda.
2. Agregar nuevas reglas de reescritura y no terminales para llevar las reglas existentes a la forma de Greibach.

Sea $P = \{S \to AB|B, A \to Aa|b, B \to Ab|c\}$ en FNC quedaría:
$P = \{S \to AB|AB'|c, A \to AA'|b, B \to AB'|c, A' \to a, B' \to b\}$

1. La regla $A \to AA'|b$ cuenta con recursión por izquierda. Podemos crear un nuevo conjunto de reglas que generen lo mismo sin esta recursión. $\{A \to b|B'D, D \to A'D|a\}$.
2. Ahora con $P = \{S \to AB|AB'|c, A \to b|B'D, B \to AB'|c, D \to A'D|a, A' \to a, B' \to b\}$ tratamos de llegar a la forma $A \to aZ_0Z_1 \cdots$.
   - 2.1 Sustituimos el llamado de A' y B': $P = \{S \to AB|AB'|c, A \to b|bD, B \to AB'|c, D \to aD|a, A' \to a, B' \to b\}$
   - 2.2 Sustituimos llamados de A: $P = \{S \to bB|bDB|bB'|bDB'|c, A \to b|bD, B \to bB'|bDB'|c, D \to aD|a, A' \to a, B' \to b\}$

Y nuestra P en FNG es:
$P = \{S \to bB|bDB|bB'|bDB'|c, A \to b|bD, B \to bB'|bDB'|c, D \to aD|a, A' \to a, B' \to b\}$

>>> Forma Normal Backus-Naur

Backus Normal Form o *Panini-Backus Form* es una metasintaxis usada para expresar gramáticas libres de contexto. BNF se utiliza extensamente como notación para las gramáticas de los lenguajes de programación, de los sistemas de comando y de los protocolos de comunicación. Existen variantes, tales como la *augmented Backus-Naur form* (ABNF).
Ejemplo, parte de gramática Python3:

```
stmt: simple_stmt | compound_stmt
simple_stmt: small_stmt (';' small_stmt)* [';'] NEWLINE
small_stmt: (expr_stmt | del_stmt | pass_stmt | flow_stmt |
import_stmt | global_stmt | nonlocal_stmt | assert_stmt)
```

Gramática completa en
https://docs.python.org/3/reference/grammar.html

>>> Forma Normal Backus-Naur (cont.)

BNF	Notación Clásica
::=	$\rightarrow$
|	| o +
<>	A,B, C
"" o ''	a,b,c

Ejemplo de gramática en notación BNF:

```
<expr> ::= <term> "+" <expr> | <term>
<term> ::= <factor> "*" <term> | <factor>
<factor> ::= "(" <expr> ")" | <const>
<const> ::= <integer>
<integer> ::= <digit>|<integer><digit>
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
"8" | "9"
```

>>> Extended Backus-Naur form

Significado	EBNF
definition	=
concatenation	,
termination	;
alternation	|
optional	[...]
repetition	...
grouping	(...)
terminal string	" ... "
terminal string	' ... '
comment	(* ... *)

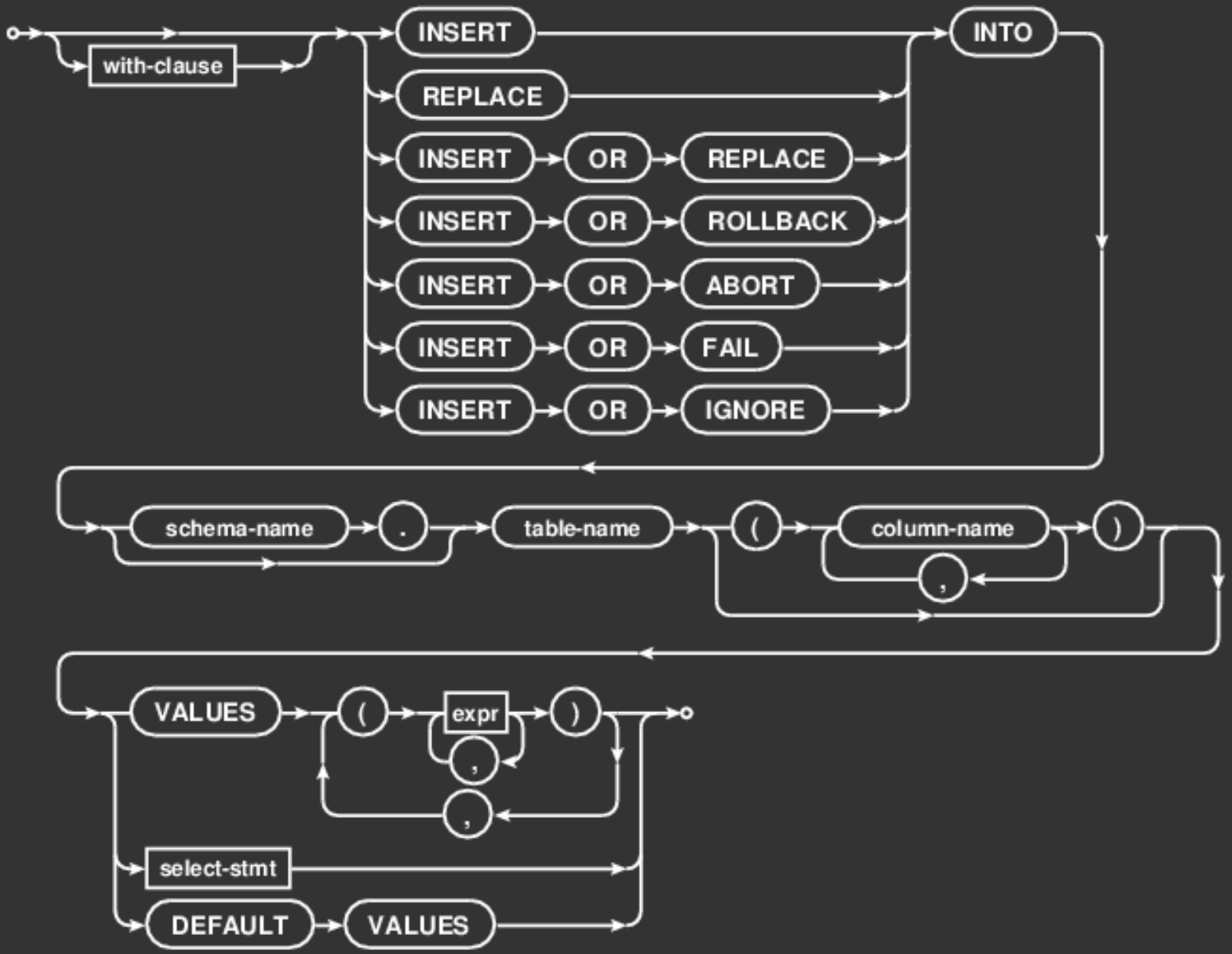
Esta denotación fue propuesta ISO/IEC 14977 por R. S. Scowen, pag 7, tabla 1.

>>> Diagramas de Sintaxis

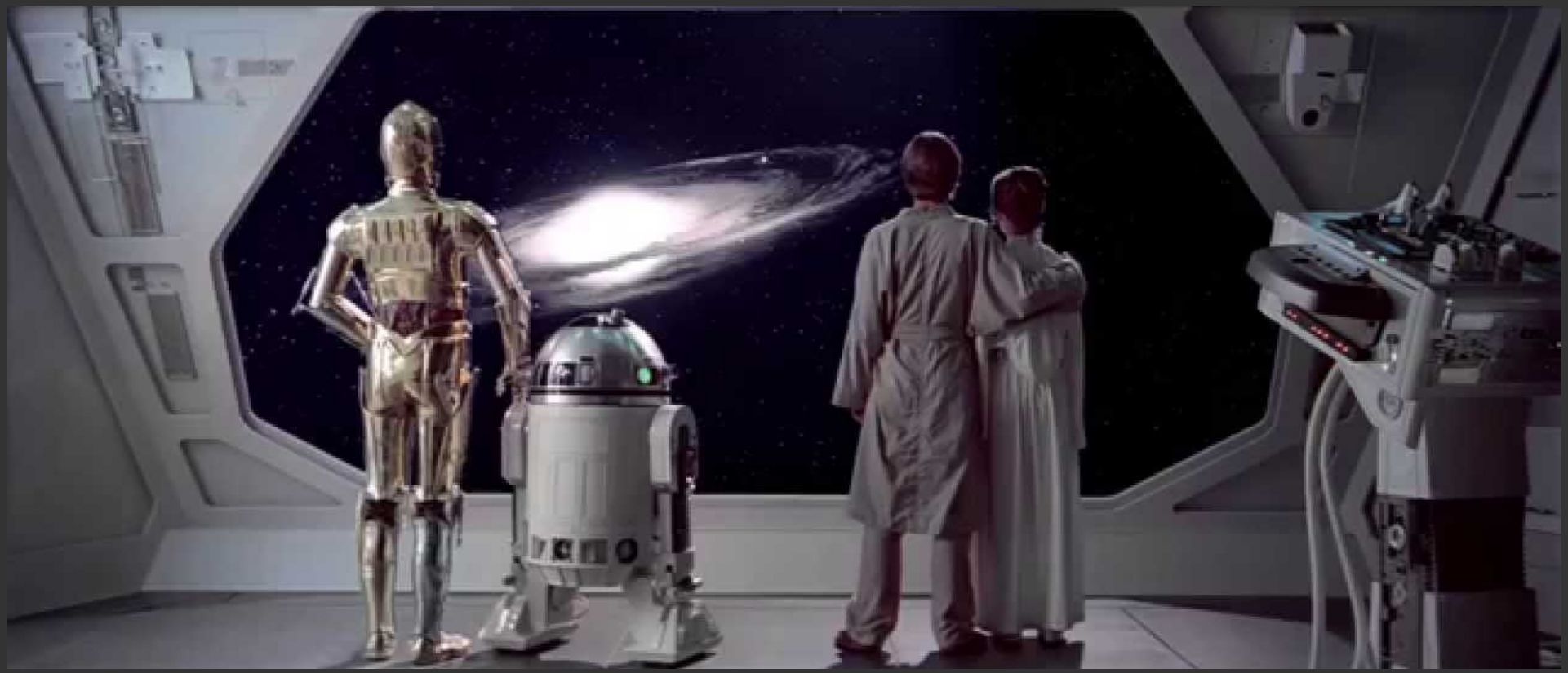
Los diagramas sintácticos son una forma gráfica de expresar la BNF extendida. Cada regla está representada por un camino que va desde la entrada ubicada a la izquierda, hasta la salida, ubicada a la derecha. Cualquier trayecto desde la entrada a la salida representa un string generado por esa regla. Las categorías sintácticas (no-terminales) se representan por rectángulos y los símbolos terminales por círculos u óvalos.

https://sqlite.org/syntaxdiagrams.html

>>> Diagramas de Sintaxis (Cont.)



>>> Gracias!



Bibliografía

1. Introduction to Automata Theory, Languages, and Computation - Hopcroft et. al 2007 (3er ed.)
2. Teoría de la Computación - Gonzalo Navarro 2011.
3. Fundamentos de Cs. de la Computación - Juan Carlos Augusto 1995.