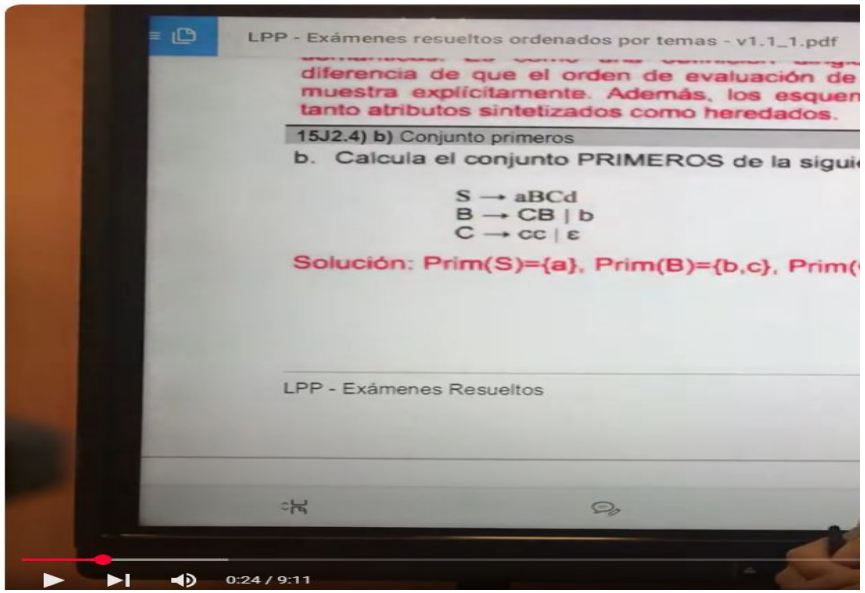
BIENVENIDOS

INICIAMOS EN BREVE

Universidad Tecnológica del Perú

Video



https://www.youtube.com/watch?v=mL9xLp9T5BU

CURSO: TEORÍA DE LENGUAJES DE PROGRAMACIÓN II

TEMA: Análisis Sintáctico Descendente

Docente: Mg. Ing. Delgado Enríquez Héctor

Universidad Tecnológica del Perú

Cronología de la Clase
Secuencia y explicación

Inicio

Actividad que permite llamar la atención:Caso
Aplicar una dinámica(Romper el hielo)

Cierre

Utilidad

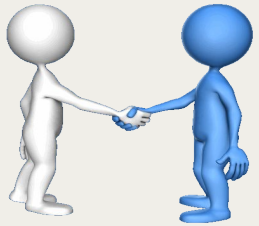
Práctica

Transformación

En Clase:

SEA RESPETUOSO

Al comunicarse con el docente y compañeros en la clase, chat, foros y en videoconferencias, respete las opiniones de los demás.



CUIDE SU REDACCIÓN

Evite escribir en mayúscula sostenida al comunicarse en los foros, chat, correo, entre otros canales de comunicación porque denota grito.

ACTÚE CON ÉTICA

Preséntese con su identidad real.
Respete los derechos de autor.
Comparta contenidos que no atenten contra la dignidad.

Levante la mano para participar

Por favor
Gracias
De nada

El tono de voz es clave en el éxito de la comunicación

Sistema de Evaluación

LA NOTA PUBLICADA EN LA SEMANA 10, ESTARÁ COMPUESTA POR:

1.- Actividades desarrolladas en clase	30%
2.- Tareas	
3.- Participación en clase – Excel (Individual y grupal)	10%
4.- Exposición de investigación	25%
5.- pc	35%
*Nota semana 10:	100%

*El estudiante que solo asista a las clases del examen, estará desaprobado; porque la nota, es el promedio de las actividades que se desarrollan en cada clase.

Universidad Tecnológica del Perú

Comentarios Positivos

Generan una sesión amigable

¿Cómo te sientes hoy?

El corazón del entendido adquiere sabiduría;
Y el oído de los sabios busca la ciencia.

Prov. 18:15

Cronología de la Clase
Secuencia y explicación

Inicio

Cierre

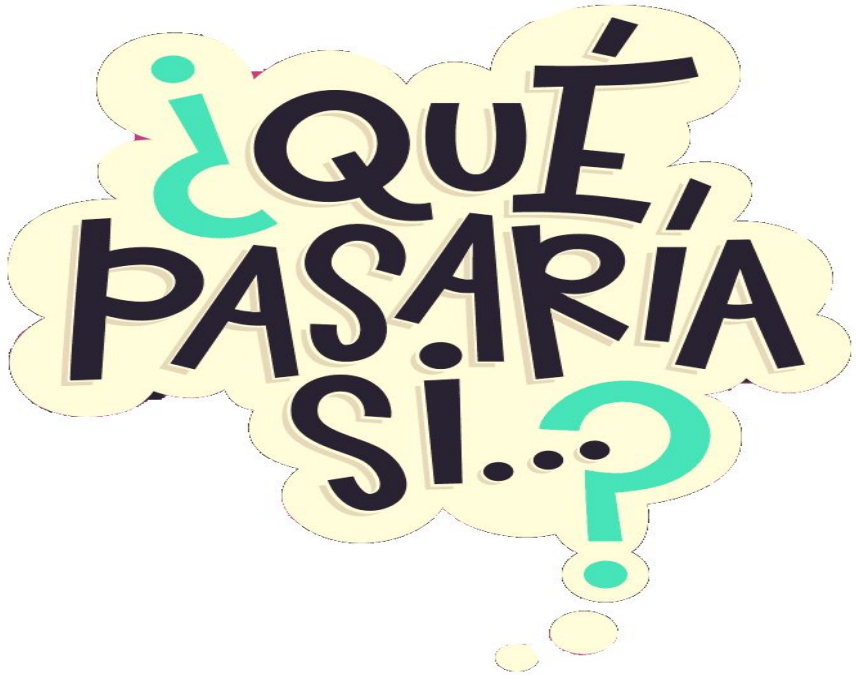
Utilidad

Recolectar saberes previos: Caso/Lluvia de ideas de la clase anterior
Logro de la sesión

Práctica

Transformación

Utilidad



… no existiera un análisis sintáctico descendente?

Sin el análisis sintáctico descendente, los compiladores perderían la capacidad de validar jerarquías gramaticales de forma ordenada, lo que generaría inconsistencias en la estructura del código. Este método es clave para analizar construcciones lingüísticas desde la raíz hasta las hojas, asegurando coherencia en la sintaxis. Su ausencia dificultaría el procesamiento de lenguajes con reglas anidadas o recursivas, afectando la eficiencia del compilador.

¿Por qué es importante el tema de la clase de hoy, para mi desarrollo profesional?

Logro

Al finalizar la sesión, el estudiante aplica el análisis sintáctico descendente, a través de la explicación del docente y actividades grupales.

tiempo, sujeto, verbo, contenido y condición (conoce/identifica/comprende).
1 verbo

Tema Anterior

- PC

Tema Actual

- Análisis Sintáctico Descendente

Recordando la clase pasada

Tipos de notación

- **Regex (1956)** → Primero en surgir, para teoría de autómatas.
- **BNF (1959)** → Base de las gramáticas formales.
- **Syntax Diagrams y BNFE (~1960s)** → Extensiones visuales y sintácticas.
- **EBNF (1971)** → Estandarización ISO de BNFE.

Recordando | Tipos de notación

Categoría	Regex (1956)	BNF (1959)	Syntax Diagrams (~1960s)	BNFE (~1960s)	EBNF (1971)
Autor	Stephen Kleene	John Backus y Peter Naur	Varios (IBM, Nassi-Shneiderman)	Derivada de BNF (autores no oficiales)	Niklaus Wirth / ISO
Concepto	Patrones para búsqueda de cadenas	Notación para gramáticas libres de contexto	Representación visual de gramáticas	BNF con extensiones	Estandarización de BNFE con sintaxis clara
Sintaxis	Metacaracteres: *, +, ?, |, [ ], ( )	<simbolo> ::= producción	Nodos y flechas gráficas	Similar a BNF + { }, [ ], ( )	simbolo = producción (usa { }, [ ], ( ), |)
¿Para qué se usa?	Tokenización y búsqueda de patrones	Definir gramáticas formales	Visualizar estructuras sintácticas	Simplificar gramáticas complejas	Gramáticas legibles y estandarizadas
Repeticiones	* (0 o más), + (1 o más), {n,m}	Recursión (ej: <lista> ::= <elem> | <lista> <elem>)	Bucles en diagramas	{ elemento } (ej: { <digito> })	{ elemento }
Opciones	? (0 o 1 vez)	No directo (simulado con reglas alternativas)	Ramas opcionales	[elemento] (ej: [<signo>] <número>)	[elemento]
Alternativas	| (ej: a|b)	| (ej: <expr> ::= <suma> | <resta>)	Múltiples caminos	|	|
Agrupación	( ) (ej: (ab)+)	No soportada directamente	No aplicable	( ) (ej: (<A> <B>) | <C>)	( )
Legibilidad	Compacta, pero críptica para patrones complejos	Menos intuitiva para estructuras complejas	Alta (visual), pero no textual	Más compacta que BNF	Alta (sintaxis clara y estandarizada)
Casos de uso, Uso típico	- Lexers (Flex) Tokenización y validación de cadenas	- Algol, Pascal - Documentación formal Gramáticas antiguas	- IDEs (ej: Eclipse) Diseño previo de gramáticas	- Protocolos de red Lenguajes modernos con sintaxis compleja	- Pascal, Ada - Herramientas (ANTLR, Yacc)
Ejemplo	[0-9]+ (uno o más dígitos)	<número> ::= <dígito> | <número> <dígito>	Diagrama con flujo para if-else	<número> ::= ["-"] { <dígito> }	número = ["-"], { dígito };

Recordando | DIFERENCIAS ENTRE BNF Y BNFE

Característica	BNF (Backus-Naur Form)	BNFE (Backus-Naur Form Extended)
Definición	Notación original para gramáticas libres de contexto (1959).	Extensión de BNF que añade símbolos para repeticiones y opciones (~1960s).
Sintaxis básica	<símbolo> ::= producción	Similar a BNF, pero con { }, [ ], y ( ).
Repeticiones	Se usa recursión: <lista> ::= <elem> | <lista> <elem>	Usa llaves { }: <lista> ::= <elem> { <elem> }
Elementos opcionales	No existe soporte directo. Se simula con reglas alternativas: <opt> ::= ε | <elem>	Usa corchetes [ ]: <opt> ::= [<elem>]
Agrupación	No soportada.	Usa paréntesis ( ) para agrupar: <expr> ::= (<A> | <B>) <C>
Legibilidad	Menos intuitiva para gramáticas complejas (requiere más reglas).	Más clara y compacta, especialmente en estructuras repetitivas o anidadas.
Ejemplo de gramática	<número> ::= <dígito> | <número> <dígito>	<número> ::= ["-"] { <dígito> }
Ventajas	- Base teórica sólida. - Ideal para explicar conceptos fundamentales.	- Reduce verbosidad. - Evita recursión innecesaria. - Usada en lenguajes modernos (XML, JSON).
Desventajas	- Verbosa para gramáticas complejas. - Difícil mantenimiento.	- Menos estandarizada que EBNF (ISO). - Algunas herramientas no la soportan directamente.
Uso en compiladores	- Gramáticas antiguas (Algol, Pascal). - Fines educativos.	- Lenguajes modernos (protocolos, DSLs). - Herramientas como ANTLR.

BNF	BNFE
Es recursivo y minimalista; Es más teórico;	BNFE añade azúcar sintáctico ({ }, [ ]). BNFE es práctico en implementaciones reales. Elimina la necesidad de reglas redundantes Ej: repeticiones sin recursión.
Ejemplo práctico: BNF (recursión): <expresión> ::= <término> | <expresión> "+" <término>	**Ejemplo práctico:** BNFE (con operadores): <expresión> ::= <término> { "+" <término> }

Verificación del logro

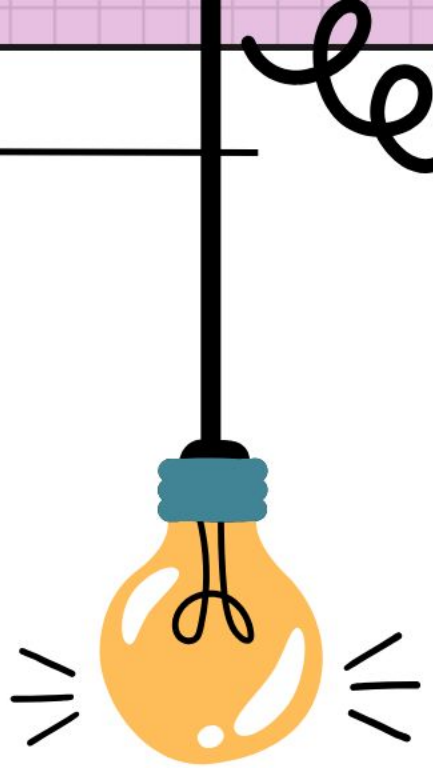
Responde verdad (V) o falsedad (F)

Notación Backus-Naur

¿Es un metalenguaje, que se usa para describir otro lenguaje?

V
F

LLUVIA de IDEAS



1.- Con sus propias palabras explique:

¿Para qué sirve el Análisis Sintáctico Descendente?

Mencione una diferencia entre: Análisis Sintáctico Descendente de izquierda a derecha, de derecha a izquierda, con retroceso y determinista.

Responde en el excel que compartimos en línea | Retroalimentación continua

Caso CodeGenius

La empresa CodeGenius, especializada en el desarrollo de entornos de desarrollo integrados (IDEs), se enfoca en crear herramientas de programación inteligentes con soporte para múltiples lenguajes. Su objetivo es mejorar la productividad de los desarrolladores mediante análisis de código en tiempo real y sugerencias de autocompletado avanzado.

Problema:
El equipo detectó que su IDE tenía lentitud al validar sintaxis en lenguajes con gramáticas complejas (como C++ o Python), generando falsos errores y retardando el análisis. Necesitaban un parser eficiente que procesara el código de manera predictiva, evitando retrocesos innecesarios en ciertos casos y optimizando el tiempo de compilación.

Resultados
Tras implementar un análisis sintáctico descendente determinista (LL(1)) para estructuras predecibles y uno con retroceso (recursivo por descenso) para casos ambiguos, lograron más de velocidad en el análisis. El IDE mejoró su precisión en la detección de errores y redujo el consumo de memoria, optimizando la experiencia del usuario.

¿Cómo lo hizo?:

Se diseñó un parser híbrido: para construcciones simples (como asignaciones o bucles), se usó un enfoque descendente izquierda-derecha determinista (LL(1)), aprovechando tablas de predicción para evitar retrocesos.
En casos ambiguos (ej: expresiones anidadas), se aplicó descenso recursivo con retroceso, permitiendo analizar alternativas.
Se priorizaron gramáticas factorizadas para eliminar ambigüedades y se optimizó el lookahead para decisiones rápidas.

La implementación demostró que combinar técnicas descendentes (deterministas y con retroceso) brinda flexibilidad y eficiencia en el análisis sintáctico.
Las ventajas clave fueron:
menor overhead computacional, mayor precisión en diagnósticos y escalabilidad para nuevos lenguajes.

¿Cuáles son las ventajas del presente caso?

Cronología de la Clase
Secuencia y explicación

Inicio

Cierre

Utilidad

Práctica

Transformación

Desarrollar los contenidos(Secuencia y explicación)
Realizar ejemplos para que comprendan la información de manera activa.
Fomentar la participación(Salir a la pizarra a resolver ejercicios)

Juego de Roles

Docente	 Estudiante 1	 Estudiante 2
Hoy analizaremos el Análisis Sintáctico Descendente. ¿Qué saben sobre los tipos de parsers descendentes?	El descendente recursivo por descenso (top-down) construye el árbol desde la raíz, expandiendo no terminales hasta llegar a los tokens.	También existen los parsers LL(k), que usan lookahead para decidir producciones sin retroceso, siendo deterministas si k=1.
Exacto. ¿Y cómo manejan los conflictos en gramáticas ambiguas?	Con retroceso (backtracking), el parser prueba alternativas hasta encontrar una derivación válida, aunque es costoso computacionalmente.	En cambio, los parsers deterministas (LL(1)) evitan retroceso con gramáticas factorizadas y predicciones basadas en tablas.
Profundicemos: ¿Qué ventajas tiene un parser descendente de izquierda a derecha frente a uno de derecha a izquierda?	El de izquierda a derecha (como LL) es más intuitivo para lenguajes con precedencia izquierda, mientras que el de derecha a izquierda (como algunos parsers LR) puede ser útil para operadores asociativos derechos	Pero en descendente puro, la derecha-izquierda es rara; suele usarse en combinación con técnicas ascendentes para manejar recursividad derecha
Excelente. Ahora, usen metáforas para explicar estos conceptos	Un parser con retroceso es como un laberinto: si tomas un camino equivocado, debes volver atrás y probar otro hasta hallar la salida	Un parser determinista LL(1) es como un tren con rieles fijos: sigue una ruta predecible sin desviaciones, gracias a las señales (lookahead)

El análisis sintáctico descendente (LL, recursivo, con/sin retroceso) es crucial en compiladores e IDEs para validar eficientemente la estructura del código. Técnicas deterministas (LL(1)) optimizan rendimiento, mientras el retroceso maneja ambigüedades a costa de mayor complejidad-

Análisis Sintáctico Descendente (Izq-Der, Der-Izq, con Retroceso y Determinista)

Indicador	Análisis Sintáctico Descendente (Izq-Der, Der-Izq, con Retroceso y Determinista)
Breve Historia	Surgió en los años 1960-1970 como parte de la teoría de lenguajes formales y compiladores. Se desarrolló para mejorar la eficiencia en el parsing de gramáticas libres de contexto.
Origen	Basado en la teoría de autómatas y gramáticas formales (Noam Chomsky). Formalizado por investigadores como Donald Knuth (LR) y Alfred Aho (LL).
¿Quién lo inventó?	Conceptos clave desarrollados por **Alfred Aho** (LL) y **Donald Knuth** (LR), aunque el enfoque descendente se basa en trabajos previos de teoría de parsing.
Concepto	Técnica de parsing que deriva el árbol sintáctico desde el símbolo inicial (raíz) hasta las hojas (tokens), usando reglas gramaticales. Puede ser: **LL(k)** (determinista), **recursivo con retroceso** o híbrido.
Características	- **LL(k)**: Lookahead de *k* tokens para decisiones sin retroceso. - **Con retroceso**: Reintenta producciones si falla. - **Izq-Der vs Der-Izq**: Predominio de izquierda en LL, derecha en casos especiales (operadores).
Ventajas	- Fácil implementación (especialmente recursivo por descenso). - Eficiente para gramáticas simples (LL(1)). - Legibilidad y mantenibilidad.
Desventajas	- Retroceso es costoso en tiempo. - Limitado a gramáticas no ambiguas (LL(k)). - No maneja bien recursividad izquierda.
Listado de programas similares	ANTLR, Yacc/Bison (ascendente LR), JavaCC, PEG parsers (Packrat).

Tabla Comparativa con otros Parsers

Parser	Tipo	Ventajas	Desventajas
LL(k)	Descendente	Simple, rápido para gramáticas LL.	No maneja recursividad izquierda.
LR(k)	Ascendente	Más poderoso (gramáticas complejas).	Complejidad de implementación.
PEG	Descendente	Sin ambigüedades (prioridad definida).	Menos eficiente en memoria.
Recursivo con Retroceso	Descendente	Flexible (gramáticas ambiguas).	Ineficiente en tiempo.

Diferencia entre Análisis LL y LR

Los términos LL y LR son dos estrategias fundamentales en el análisis sintáctico de lenguajes de programación.

Análisis LL (Left-to-right, Leftmost derivation)	**Análisis LR (Left-to-right, Rightmost derivation)**
Tipo: Descendente (construye el árbol de arriba hacia abajo). **Dirección de lectura:** Izquierda a derecha (Left-to-right). **Derivación**: Leftmost (expande el no terminal más a la izquierda primero). Lookahead: Usa k símbolos para tomar decisiones (ej: LL(1), LL(k)). Gramáticas soportadas: Solo gramáticas no ambiguas y sin recursión izquierda	**Tipo**: Ascendente (construye el árbol de abajo hacia arriba). **Dirección de lectura:** Izquierda a derecha (Left-to-right). **Derivación**: Rightmost (reduce la producción más a la derecha primero). **Lookahead**: Usa k símbolos (ej: LR(0), SLR(1), LALR(1), LR(1)). **Gramáticas soportadas:** Gramáticas más generales (incluyendo recursión izquierda).
Ventajas: Fácil de implementar (ej: parser recursivo por descenso). Eficiente para lenguajes simples. **Desventajas:** No maneja gramáticas ambiguas o con recursión izquierda. Menos potente que LR. **Ejemplo de uso:** Lenguajes como Python y Pascal usan parsers LL(1). Herramientas: ANTLR (modo LL(*)), JavaCC.	**Ventajas**: Más poderoso que LL (puede analizar más gramáticas). Eficiente en tiempo y memoria para lenguajes complejos. **Desventajas**: Más difícil de implementar (requiere tablas de parsing). Genera parsers más grandes. **Ejemplo de uso:** Compiladores como GCC (usando Bison/Yacc). Lenguajes como C, Java, Rust usan parsers LR/LALR.

Tabla Comparativa LL vs LR

Criterio	LL (Descendente)	LR (Ascendente)
Tipo de parsing	Top-down (raíz a hojas)	Bottom-up (hojas a raíz)
Derivación	Leftmost	Rightmost (en reversa)
Lookahead	Necesita k símbolos (LL(k))	Usa k símbolos (LR(k))
Gramáticas	No recursión izquierda	Soporta recursión izquierda
Implementación	Más simple (recursivo)	Compleja (tablas de estados)
Rendimiento	Rápido para gramáticas simples	Más eficiente en gramáticas complejas
Herramientas	ANTLR, JavaCC	Yacc/Bison, GNU Flex

- LL es ideal para lenguajes simples y desarrollo rápido (ej: DSLs).
- LR es mejor para lenguajes complejos (ej: C++, Java).
- LL(k) es más intuitivo, pero LR(k) es más potente.

Si estás diseñando un lenguaje nuevo, ANTLR (LL(*)) es una buena opción para prototipado, mientras que Bison/Yacc (LALR) es estándar en compiladores profesionales.

tabla comparativa detallada que diferencia los cuatro tipos de análisis sintáctico descendente:

Característica	Descendente Izq-Der	Descendente Der-Izq	Con Retroceso	Determinista (LL)
Dirección de parsing	De izquierda a derecha	De derecha a izquierda	Izq-Der (común)	Izq-Der
Estrategia	Expande no terminales izquierdos	Expande no terminales derechos	Prueba alternativas hasta éxito	Usa lookahead para decisiones
Gramáticas soportadas	No recursión izquierda	Gramáticas especiales	Gramáticas ambiguas	Solo gramáticas LL(k)
Complejidad	O(n) para LL(1)	Rara vez implementado	Exponencial en peor caso	O(n) para LL(1)
Uso de memoria	Bajo	Bajo	Alto (mantiene estados)	Bajo
Implementación típica	Recursivo por descenso	Casi no se usa	Backtracking con pila	Tablas de parsing
Ventajas	Simple y eficiente	Útil para operadores derechos	Maneja ambigüedades	Rápido y predecible
Desventajas	No maneja recursión izquierda	Limitado en aplicaciones	Muy ineficiente	Gramáticas restringidas
Ejemplo de uso	Python (ANTLR)	Operadores unarios derechos	Prolog	Pascal, JavaCC
Lookahead	Opcional (LL(k))	No aplicable	No aplicable	Requerido (k símbolos)
Tolerancia a errores	Moderada	Baja	Alta (reintenta)	Baja (falla rápido)

Recuerda

Izq-Der (LL): El estándar en parsers descendentes (ej: ANTLR). Avanza linealmente expandiendo la izquierda primero.

Der-Izq: Raro en la práctica, útil para gramáticas con asociatividad derecha (ej: x = y = z).

Con retroceso: Versátil pero costoso. Usado en lenguajes lógicos (Prolog) o gramáticas ambiguas.

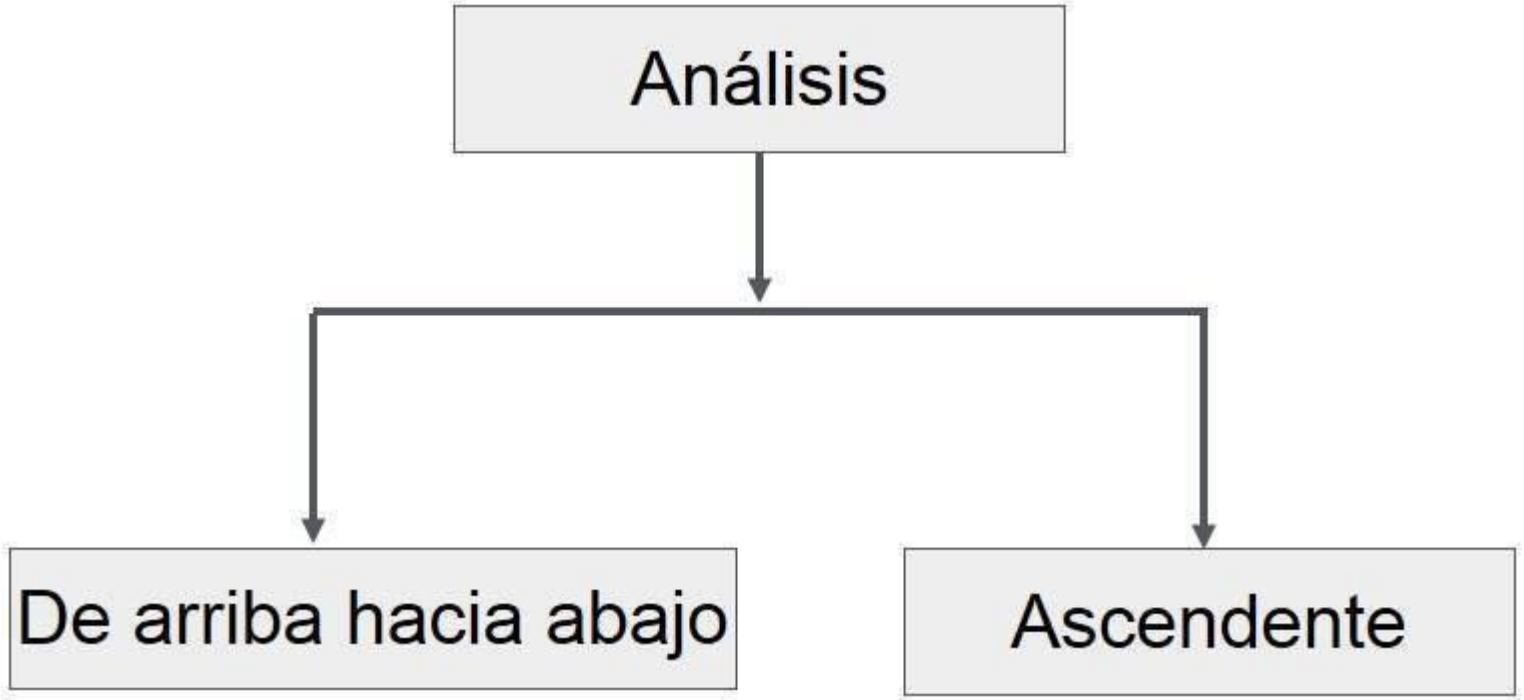
Determinista (LL): El más eficiente, pero requiere gramáticas bien estructuradas (sin ambigüedades).

Características del Análisis Sintáctico

- Lee componentes léxicos (tokens).
- Comprueba que el orden de estos corresponde a la sintaxis predeterminada.
- Genera errores en caso de que el flujo de tokens no responda a la sintaxis.
- Genera árboles de análisis sintáctico.
- Se suele conocer como “Parser”.
- El análisis sintáctico desarrolla el esqueleto de toda la fase de análisis
- Utiliza el analizador léxico como una rutina dentro del análisis sintáctico (getNextToken()).
- Integra el análisis semántico como un conjunto de rutinas a ejecutar durante la comprobación de la sintaxis.
- El análisis sintáctico se especifica mediante una gramática libre de contexto

Tipos de Análisis Sintáctico

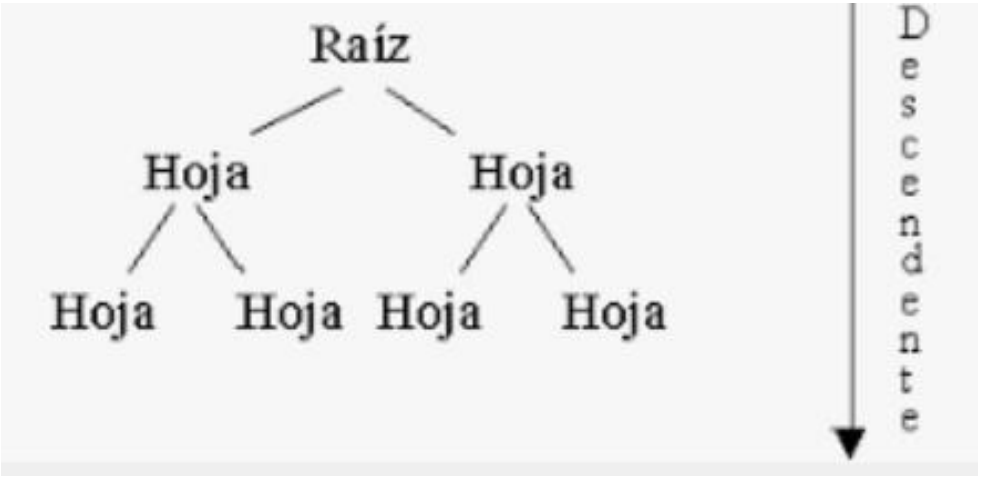
La forma en que las normas de producción se aplican (derivación) divide el análisis en dos tipos: de arriba a abajo y el análisis bottom-up análisis.



Análisis Sintáctico Descendente

Los analizadores sintácticos descendentes son lo que construyen el árbol sintáctico de la sentencia a reconocer de una forma descendente, comenzando por el símbolo inicial o raíz, hasta llegar a los símbolos terminales que forman la sentencia.

El analizador sintáctico descendente intenta encontrar entre las producciones de la gramática la derivación por la izquierda del símbolo inicial para una cadena de entrada.



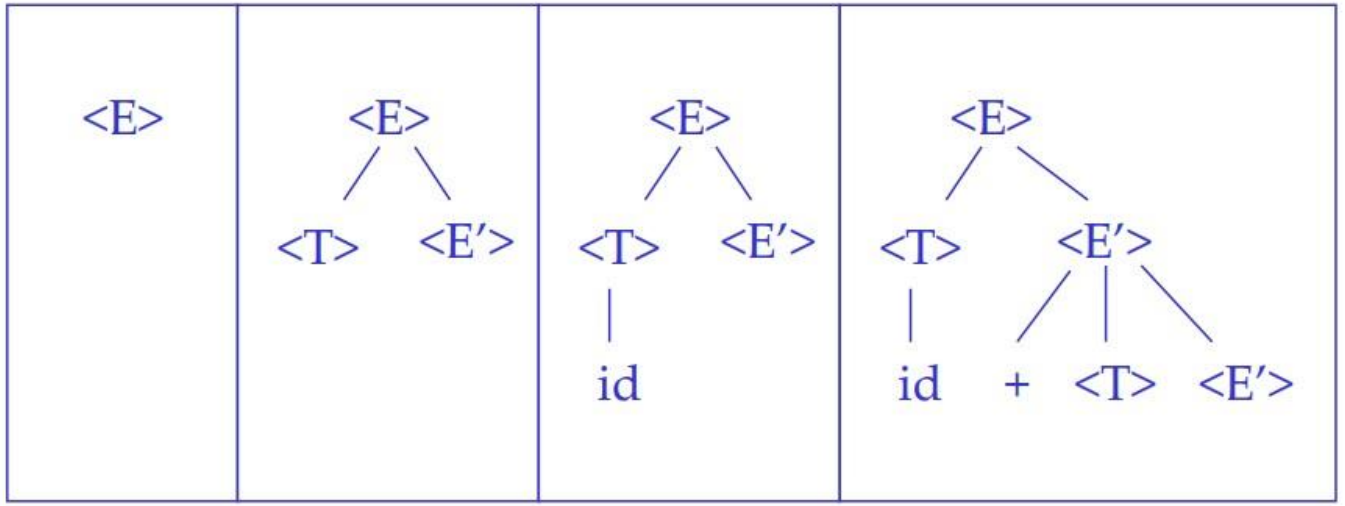
Análisis Sintáctico Descendente - Ejemplo

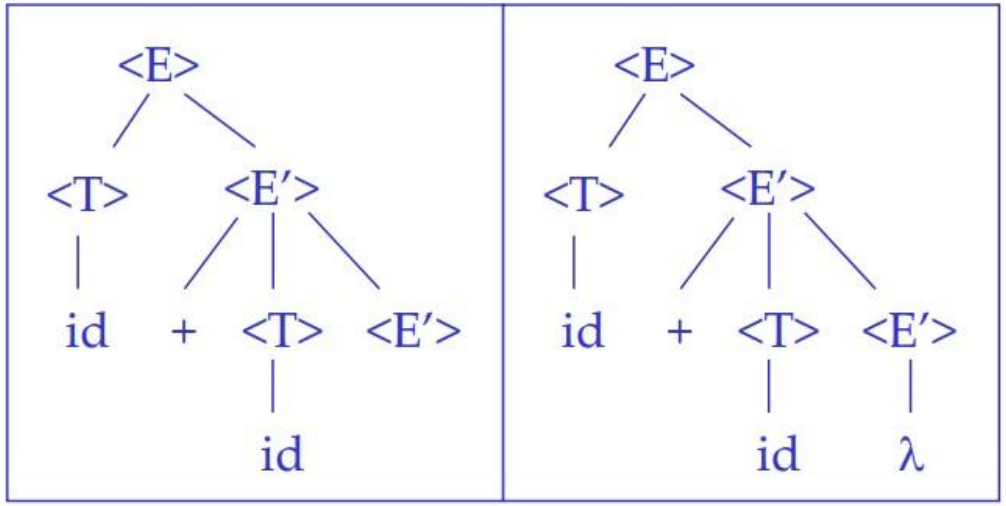
Gramática

- <E> → <T> <E′>
- <E′> → + <T> <E′>
- <E′> → λ
- <T> → id

Sentencia

id + id





Análisis Sintáctico Descendente con retroceso

En análisis sintáctico descendente intenta encontrar entre las producciones de la gramática la derivación por la izquierda del símbolo inicial para una cadena de entrada.

Ejemplo:

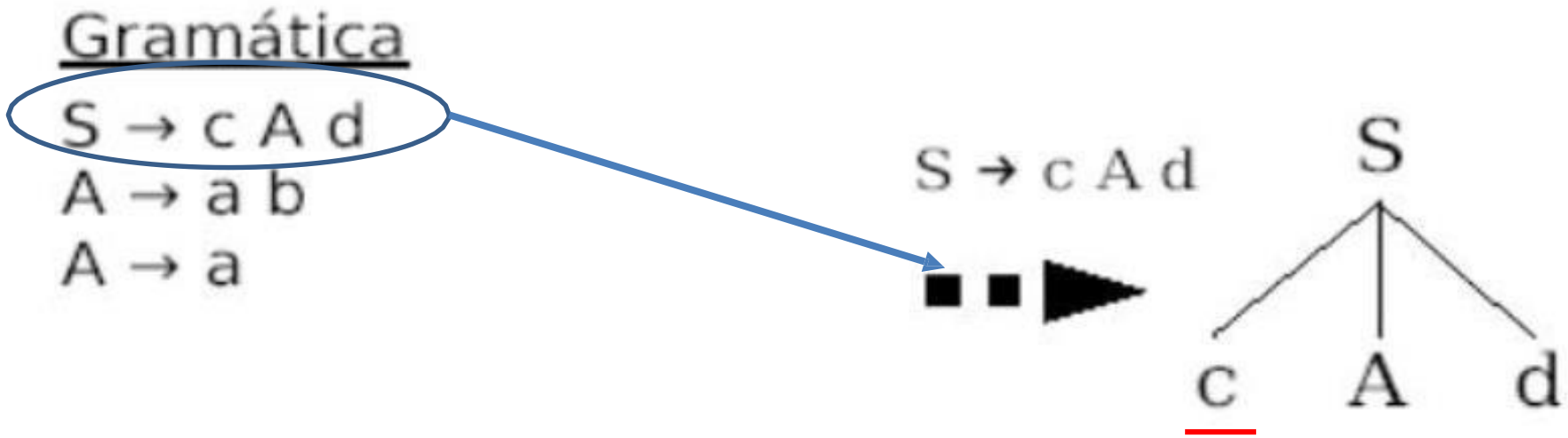
Dada la cadena de entra “cad”, realizar el análisis sintáctico descendente para las siguientes producciones de la gramática:

$$S \rightarrow c\ A\ d$$

$$A \rightarrow a\ b \mid a$$

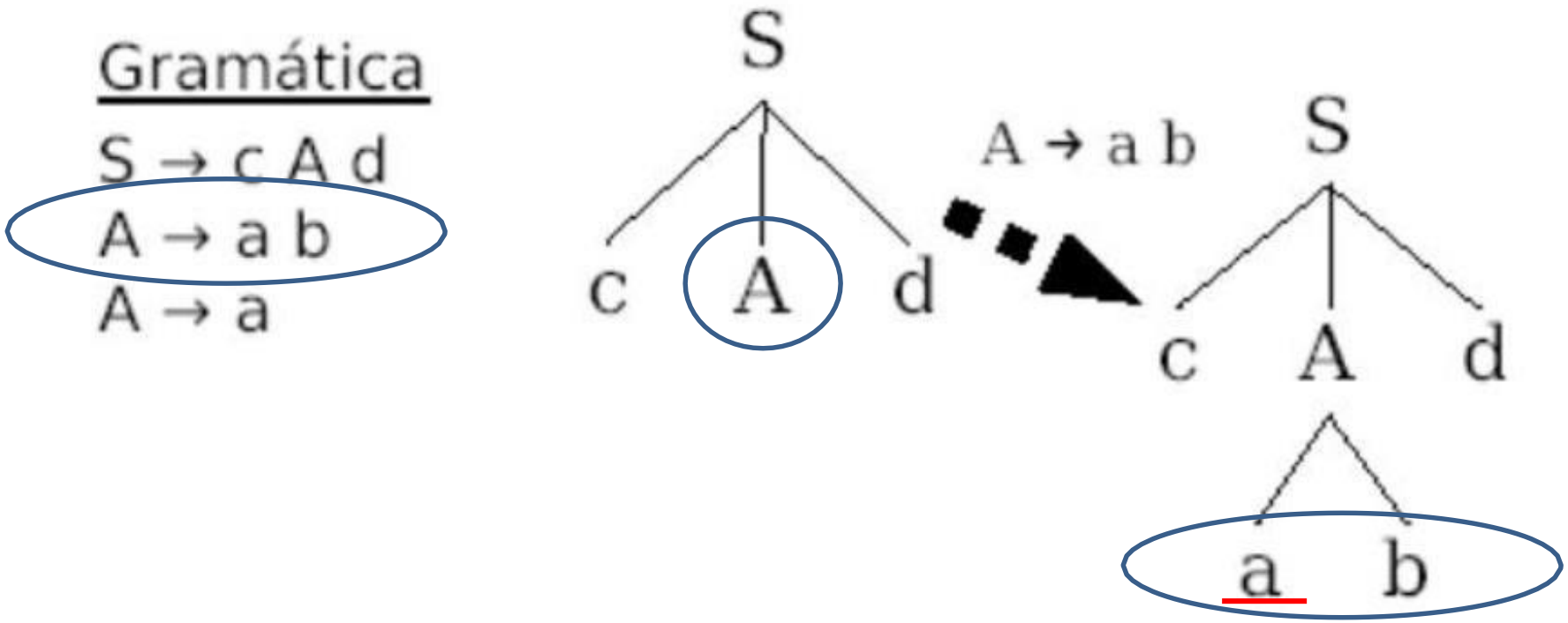
ASD con retroceso: Ejemplo (1)

Paso 1: Se toma la primera producción para **cad**. El primer elemento de la cadena se compara con la primera hoja del árbol.



ASD con retroceso: Ejemplo (2)

a

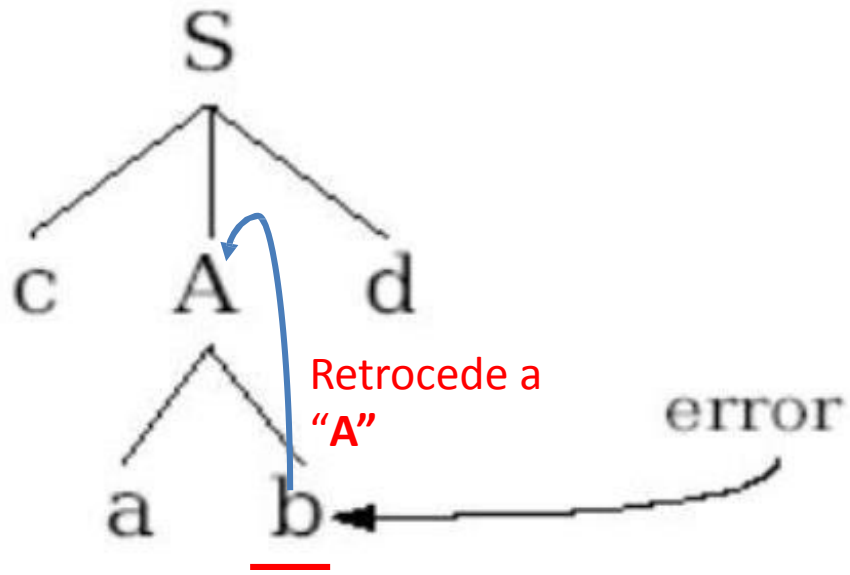


ASD con retroceso: Ejemplo (3)

Paso 3: La cadena de entrada **cab** se compara con la siguiente hoja del árbol etiquetada con el terminal **b**. Como no concuerda, se produce un error y retrocede al terminal **A** para visualizar si existen mas producciones no usadas para reemplazar.

Gramática

S → c A d
A → a b
A → a

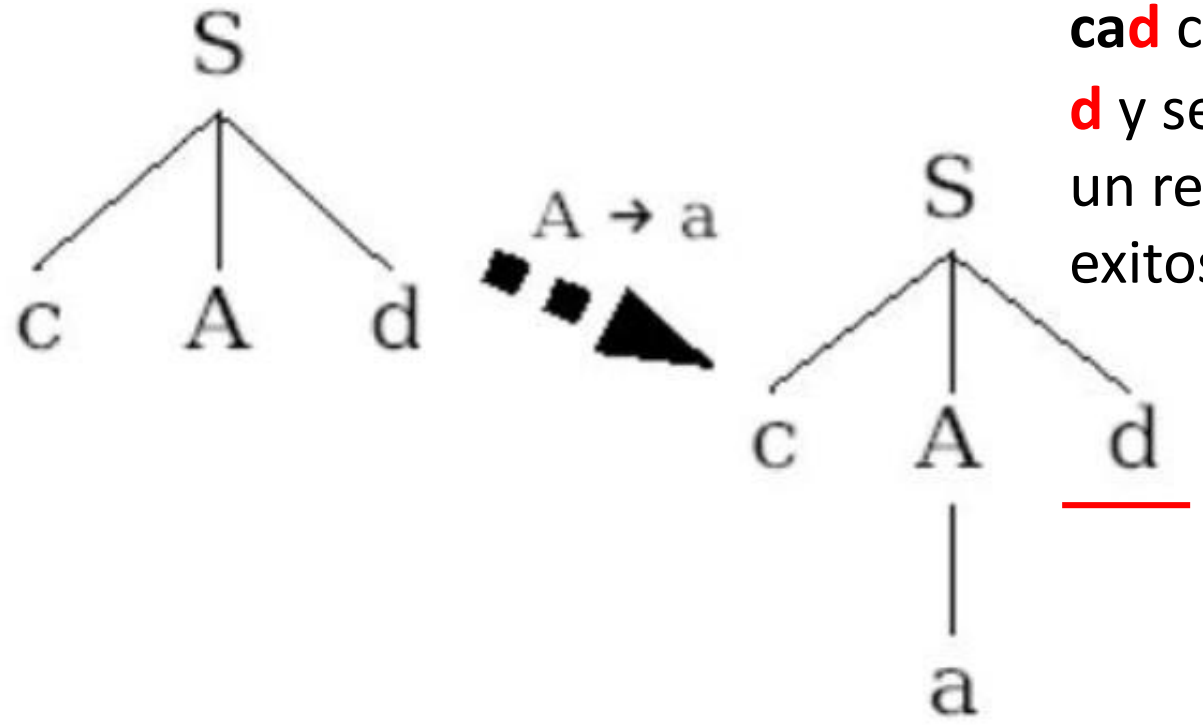


ASD con retroceso: Ejemplo (4)

Paso 4: Se toma la siguiente alternativa de producción para realizar el reemplazo y nuevamente la cadena de entrada **cad** se compara con la siguiente hoja del árbol de derivación.

Gramática

S → c A d
A → a b
A → a



cad coincide con **d** y se produce un resultado exitoso

ASD con retroceso: Ejercicio

Dada la cadena de entra “read”, realizar el análisis sintáctico descendente para las siguientes producciones de la gramática:

```
S → rXd | rZd
X → oa | ea
Z → ai
```

ASD con retroceso: Procedimiento

Consiste en intentar sustituir cada símbolo no terminal por cada una de sus producciones hasta encontrar la producción adecuada. El procedimiento a seguir es el siguiente:

- Se parte del símbolo inicial.
- Se realiza una derivación por la izquierda .
- En cada paso se sustituye un no terminal por una de sus reglas de producción.
- Si se produce un error se elimina la última derivación y se sustituye el no terminal por la siguiente regla de producción.

Para el ejemplo 1

Procedimiento:

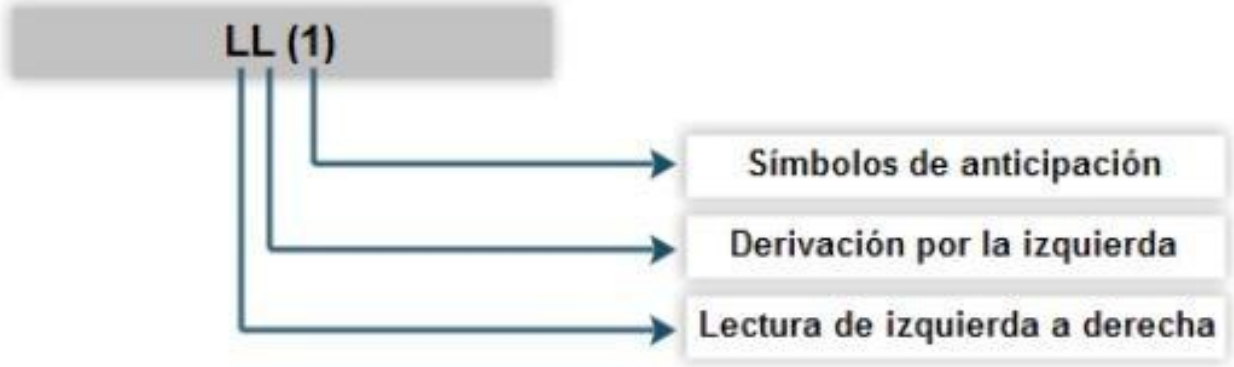
- <S> ⇨ c <A> d [consumir c]
- <S> ⇨ c <A> d ⇨ c a b d [consumir a] [consumir b] ERROR
- <S> ⇨ c <A> d ⇨ c a d [consumir a] [consumir d] CORRECTO

Problemas en el Análisis Sintáctico Descendente

- La recursividad a izquierda da lugar a un bucle infinito de recursión.
- Problemas de indeterminismo cuando varias alternativas en una misma producción comparten el mismo prefijo. No sabríamos cual elegir. Probaríamos una y si se produce un error haríamos backtracking. Si queremos que el método sea determinista hay que evitarlas.

Gramática LL(1)

Las gramáticas LL(1) forman parte de los analizadores sintácticos descendentes predictivos (distintos a los recursivos). Los significados de las siglas son:



Por tanto, este tipo de gramáticas recorren el árbol sintáctico de izquierda a derecha, y en ese recorrido selecciona las derivaciones más a la izquierda y son capaces de reconocer si la sentencia pertenece a la gramática con solo ver un (1) símbolo por anticipado.

Gramática LL(1). Factorización

Cuando dos alternativas para un no terminal comienzan con el mismo símbolo, el analizador no sabe que alternativa usar para poder expandir el lenguaje. Por ejemplo:

$$\begin{aligned} \text{Stmt} \rightarrow\ & \textbf{if}\ \text{E}\ \textbf{then}\ \text{Stmt}\ \textbf{else}\ \text{Stmt} \\ \mid\ & \textbf{if}\ \text{E}\ \textbf{then}\ \text{Stmt} \\ \mid\ & \textbf{otras} \end{aligned}$$

Si en la cadena de entrada tenemos como primer símbolo el token **if**, no se podría determinar cuál de las dos alternativas elegir. La idea es reescribir la producción para retrasar la decisión y elegir la opción correcta.

$$\begin{aligned} \text{Stmt} &\rightarrow \textbf{if}\ \text{E}\ \textbf{then}\ \text{Stmt}\ \text{Stmt'} \mid \textbf{otras} \\ \text{Stmt'} &\rightarrow \textbf{else}\ \text{Stmt} \mid \epsilon \end{aligned}$$

Gramática LL(1). Factorización: Ejercicio

Se tienen las siguiente reglas de producción:

$S \longrightarrow \alpha\beta$

$S \longrightarrow 1$

$\alpha\beta$

¿Cómo quedarían las producciones si estas se reescriben para conseguir LL(1)?

Análisis Sintáctico Descendente con Predictivo

El analizador debe realizar la previsión de la regla a aplicar sólo con ver el primer símbolo que produce para que el algoritmo tenga una complejidad lineal.

Ejemplo:

- **if** Express then Sent
- **while** Express do the Sent
- **begin** Sent end

Existe sólo una posibilidad de derivación, según el primer símbolo que se reciba como entrada, pudiendo ser **if, while o begin**.

Análisis Sintáctico Descendente con Predictivo

Las gramáticas que son susceptibles de ser analizadas sintácticamente de forma descendentemente mediante un análisis predictivo y consultando únicamente un símbolo de entrada pertenecen al grupo LL(1).
A partir de gramáticas LL(1) se pueden construir analizadores sintácticos descendentes predictivos, que son sin retroceso.

Conjuntos de Predicción

Son conjuntos de símbolos terminales que ayudan a predecir que regla se debe aplicar para el no terminal que hay que derivar.
Se construyen a partir de los símbolos de las partesderechas de las producciones de la gramática.
El analizador consulta el siguiente símbolo en la entrada, si pertenece al conjunto de predicción de una regla, aplica esa regla, sino da error.

Ejemplo:

Supongamos que tenemos la entrada "**babxcc**", en donde ya se han leído los tres primeros símbolos **bab**, y se tiene la siguiente gramática:

$$A \rightarrow a\ B\ c \mid x\ C \mid B$$
$$B \rightarrow bA$$
$$C \rightarrow c$$

Conjuntos de Predicción: Ejemplo (1)

$A \rightarrow a\ B\ c \mid x\ C \mid B$

$B \rightarrow bA$

$C \rightarrow c$

babxcc

La cadena de derivación ha sido la siguiente:

$A \rightarrow B \rightarrow bA \rightarrow baBc \rightarrow babAc$

Ahora hay que seguir desarrollando la variable A utilizando los conjuntos de predicción.

Como la siguiente letra es un “x” se elige la producción $A \rightarrow xC$

Conjuntos de Predicción: Ejemplo (2)

La gramática:

$$A \rightarrow aBc \mid aC \mid B$$

No cumple con los requisitos para LL(1) porque si aparece una “a” en la entrada hay dos posibles opciones.

Por lo tanto:

- El análisis no puede ser predictivo
- La gramática no es LL(1)

Conjuntos de Predicción: Ejercicio

Para la siguiente entrada **aabb**, cuya gramática propone las siguientes reglas de deriva

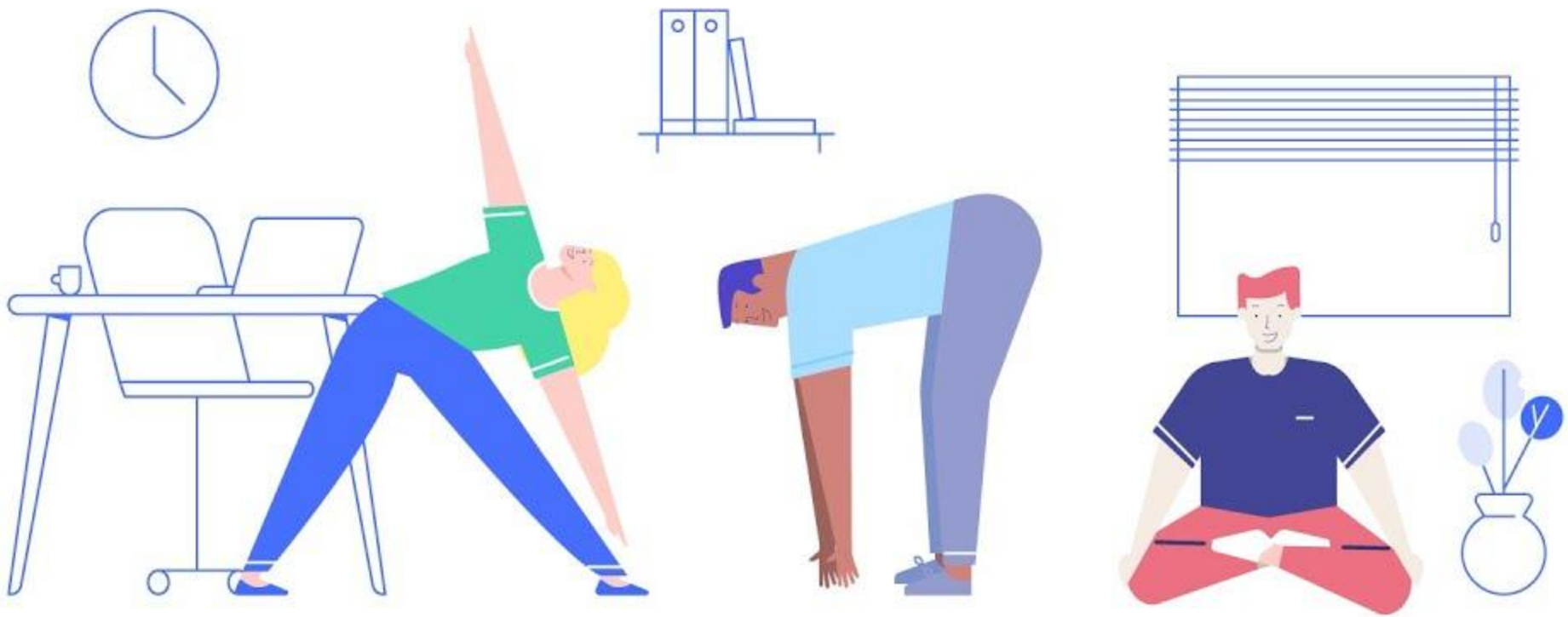
$$S \rightarrow a\ B$$

$$B \rightarrow b \mid a\ B\ b$$

Construir el árbol de derivación correspondiente haciendo uso de el ASDP.

PAUSA ACTIVA

Deportes y Tiempo Libre



Pausa activa de 5 minutos

Y Cuál Es El Sentido De La Vida ? Hermoso Cortometraje (Weeds)

https://www.youtube.com/watch?v=8ntV-PwCKdY

Cronología de la Clase
Secuencia y explicación

Inicio

Cierre

Utilidad

Práctica

Transformación

HORA PRÁCTICA Actividad

Elabora una tabla con 3 características de Análisis Sintáctico Descendente:

	de izquierda a derecha,	De derecha a izquierda	Con retroceso	Determinista
Característica 1				
Característica 2				
Característica 3				

Tarea

Cada Integrante del grupo elige un tema y realiza su investigación en forma INDIVIDUAL

Concepto+su respectivo ejemplo

Grupo 1 = Tema1:Análisis Sintáctico

- Descendente de izquierda a derecha.
- Descendente de derecha a izquierda.
- Descendente con retroceso.

Grupo 2 = Tema2: Factorización

- Análisis Sintáctico Descendente determinista.
- Factorización izquierda
- Eliminación de recursividad izquierda

Grupo 3 = Tema3: Gramáticas

- Gramáticas LL (1) Simples sin reglas vacías.
- Cadena de entrada sin reconocer
- . Pila
- . Producciones usadas.
- Tabla de la Función Acción.

Un grupo en forma aleatoria, presentará su avance

Aleatoriamente 1 grupo presentará su avance

Retroalimentación

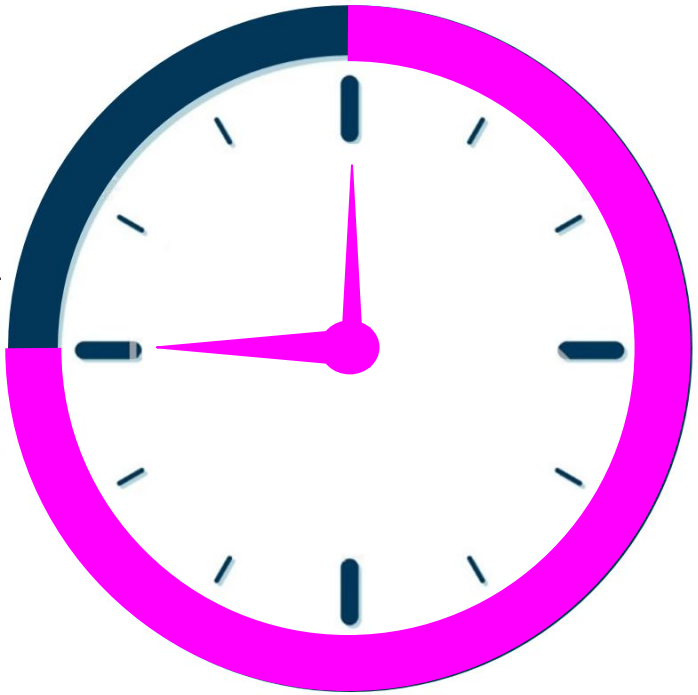
https://es.piliapp.com/random/wheel/

Cronología de la Clase
Secuencia y explicación

Inicio

Cierre

Verificar el logro de la sesión
A través de un caso; verificar si entendieron el tema.
Reforzar los puntos débiles.



Utilidad

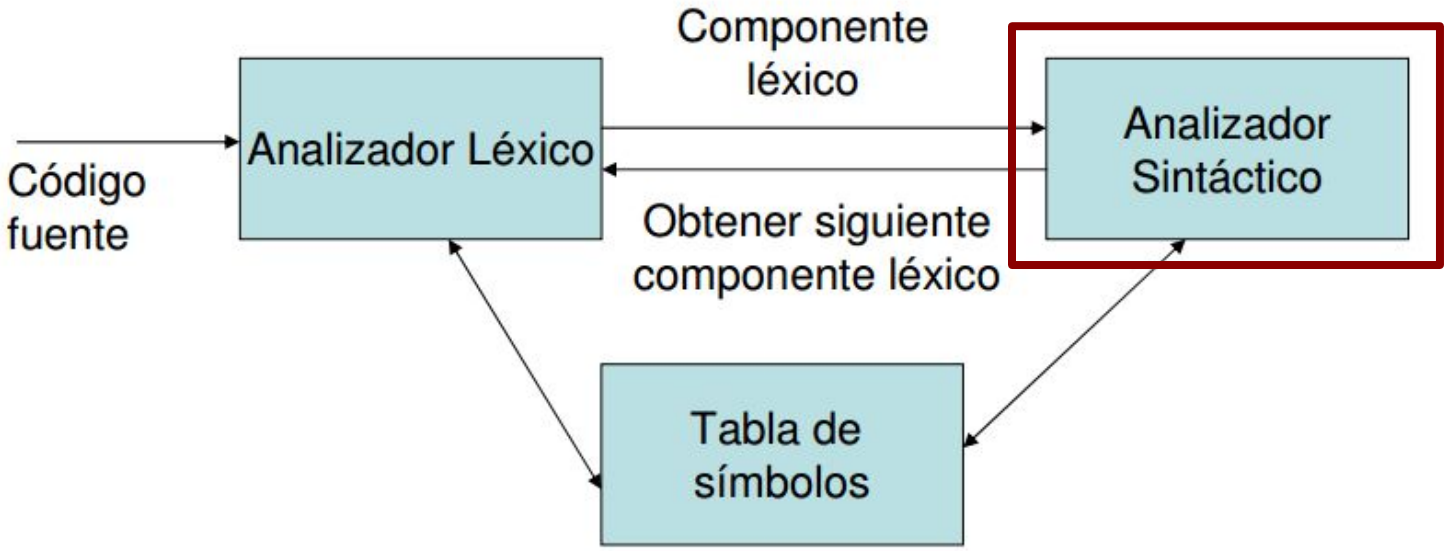
Práctica

Transformación

4.- Conclusiones

Análisis Sintáctico

Actividad en la que se examina una secuencia de entrada de cadenas para determinar su estructura gramatical



Verificación del logro

Responde verdad (V) o falsedad (F)

Análisis Sintáctico

¿Actividad en la que se examina una secuencia de entrada de cadenas para determinar su estructura gramatical?

V

F

Responda en el chat

Exclusión mutua
excluyen mutuamente

Metacognición

¿De qué forma aprendí el tema tratado en clase?
¿Qué aprendí el día de hoy?

Espacio para generar sus preguntas

Respuesta a intervenciones

Gracias

Universidad Tecnológica del Perú

Facultad de Ingeniería
Departamento Académico de Sistemas y Electrónica