Clase 4
Modelado Formal con Gramáticas

- Reglas de producción y sistemas dirigidos por eventos.
- Gramáticas libres de contexto (GLC): G=(V,Σ,P,S)G=(V,Σ,P,S).
- Ejemplos: Simulación de crecimiento (L-systems).
- Modelado de protocolos o tareas con Gramáticas Libres de Contexto.
- Aplicación: cómo definir el comportamiento de un modelo discreto con reglas

BIBLIOGRAFÍA GENERAL

- Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2ª ed.). Addison-Wesley.
- Chomsky, N. (1959). On Certain Formal Properties of Grammars. Information and Control.
- Prusinkiewicz, P., & Lindenmayer, A. (1990). The Algorithmic Beauty of Plants. Springer.

Modelado Formal con Gramáticas

- En la modelización de sistemas complejos, como el lenguaje natural (NLP), los parsers formales son fundamentales.
- El proceso se inspira en los compiladores:
  - Análisis Léxico (Lexical Analysis)
    - Transformar la secuencia de caracteres en tokens.
  - Análisis Sintáctico (Parsing)
    - Verificar si la secuencia de tokens pertenece a un lenguaje formal definido por una Gramática Libre de Contexto (CFG).
  - Análisis Semántico
    - Asociar significados, aplicar restricciones contextuales y construir representaciones intermedias (árboles de sintaxis abstracta, lógicas de predicados, etc.).
  - Traducción Dirigida por la Sintaxis (SDT)
    - Enriquecer la CFG con acciones semánticas (traducción a estructuras intermedias).

Modelado Formal con Gramáticas

- Aplicaciones en Modelos y Simulación
  - Chatbots y asistentes virtuales: parsing formal para controlar el flujo de diálogo.
  - Simulación basada en reglas: CFG + SDT para modelar interacciones de agentes en lenguaje natural.
  - Procesamiento previo a ML/NLP: limpieza y estructuración de frases antes de embeddings/transformers.

Modelado Formal con Gramáticas

- Embeddings
  - Representaciones vectoriales de palabras, frases o documentos en un espacio de dimensión finita (R^d).
  - Intuición: convierten texto en números que capturan similitud semántica.
    - Ejemplo: “gato” → [0.2,−0.5,0.8,0.1]
    - “perro” → cercano a “gato” en el espacio.
  - Uso: entradas para modelos de NLP, búsqueda semántica, clasificación.

Modelado Formal con Gramáticas

- Embeddings
  - Con varias dimensiones (ejemplo 4D)
  - Al usar un vector en R^d cada coordenada puede capturar un aspecto diferente (aunque no interpretable directamente):
    - "gato" → [0.2,−0.5,0.8,0.1]
    - "perro" → [0.1,−0.6,0.7,0.0]
    - "avión" → [−0.9,0.2,−0.1,0.5]
  - “gato” y “perro” quedan cerca en el espacio (vectores similares), pero “avión” queda lejos.
  - No es que la primera coordenada sea “animal” y la segunda “tamaño”.
  - Más bien, el conjunto completo del vector es lo que captura el significado.

Modelado Formal con Gramáticas

- Embeddings
  - Dimensiones típicas
    - Word2Vec → 50 a 300 dimensiones.
    - GloVe → 100 a 300.
    - BERT / GPT → 768, 1024 o más.
  - Cuantas más dimensiones → más capacidad para representar relaciones semánticas finas, aunque también más costo de cómputo.
  - Imaginar que cada palabra es un punto en un espacio de alta dimensión.
    - Distancia pequeña → significados cercanos ("gato", "perro").
    - Distancia grande → significados lejanos ("gato", "avión").
    - Direcciones → relaciones semánticas ("rey - hombre + mujer ≈ reina").

Modelado Formal con Gramáticas

- Embeddings
  - Una palabra no se puede describir con un solo número porque el lenguaje tiene muchos matices.
  - Un embedding de varias dimensiones permite capturar múltiples aspectos de significado.
  - No miramos cada número por separado → lo importante es la posición relativa de los vectores en el espacio.

Modelado Formal con Gramáticas

- Embeddings
  - ¿Cada elemento del vector representa una caracteristica (cercana o lejana)?
  - Cada elemento del vector es una coordenada.
  - En un embedding, la palabra se representa como un punto en un espacio de muchas dimensiones.
  - Cada número es una coordenada en ese espacio (igual que un punto en 2D tiene (x,y) pero aquí hay cientos de coordenadas).
  - Ejemplo en 2D (fácil de visualizar): (en 4D o más ya es imposible visualizarlos)
    - "gato" → (2, 1)
    - "perro" → (2.2, 1.1)
    - "avión" → (-3, 5)
    - “Gato” y “perro” quedan cerca en el plano, “avión” queda lejos.

Modelado Formal con Gramáticas

- Embeddings
  - ¿Cada coordenada = representa una característica?
  - No necesariamente.
    - No es que el primer número sea “animal”, el segundo “pelaje” o el tercero “doméstico”.
  - Los modelos no aprenden significados humanos directos, sino patrones estadísticos del lenguaje.
  - Lo que buscamos es la combinación de todas las coordenadas que ubica cada palabra en un lugar donde la distancia y la dirección reflejan similitud semántica.
  - Si solo tenemos 1 dimensión (una línea), solo podemos decir “más a la izquierda” o “más a la derecha”.
  - Con 2 dimensiones (un plano), ya podemos distinguir norte-sur y este-oeste.
  - Con 300 dimensiones, cada objeto (palabra) se ubica de forma que refleja muchos matices de similitud al mismo tiempo.

Modelado Formal con Gramáticas

- Transformers
  - Arquitectura de red neuronal (2017, “Attention is All You Need”) que revolucionó NLP.
  - Usa mecanismo de atención para procesar texto en paralelo y capturar relaciones de largo alcance.
  - Ejemplo: en “El gato duerme en la cama”, el modelo entiende que “cama” se relaciona más con “duerme” que con “El”.

Modelado Formal con Gramáticas

```
%{
#include "y.tab.h"
%}

digit   [0-9]
id      [a-zA-Z_][a-zA-Z0-9_]*

%%
{digit}+        { yylval = atoi(yytext); return NUMBER; }
{id}            { return ID; }
"+"             { return PLUS; }
"*"             { return TIMES; }
"("             { return LPAREN; }
")"             { return RPAREN; }
[ \t\n]         ;
.               { return yytext[0]; }
%%
```

Modelado Formal con Gramáticas

```
%{
#include <stdio.h>
#include <stdlib.h>
%}

%token NUMBER ID
%token PLUS TIMES LPAREN RPAREN

%%
expr    : expr PLUS term   { printf("Suma\n"); }
        | term
        ;
term    : term TIMES factor { printf("Producto\n"); }
        | factor
        ;
factor  : LPAREN expr RPAREN
        | NUMBER
        | ID
        ;
%%
int main() { return yyparse(); }
int yyerror(char *s) { printf("Error: %s\n", s); return 0; }
```

Modelado Formal con Gramáticas

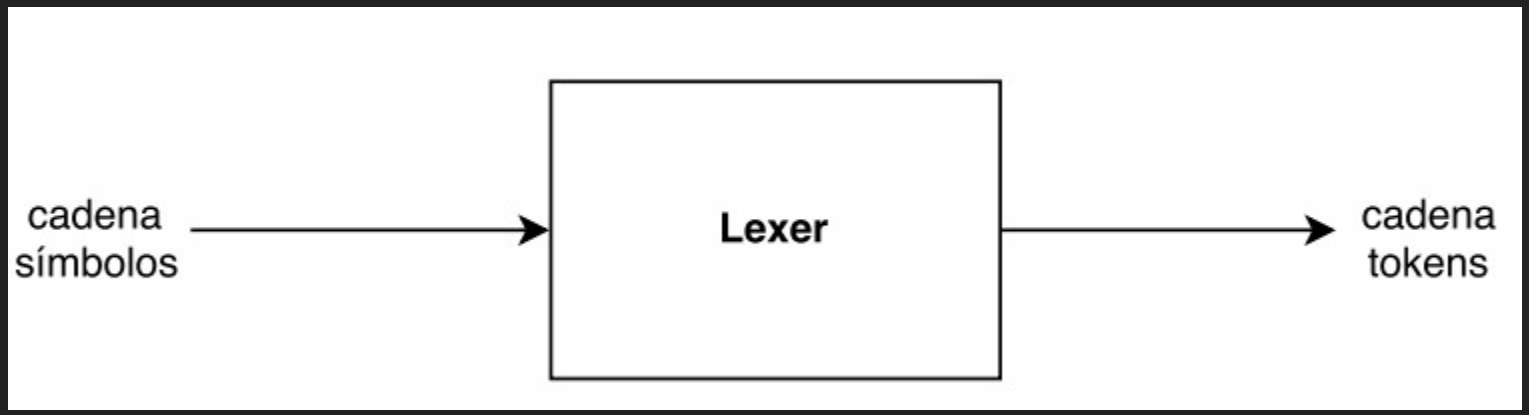
- Análisis Léxico: Se encarga de procesar las cadenas de caracteres y generar los lexemas correspondientes.
  - Por cada lexema se generará las lista de tokens (par token-valor).
- Análisis Sintáctico: A partir de la salida del analizador léxico se genera el árbol sintáctico correspondiente.
- Análisis Semántico: Se genera información para las etapas subsiguientes, y en particular se encargará del chequeo de tipos.

Modelado Formal con Gramáticas

```
expr : expr PLUS term   { $$ = $1 + $3; }
     | term             { $$ = $1; }
     ;
term : term TIMES factor { $$ = $1 * $3; }
     | factor            { $$ = $1; }
     ;
factor : NUMBER          { $$ = $1; }
       | LPAREN expr RPAREN { $$ = $2; }
       ;
```

Modelado Formal con Gramáticas

- Analizador Léxico o Lexer:
  - Usar Expresiones Regulares.
  - Agrupar símbolos de la entrada en tokens.
  - Cada token ⇒ {<tipo>,<valor>}



Modelado Formal con Gramáticas

- Analizador Léxico o Lexer:

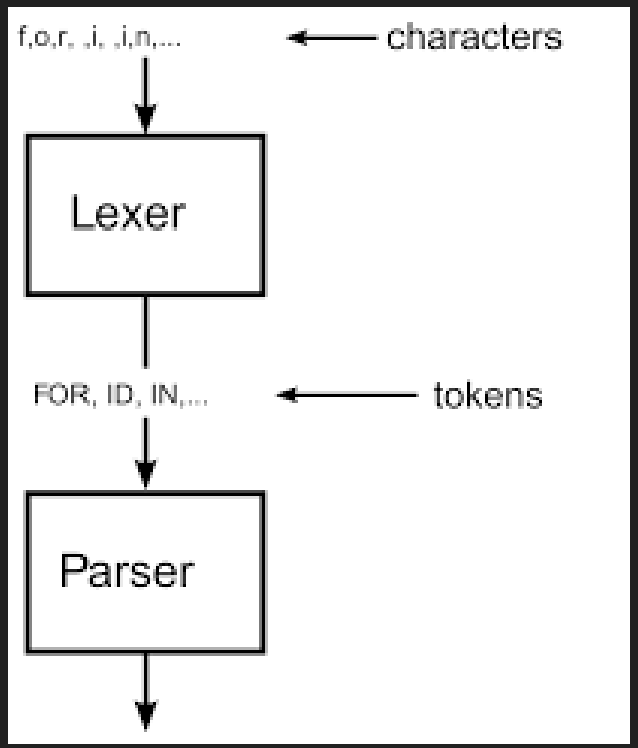
if	sueldo	==	1000	sueldo	*	0.25	;

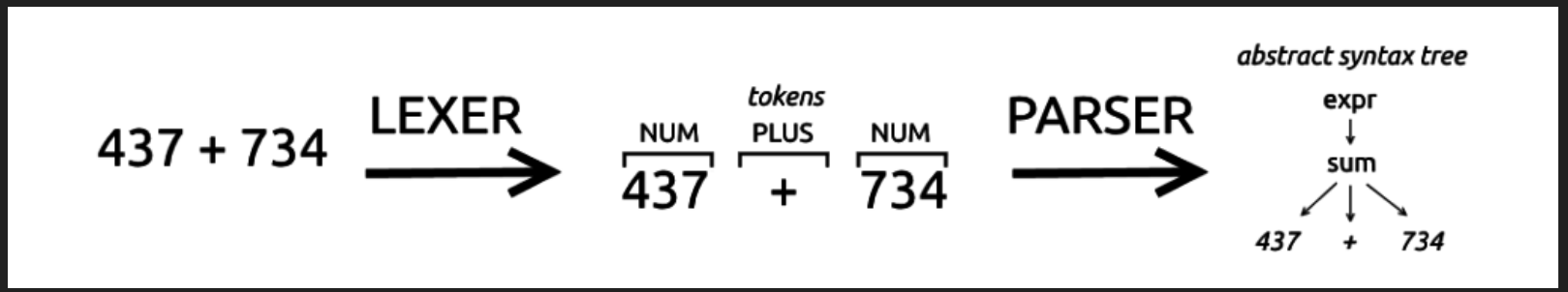
y les asigna su atributo cuyo significado se ha definido previamente:

Token	Atributo	Observaciones
If	20	Palabra reservada
Sueldo	1	Identificador
==	15	Operador de comparación
1000	8	Valor numérico o constante
*	12	Operador aritmético
0.25	8	Valor numérico o constante
;	27	Separador de sentencias

Modelado Formal con Gramáticas

- Lexer y Parser:





Modelado Formal con Gramáticas

- Tareas necesarias para modelar un lenguaje:
  - Aprender o Definir la sintaxis y la semántica de un lenguaje.
  - Traducir la sintaxis y semántica del lenguaje a otra forma contextual.
  - Sintaxis: es cómo se ve el lenguaje escrito
  - CGF ó BNF (Backus-Naur Form)
- Semántica: qué significan los elementos de un lenguaje.
  - Satisfacen ciertos aspectos y restricciones que no pueden ser explícitamente especificados por la sintaxis.
  - Traducción: cómo realizar la traducción a algo interpretable por una computadora o IA.
  - Traducción dirigida por la sintaxis: la sintaxis especificada por CGF/NBF puede ser usada para guiar el proceso de traducción

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- CFG: Context Free Grammar
  - Se usa para la especificación de la sintaxis de un lenguaje
  - Describe la estructura jerárquica con reglas de producción
  - Ejemplo: if (expresión) sentencia else sentencia
    - sentencia=stmt (statement)
    - CFG: stmt $\rightarrow$ if (expr) stmt else stmt
    - Tokens: elementos léxicos en una regla de producción
    - Palabras clave (“if”, “else”), paréntesis (“(“,”)”)
    - no-terminales (non-terminals): variables como expr y stmt

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- CFG: especificación formal del árbol de análisis
  - G=(Σ, N, P, S)
  - Σ= símbolos terminales
  - N= no terminales
  - P= reglas de producción
  - S= símbolo de arranque (start symbol)
- Lenguaje G definido por la gramática:
  - G =< {E, T, F},{+,∗,num, (,)}, P, E >

$$
\begin{aligned}
E &\rightarrow E + T \\
E &\rightarrow T \\
T &\rightarrow T * F \\
T &\rightarrow F \\
F &\rightarrow num \\
F &\rightarrow (E)
\end{aligned}
$$

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Una gramática (G) deriva cadenas de palabras.
- Empieza con el símbolo inicial.
- Reemplaza repetidamente un no-terminal con la RHS de la producción de ese no-terminal.
- Finaliza con una cadena de terminales.
  - Derivar == generar == reescribir
  - Derivación == generación == reescritura

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- LENGUAJE : son las cadenas de palabras o tokens que pueden ser derivados del símbolo inicial.

- L(G)={cadenas de palabras derivadas del símbolo inicial} = {s | $S =^{*} => s$}

- Cadena Vacía: es la cadena que tiene cero palabras, se define como β o también como ε ó Φ.

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- 4 componentes de un CFG
  - Conjunto de tokens: símbolos terminales
  - Conjunto de no terminales
  - Conjunto de producciones
  - LHS (Left-hand side): no terminal
  - RHS (Right-hand side): una secuencia de terminales o no terminales
  - Símbolo de arranque (start symbol) (de no-terminales) que especifica la construcción primaria (o la más alta en el árbol) del programa

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Σ= símbolos terminales
  - Símbolos de entrada del lenguaje
  - Lenguajes naturales o de programación: tokens (palabras reservadas, variables, operadores,...)
  - Lenguajes naturales: palabras o partes de un idioma
  - Pre-terminal: partes de un idioma cuando las palabras son vistas como terminales
- N= símbolos no terminales
  - Grupos de terminales y/o otros no-terminales

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- S: start symbol: el mayor constituyente de un árbol de análisis
- P: reglas de producción (volver a escribir)
  - forma: $\alpha \rightarrow \beta$ (α: no-terminal, β: cadena de terminales y no-terminales)
  - significa: α vuelve a escribir (“consiste de”, “derivada en”)β, o β reducida a α
- Comienza con “producciones S “S-productions”($S \rightarrow \beta$)

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Dada la cadena:
  - $\alpha : (15 + 6) * 3$
  - $¿\alpha \in G?$
  - Cómo sabe la computadora qué es un num
  - Reemplazamos al terminal num por los terminales 15, 6 y 3 ?
  - Reemplazamos la producción $F \rightarrow$ num por las producciones $F \rightarrow 15$, $F \rightarrow 6$ y $F \rightarrow 3$?
  - Se puede hacer esto para todos los números?, pues son infinitos.

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Gramática para lista de dígitos separados por signos positivos y negativos
  - lista→lista + dígito
  - lista→lista - dígito
  - lista→dígito
  - digíto →0 | 1 | 2 | ...| 9
- En forma resumida
  - lista→lista + dígito | lista – dígito | dígito
  - digíto →0 | 1 | 2 | ...| 9
- Tokens terminales: ‘+’. ‘-’, ‘0’, ‘1’, ..., ‘9’
- no-terminales: { lista, dígito }
- Símbolo de arranque: lista
- Lenguaje: L(G-dígito)= { 9-5+2, 2+3+4-5 , ...}

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- “ Producción de un no-terminal NT”
- no-terminal LHS de una producción
  - $\alpha \rightarrow \beta$:”producción para β”
- Producción-N
  - Conjunto de reglas de producción para el símbolo NT N
  - Ejemplo: Producción-S para el símbolo de arranque o de inicio
- “ cadena de tokens o palabras“
  - Es una secuencia de cero o más palabras
- Cadena vacía (ε) es la cadena sin palabras (cero)

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Especificación de las Estructuras y sus miembros (circunscripciones).
- Arbol de Análisis o Parse Tree es la representación gráfica de la estructura.
  - Nodo Raíz (S): es la estructura a nivel de sentencia
  - Nodos internos: constituyentes de la sentencia
  - Arcos: relaciones entre los nodos padre y sus hijos (constituyentes)
  - Nodos terminales: símbolos de entrada (ej. palabras)
  - Representación alternativa con corchetes:
    - [Yo ví [al [perro [en [el parque]]]]]

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Gramática con Producción-ε:
  - " lista de parámetros opcionales, separados con comas "
    - call $\rightarrow$ id (paramop)
    - paramop $\rightarrow$ params | ε
    - param $\rightarrow$ params, param | param
    - Ej: max(x,y)

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Gramática con Producción-ε:
  - “ lista de sentencias (stmt) separadas con punto y coma“
  - block → begin opt_stmts end
    - opt_stmts → stmt_list | ε
    - stmt_list → stmt_list ; stmt | stmt
    - stmt → if-stmt | assgn-stmt |…
    - no-terminales: {block, opt_stmts, stmt_list, stmt}
    - terminales: {“begin”, “end”, other variables … }
- Símbolo de inicio: block
- Comparar con “list”(G-digit)
- Una lista vacía puede aparecer entre “begin”-”end”
- ‘;’↔ ‘+’, ‘-’
- stmt ↔ ‘0’, ‘1’, …, ‘9’

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Reglas de la Gramática
  - S $\rightarrow$ NF VF
  - NF $\rightarrow$ Pron | Nom-Propio | Art Sust
  - Sust $\rightarrow$ Nom Sust | Nom
  - VF $\rightarrow$ Verbo | Verbo NF | Verbo NF FP | Verbo FP
  - FP $\rightarrow$ Prep NF
  - S = Sentencia
  - NF = Nombre en Frase
  - Pron = Pronombre,
  - Art = Artículo, Prep = Preposición, Sust = Sustantivo
  - FP = Frase Preposicional

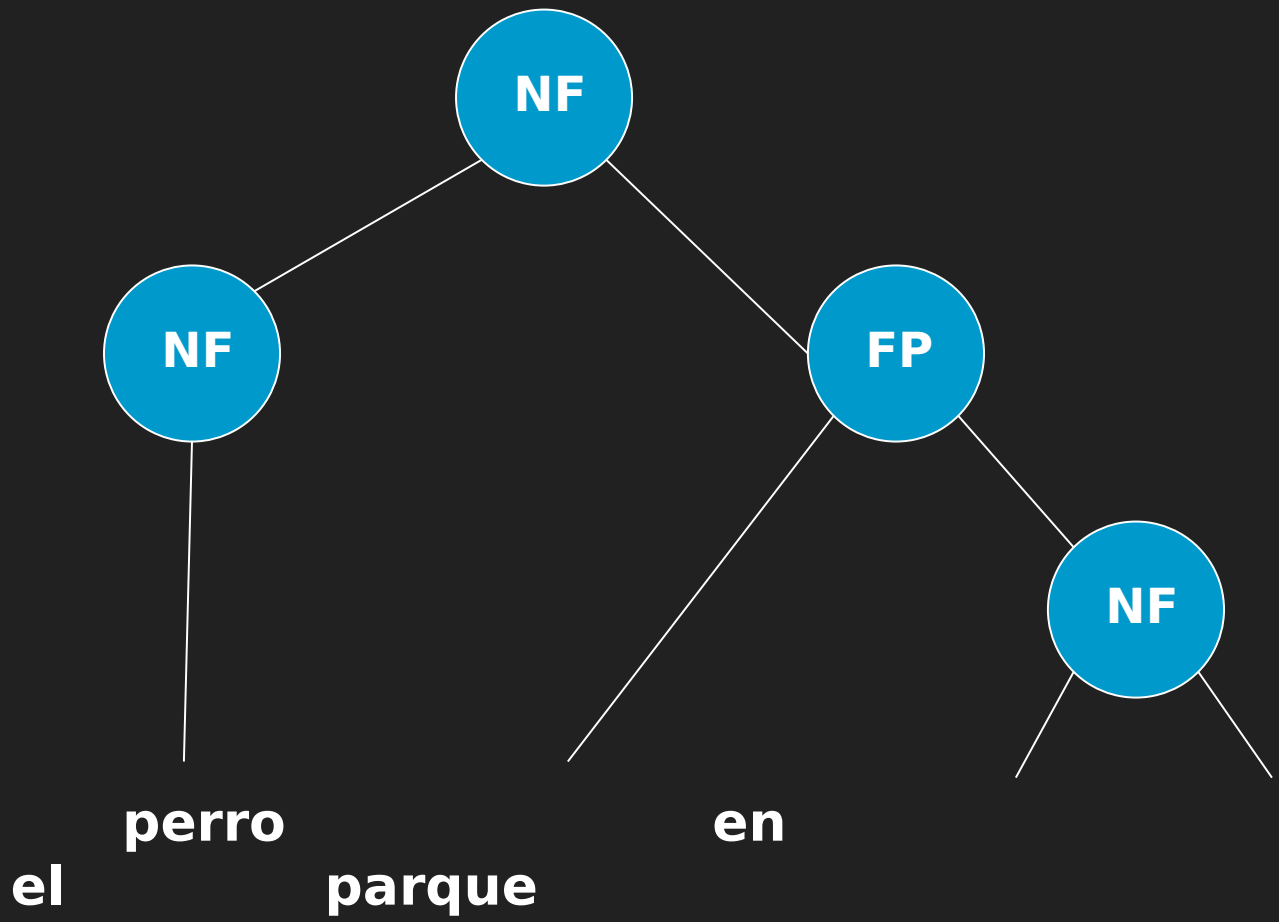
Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Léxico (en forma CFG)
  - Nom Sust → perro | parque | escritorio
  - Verbo → quiero | es | ví | caminé
  - Prep → por | en | con | para
  - Art → el | este | ese
  - Pron → Yo | tú | ella | él
  - Nom Propio → Buenos Aires | Aconcagua

Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)

- Léxico (en forma CFG)
  - Nom Sust → perro | parque | escritorio
  - Verbo → quiero | es | ví | caminé
  - Prep → por | en | con | para
  - Art → el | este | ese
  - Pron → Yo | tú | ella | él
  - Nom Propio → Buenos Aires | Aconcagua

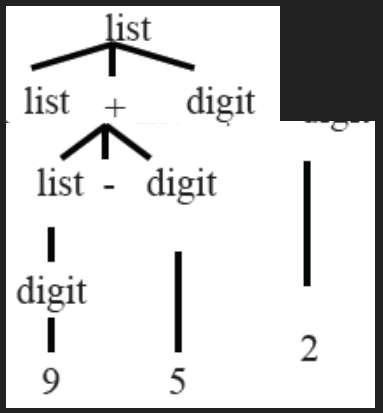
Modelado Formal con Gramáticas
Gramáticas libres de contexto (GLC-GFC)



Modelado Formal con Gramáticas

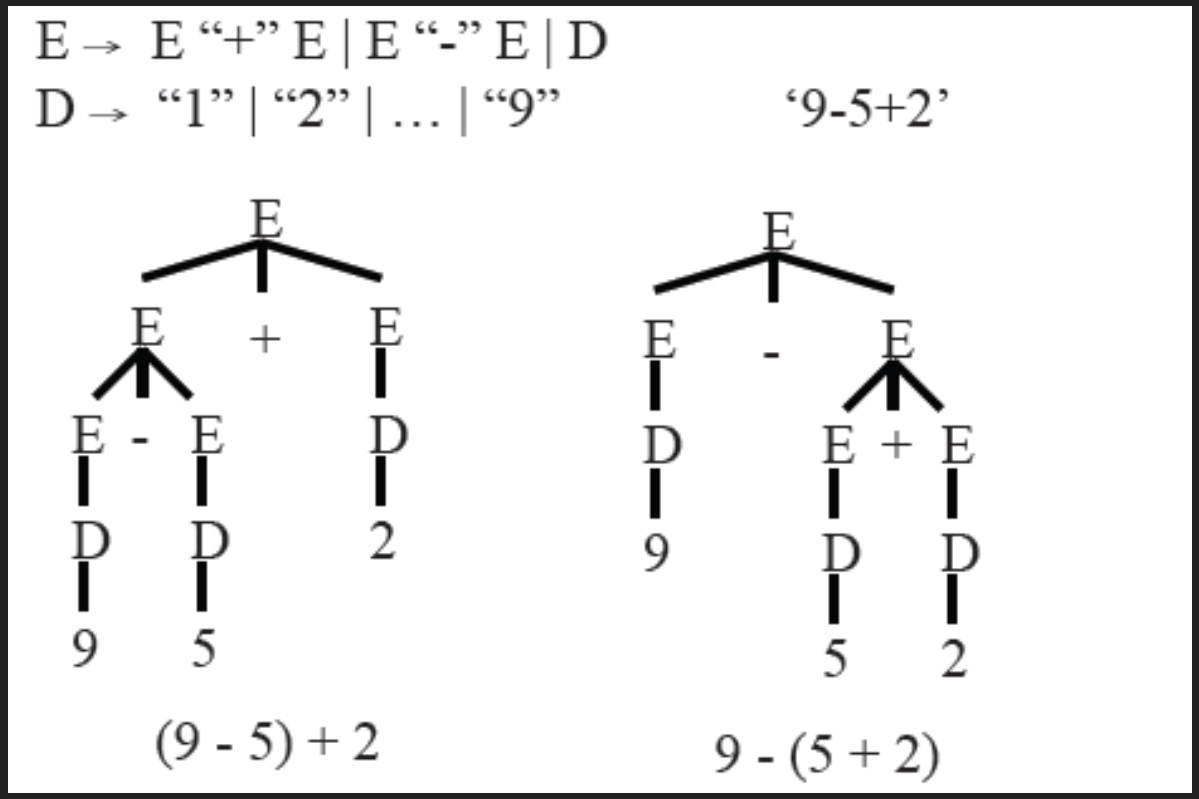
Verificación de la Sintaxis: Análisis Sintáctico (Parsing)

- Parsing: puede ser visto como el proceso inverso de la derivación
- Obtener una secuencia de derivación que produzca una cadena de palabras o tokens de entrada
- Encontrar un árbol de análisis (parse tree) para una cadena de palabras o tokens
  - 9-5+2 = ? => list
- Parse tree (árbol concreto de la sintaxis)
  - raíz (root): símbolo de arranque o de inicio
  - hoja (leaf) : terminal
  - nodos internos : no-terminales
- Producción
  - los nodos se leen de izquierda a derecha (yield leave)



Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Una gramática puede tener más de un árbol de análisis capaz de generar una misma cadena de palabras.
- Es posible tener una gramática propia para un lenguaje.
- Una gramática ambigua vs. un lenguaje ambiguo.



Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Asociatividad de Operadores
  - Asociación por izquierda (LA): 9-5+2 $\leftrightarrow$(9-5)+2
    - LA: cout << “Hello”<< “World”<< endl ;
    - $\leftrightarrow$ cout << “Hello”; cout << “World”; cout << endl;
    - LA: cin >> year >> month >> day ;
    - $\leftrightarrow$ cin >> year; cin >> month; cin >> day;
  - Asociación por derecha (RA): a=b=c $\leftrightarrow$ a=(b=c)
    - derecha $\rightarrow$ letra = derecha | letra
    - letter $\rightarrow$ a | b | …| z
    - RA: a += b += c $\leftrightarrow$ a += (b += c)
  - Precedencia de operadores
    - 9+5*2 $\leftrightarrow$ 9+(5*2)
    - NOT: (9+5) * 2

Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Resolución de Ambiguedades
- Creando un lenguaje no ambiguo o escribiendo una gramática no ambigua:
  - Usar palabras claves para identificar bloques de estructura (“begin-end”).
  - Usar paréntesis (“(“,”)”) para delimitar bloques de sentencias.
  - Hacer cumplir la sinaxis de los lenguajes.
  - Lenguaje artificial, NO lenguaje natural.
  - Escribir una gramática no ambigua que refleje la asociación y la precedencia.
  - Cambio de gramática sin cambio del lenguaje..

Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Resolución de Ambiguedades
- Resolver la asociatividad de operadores
  - LA: Producciones con ramas hacia la izquierda
  - RA: Producciones con ramas hacia la derecha
    - Ejemplo: (RA)
      - $R \rightarrow L = R \mid L$
      - $L \rightarrow a \mid b \mid \ldots \mid z$
    - Ejemplo: (LA)
      - $L \rightarrow L + D \mid L - D \mid D$
      - $D \rightarrow 1 \mid 2 \mid \ldots \mid 9$
- Resolver la precedencia de operadores
  - Definir primero las expresiones con alta precedencia (incluyendo unidades atómicas)
  - Los operadores de baja precedencia operan en expresiones de alta precedencia

Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Resolución de Ambiguedades
- Operadores con Alta Precedencia vs. Operadores con baja precedencia.
- Ejemplo: Expresiones matemáticas
  - factor: unidades básicas
    - factor $\rightarrow$ digito | (expr)
  - term: término, unidades operadas con operadores de alta precedencia (forma LA)
    - term $\rightarrow$ term * factor | term / factor | factor
  - expr: expresión, unidades operadas con operadores de baja precedencia
    - expr $\rightarrow$ expr + term | expr - term | term

Modelado Formal con Gramáticas
Definición de la Sintaxis: Ambiguedad

- Ejemplo: Expresiones matemáticas
  - expr → expr + term | expr - term | term
    - expr = {Lista de términos separados por operadores ‘+’ o ‘-’)
    - ={Lista de mul-o-div-sub-expresiones separadas por operadores‘+’o ‘-}
  - term → term * factor | term / factor | factor
    - term = {Lista de factores separados por operadores ‘*’o ‘/’ }
    - = { mul-o-div-sub-expresiones}
    - ={Lista de operadores primitivos (incluyendo parentesis-cerrando-expr) separados por operadores ‘*’o ‘/’}
  - factor → digit | (expr)
    - factor = {operadores primitivos o/incluyendo parentesis-cerrando-expr }

Modelado Formal con Gramáticas

Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- La traducción del lenguaje fuente es dirigida totalmente por el analizador (parser)
- El proceso de análisis y los árboles de análisis se usan para dirigir el análisis semántico y la traducción del lenguaje a formatos entendibles por máquinas.
- Se mejorando la gramática convencional con información para controlar el análisis semántico y la traducción.
- Estas gramáticas se llaman gramáticas con atributos.
- Una traducción dirigida por sintaxis es un formalismo para especificar las traducciones para las construcciones en función de atributos asociados con sus componentes sintácticos.
- Utiliza una gramática independiente de contexto para especificar la estructura sintáctica de la entrada.

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- La idea es asociar con cada símbolo de la gramática:
  - Con un conjunto de atributos
    - Sintetizados: su valor en un nodo del árbol de análisis sintáctico se determina a partir de los valores de atributos de los hijos de ese nodo (como decir de abajo hacia arriba). Se pueden calcular mediante un solo recorrido ascendente del árbol de análisis sintáctico, lo que es muy deseable.
    - Heredados: su valor en un nodo de un árbol de análisis sintáctico está definido a partir de los atributos en el padre y los hermanos de dicho nodo. Éstos sirven para expresar la dependencia de una construcción de un lenguaje de programación en el contexto en el que aparece.
  - Con un conjunto de reglas semánticas
    - para calcular los valores de los atributos asociados con los símbolos que aparecen en esa producción.

Modelado Formal con Gramáticas

Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- Se mejora o aumenta la gramática asociando atributos a cada símbolo que describe sus propiedades
  - Un atributo tiene un nombre y un valor asociado: cadena de caracteres, número, tipo, ubicación en memoria, registro. (Cualquier información que se necesite).
  - Por ejemplo:
    - variables
    - atributo “tipo” -> registra el tipo declarado para una variable, necesario para un posterior control de tipo.
    - constante entera
    - atributo”valor” -> que puede se necesario más tarde para generar el código.

Modelado Formal con Gramáticas

Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- Con cada regla de producción en una gramática definimos reglas semánticas o “acciones”.
- Describen cómo computar los valores de los atributos asociados con cada símbolo de la gramática en una regla de producción.
- El valor del atributo para un nodo del análisis puede depender de información de sus nodos hijos, de sus pares o padres.
  - Ejemplo de regla de producción aumentada con un conjunto de acciones que usan el atributo “valor” de un nodo dígito para almacenar el valor numérico adecuado
    - X.a -> atributo a asociado con el símbolo X
-

```
digit    -> 0   {digit.value = 0}
          |  1   {digit.value = 1}
          |  2   {digit.value = 2}
          ...
          |  9   {digit.value = 9}
```

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- Los atributos pueden ser pasados hacia arriba en el árbol de análisis, para ser usados por otras producciones.

```
int1     -> digit        {int1.value = digit.value}
         |  int2 digit    {int1.value = int2.value*10 + digit.value}
```

Nota: los sufijos 1 y 2 denotan diferentes instancias del mismo símbolo no-terminal

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- Pregunta: luego de los resultados de un análisis sintáctico (parse tree), cómo lo transformo en una representación intermedia ?
  - Cómo especificar “reglas de traducción”
  - mapeo “entrada => salida”
  - Ejemplo: reglas de traducción infix-a-postfix
  - E.p = E si E es una variable o constante
  - E.p = E1.p E2.p op si E → E1 op E2
  - E.p = E1.p si E → (E1) [entre paréntesis]
  - Traducir desde sub-expresiones locales, luego propagarlas a las padres o superiores

- Qué se hizo ?
  - Una aproximación dirigida por sintaxis
  - Se asocia cada estructura local con un conjunto de reglas o acciones de traducción
  - Se guardan algunas variables (atributos) para cada símbolo LHS

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- Atributos asociados con las construcciones:
  - mantener la información necesaria para la traducción (o la comprobación semántica)
  - por ejemplo, el tipo, de cadena, ubicación en memoria, o lo que sea
- Dos maneras de especificar el proceso de la traducción dirigida por sintaxis
  - SDD: definición dirigida por sintaxis.
    - Especificación formal de la traducción.
    - Especificar la traducción de una construcción en términos de atributos asociados con componentes sintácticos
  - TS: esquema de traducción
    - Notación procedural para especificar traducciones

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

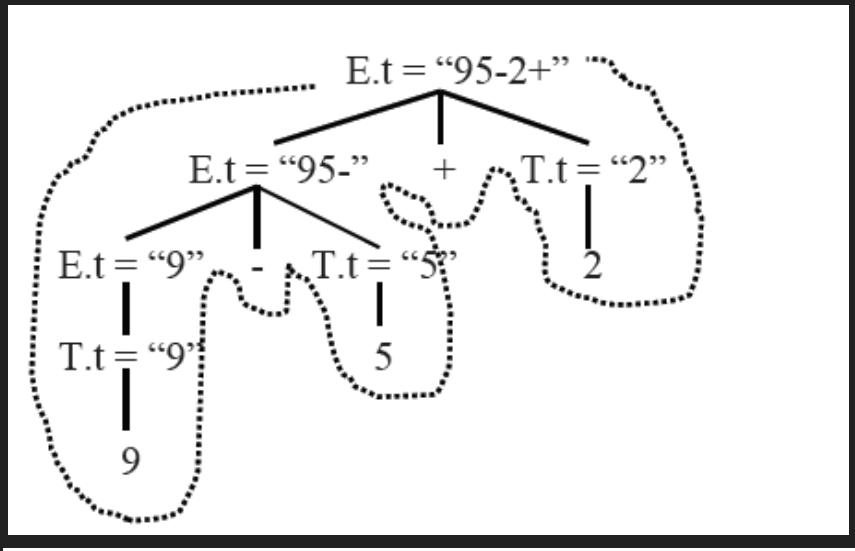
- SDD
- Entrada: CFG, dónde CFG especifica la sintaxis
  - Cada símbolo de la gramática ↔ se anota con un conjunto de atributos para almacenar los resultados de las traducciones locales de los sub-árboles o estructuras sitácticas parciales.
  - Atributos auxiliares: cada producción ↔ se anota con un conjunto de reglas semánticas para computar los valores de los atributos de los símbolos gramaticales de la producción (de la forma de atributos de los padres, hermanos o hijos)
- Salida: árbol de análisis sintáctico con anotaciones (con anotación de atributo)
- Proceso de Traducción para una entrada x basada en SDD
  - 1. Construir el árbol de análisis de x
  - 2. **X.a** (atributo de **X**) en el nodo **n** es evaluado usando las reglas semánticas para el atributo **a** asociado a la producción **X**

Modelado Formal con Gramáticas
Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)

- SDD aplicado a traducción Infix-a-Postfix
- Reglas de Producción:
  - expr → expr1 + term
  - expr → expr1 - term
  - expr → term
  - term → 0
  - term → 1
  - …
- Reglas Semánticas:
  - expr.t := expr1.t || term.t || '+'
  - expr.t := expr1.t || term.t || '-'
  - expr.t := term.t
  - term.t := '0'
  - term.t := '1'
  - …

Modelado Formal con Gramáticas

Traducción Dirigida por la Sintaxis
Syntax Directed Translation (SDT)




```
E'   -> E          { printf("%d\n", E.val); }

E    -> T          { E.val = T.val; }
      | E A T      { switch(A.op) {
                       case ADD: E.val = E.val + T.val; break;
                       case SUB: E.val = E.val - T.val; break; }
                   }
T    -> F          { T.val = F.val;}
      | T M F      { switch(M.op) {
                       case MUL: T.val = T.val * F.val; break;
                       case DIV: T.val = T.val / F.val; break; }
                   }
F    -> (E)        { F.val = E.val; }
      | int        { F.val = int.val; }

A    -> +          { A.op = ADD; }
      | -          { A.op = SUB; }

M    -> *          { M.op = MUL; }
      | /          { M.op = DIV; }
```

Modelado Formal con Gramáticas
Análisis Semántico

- El análisis semántico es la tarea de asegurar que las declaraciones y declaraciones de los programas son semánticamente correctos.

- También asegura que su significado es claro y consistente con la forma en que las estructuras de control y los datos deben ser utilizados.

Modelado Formal con Gramáticas
Análisis Semántico

- El análisis semántico típicamente involucra:
  - Comprobación de tipos (type checking)
    - Los tipos de datos se utilizan de una manera que es consistente con su definición (es decir, sólo con tipos de datos compatibles, sólo con operaciones que son definido para ellos, etc.)
  - Comprobación de etiquetas (labels)
    - Las referencias de etiquetas en un programa deben existir.
  - Controles de control de flujo
    - Se deben utilizar estructuras de control en su forma apropiada (no GOTOs), declaración DO, sin interrupciones fuera de un bucle.

Modelado Formal con Gramáticas
Análisis Semántico

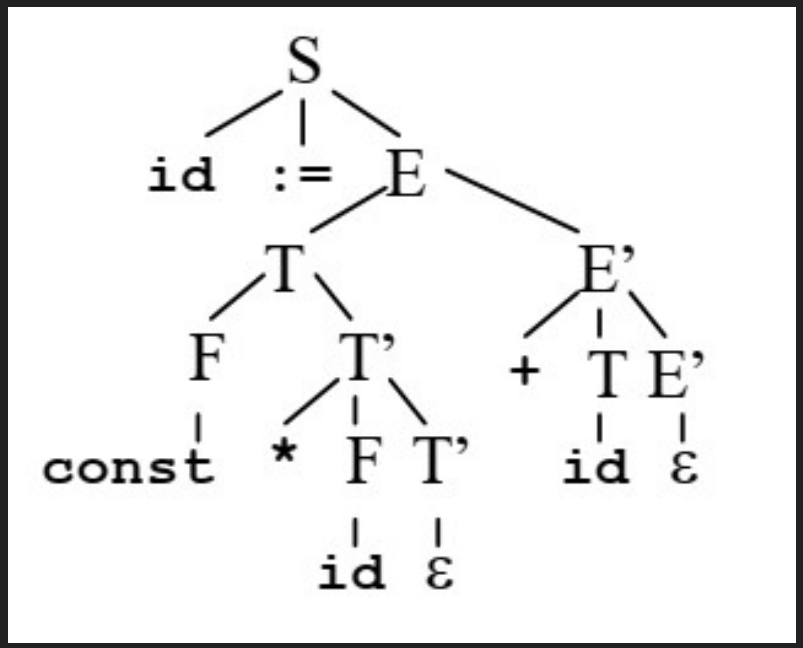
- Barrido
  S → id := E
  E → T E'
  E' → + T E'
  E' → ε
  T → F T'
  T' → * F T'
  T' → ε
  F → id
  F → const
  F → (E)

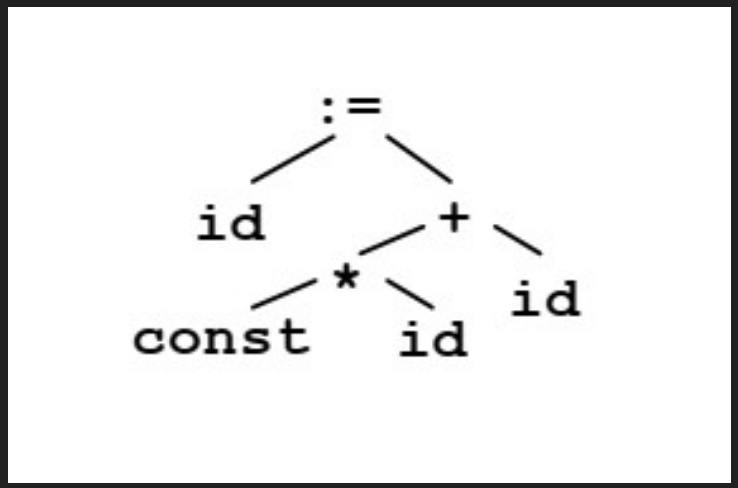
- Insertamos acciones
  S → id {*pushid*} := {*pushassn*} E {*buildassn*}
  E → T E'
  E' → + {*pushop*} T {*buildexpr*} E'
  E' → ε
  T → F T'
  T' → * {*pushop*} F {*buildterm*} T'
  T' → ε
  F → id *{pushid}*
  F → const *{pushconst}*
  F → (E) *{pushfactor}*

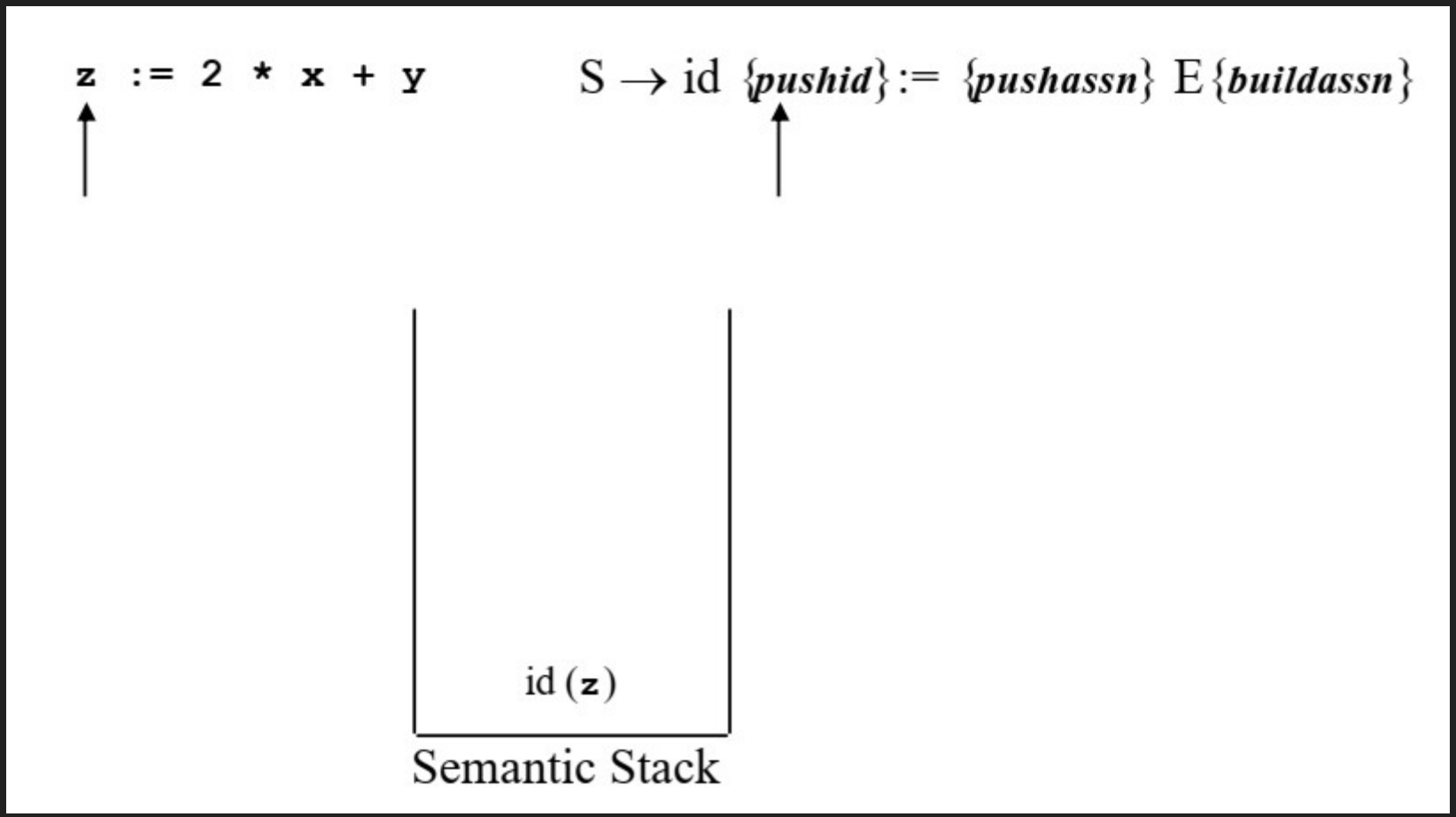
Modelado Formal con Gramáticas
Análisis Semántico

z := 2 * x + y

AST
(Abstract Syntax Trees)

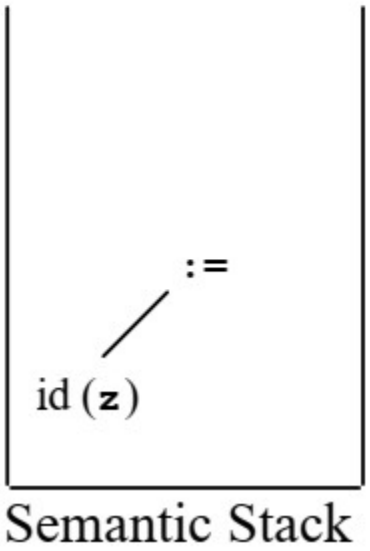


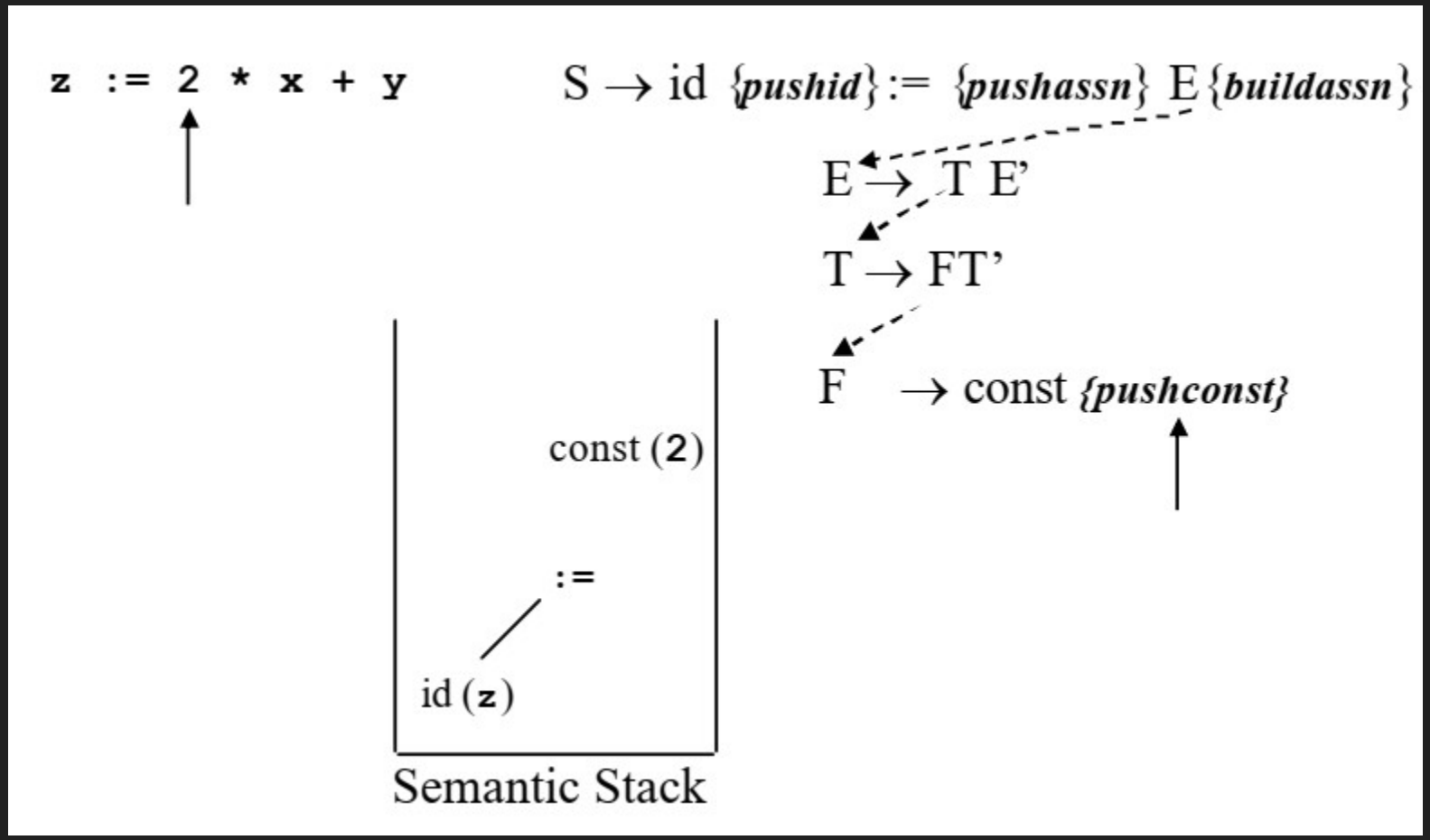


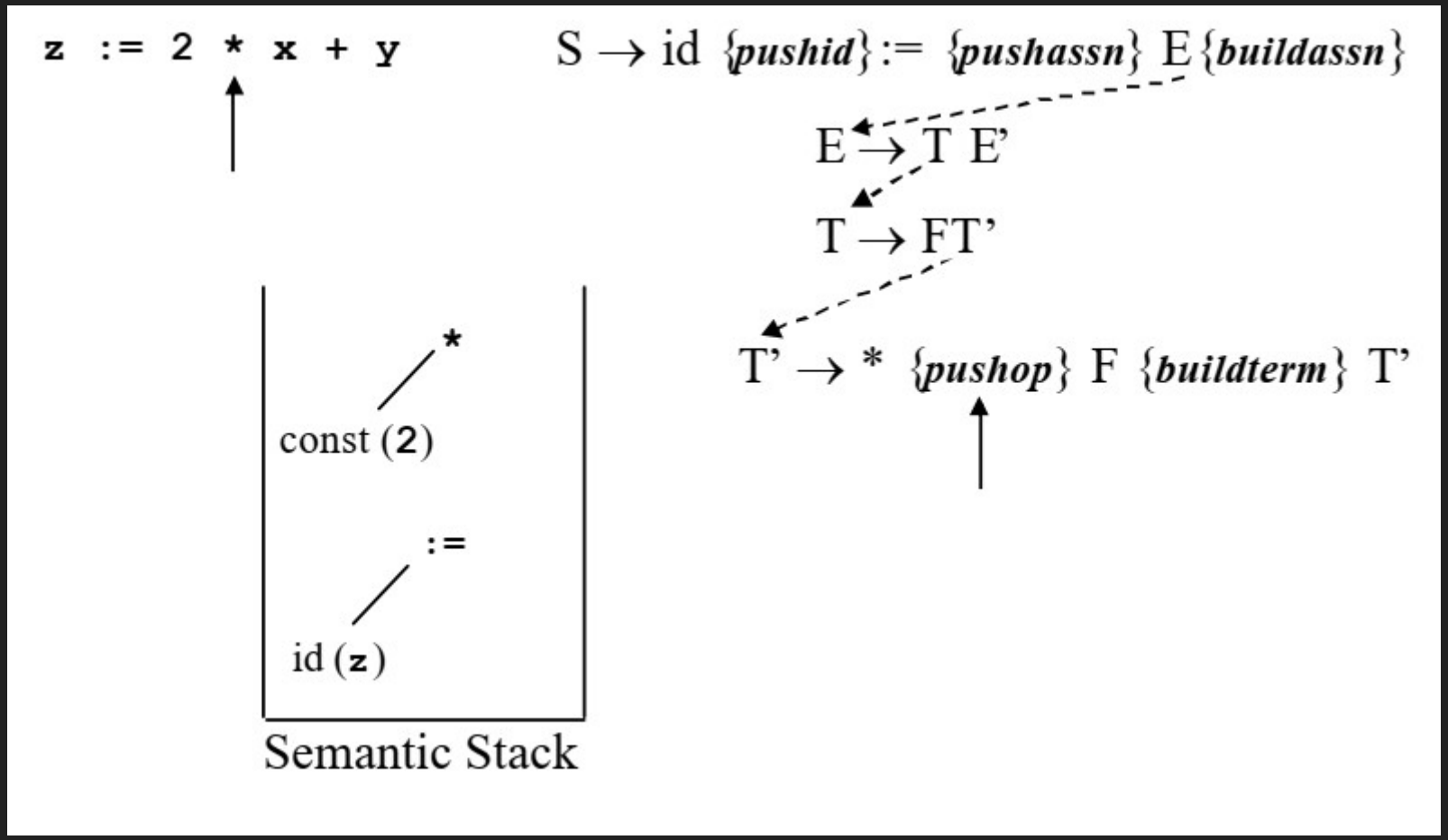

z := 2 * x + y
S → id {pushid} := {pushassn} E {buildassn}
id (z)
Semantic Stack

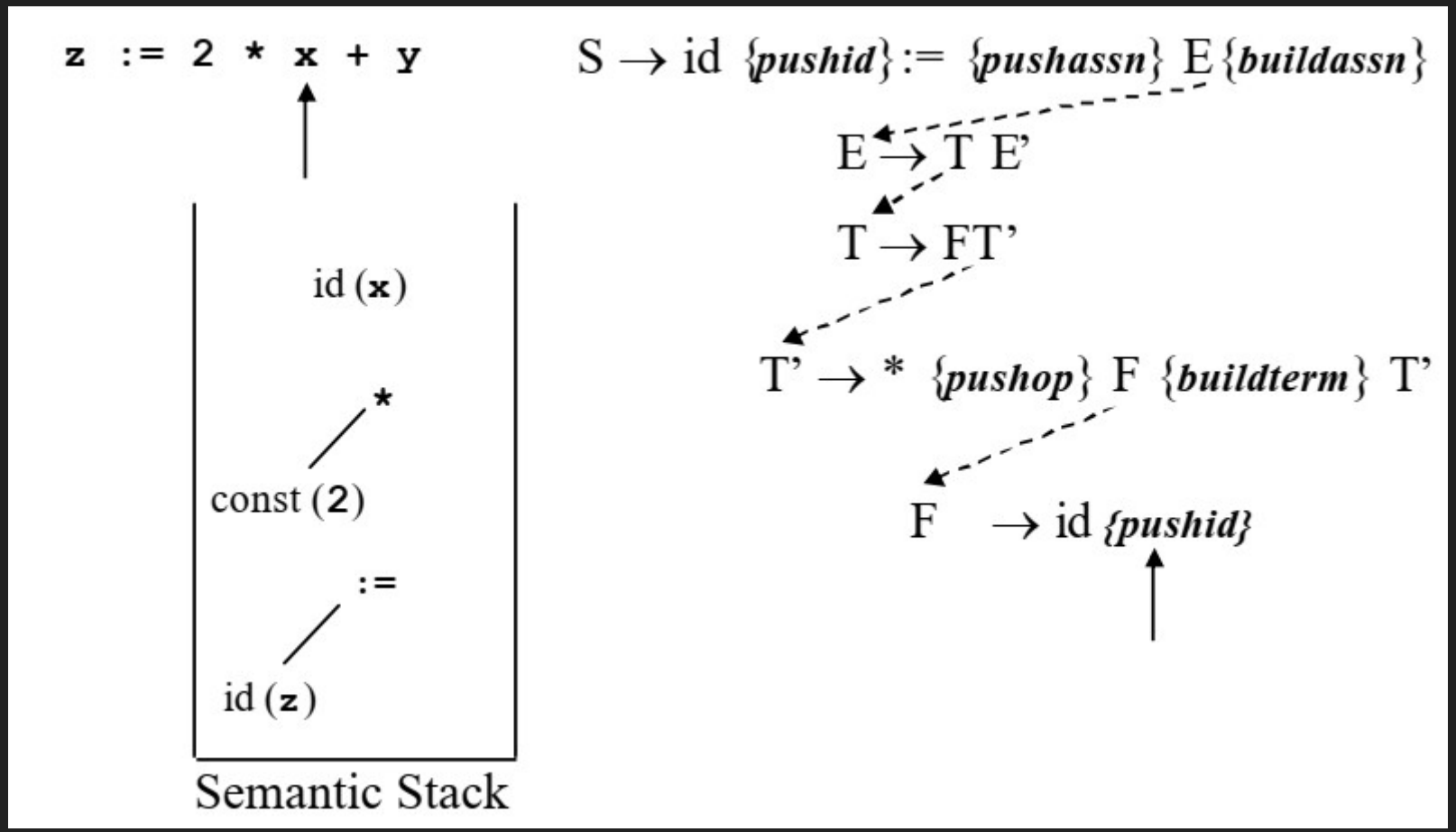
z := 2 * x + y

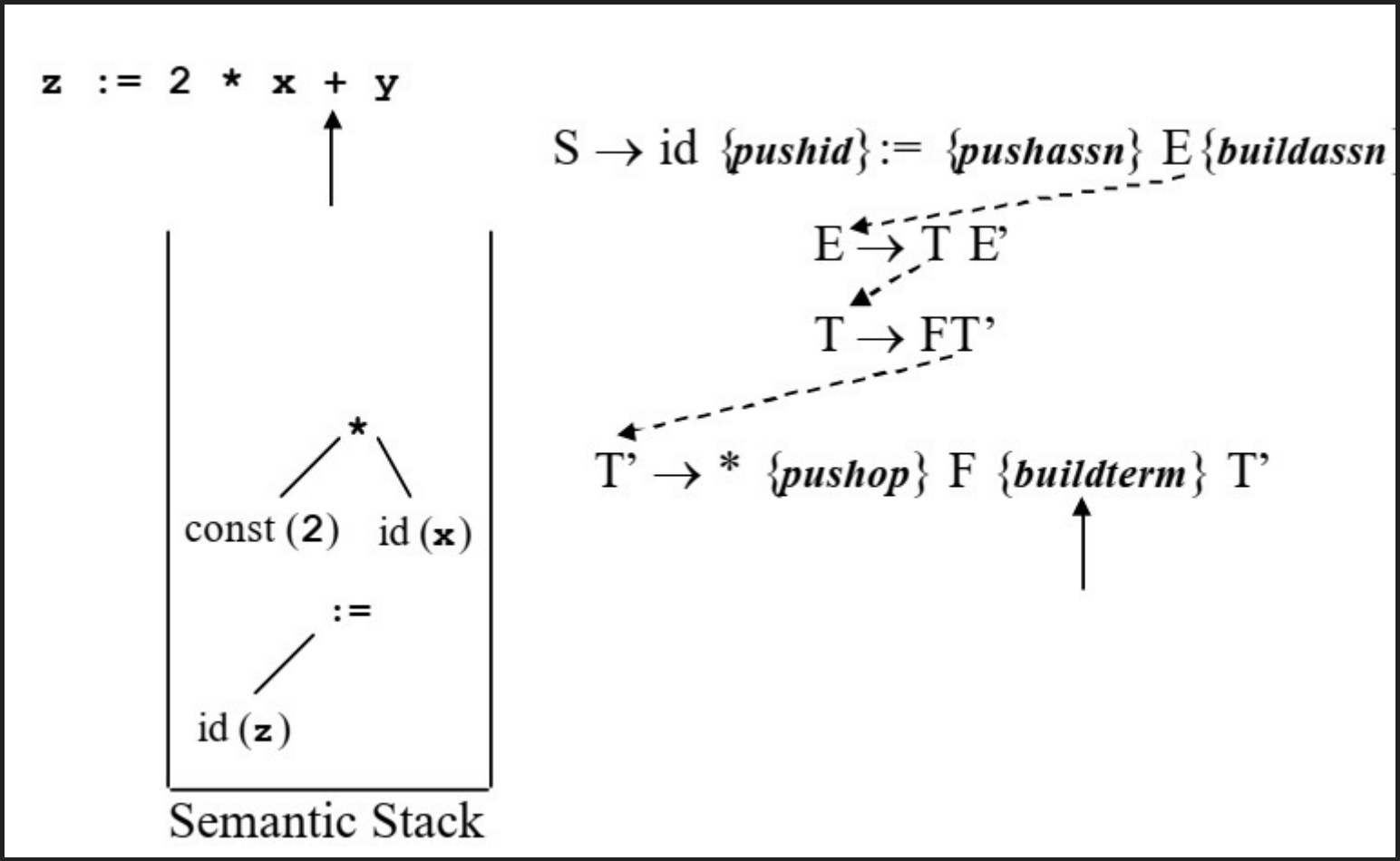
$S \rightarrow \text{id}\ \{pushid\} := \{pushassn\}\ E\ \{buildassn$

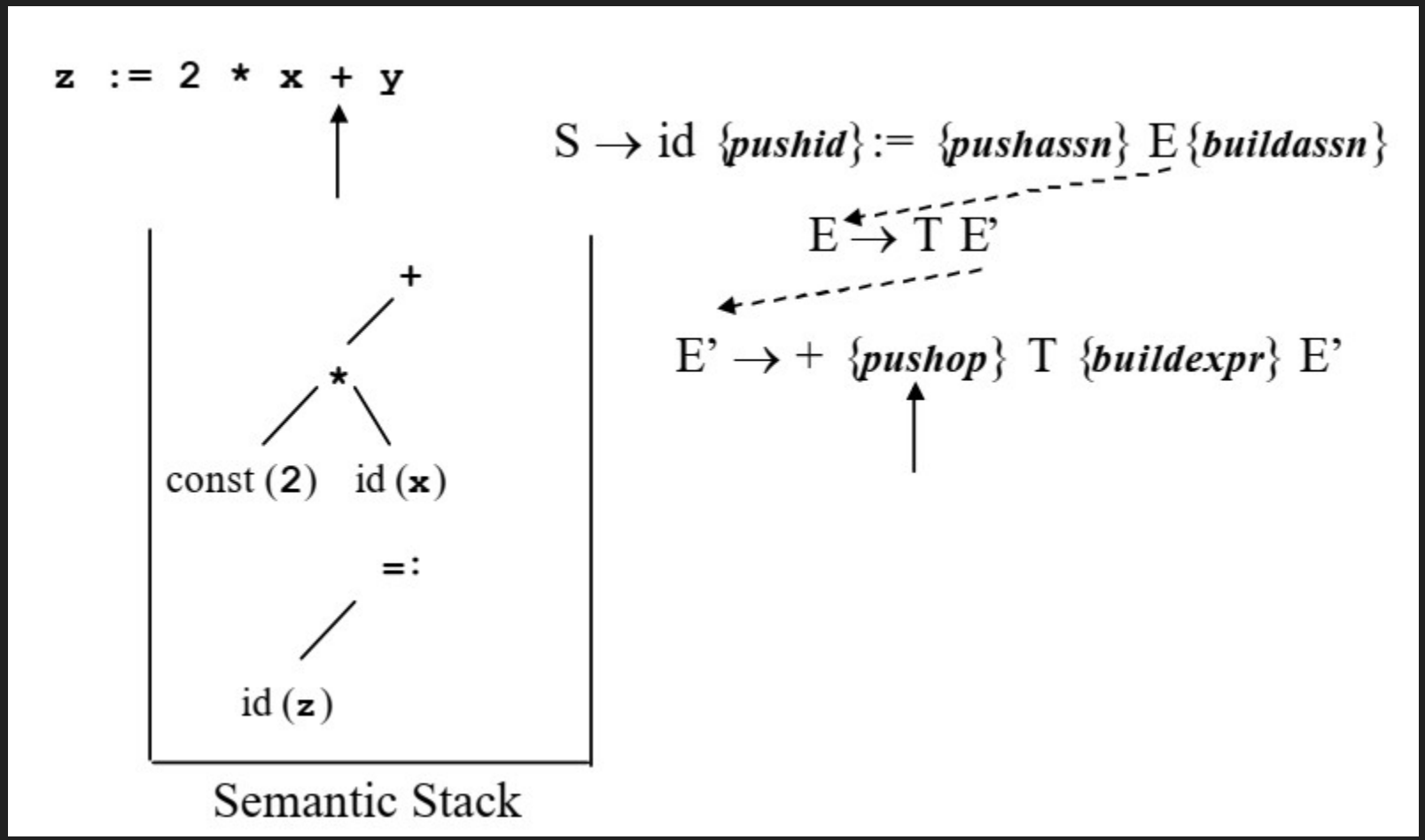


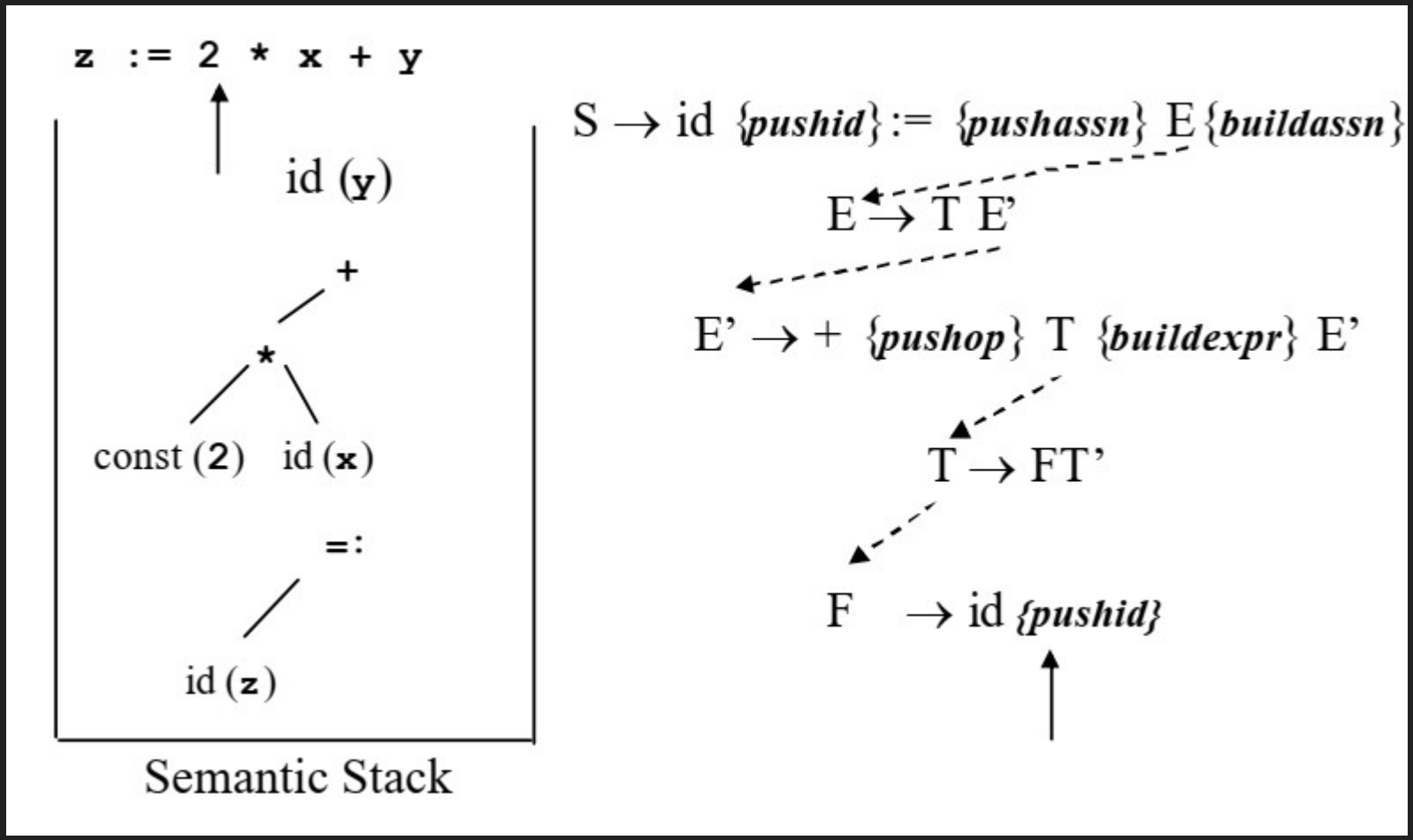

z := 2 * x + y
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
T → FT'
F → const {pushconst}
const (2)
:=
id (z)
Semantic Stack

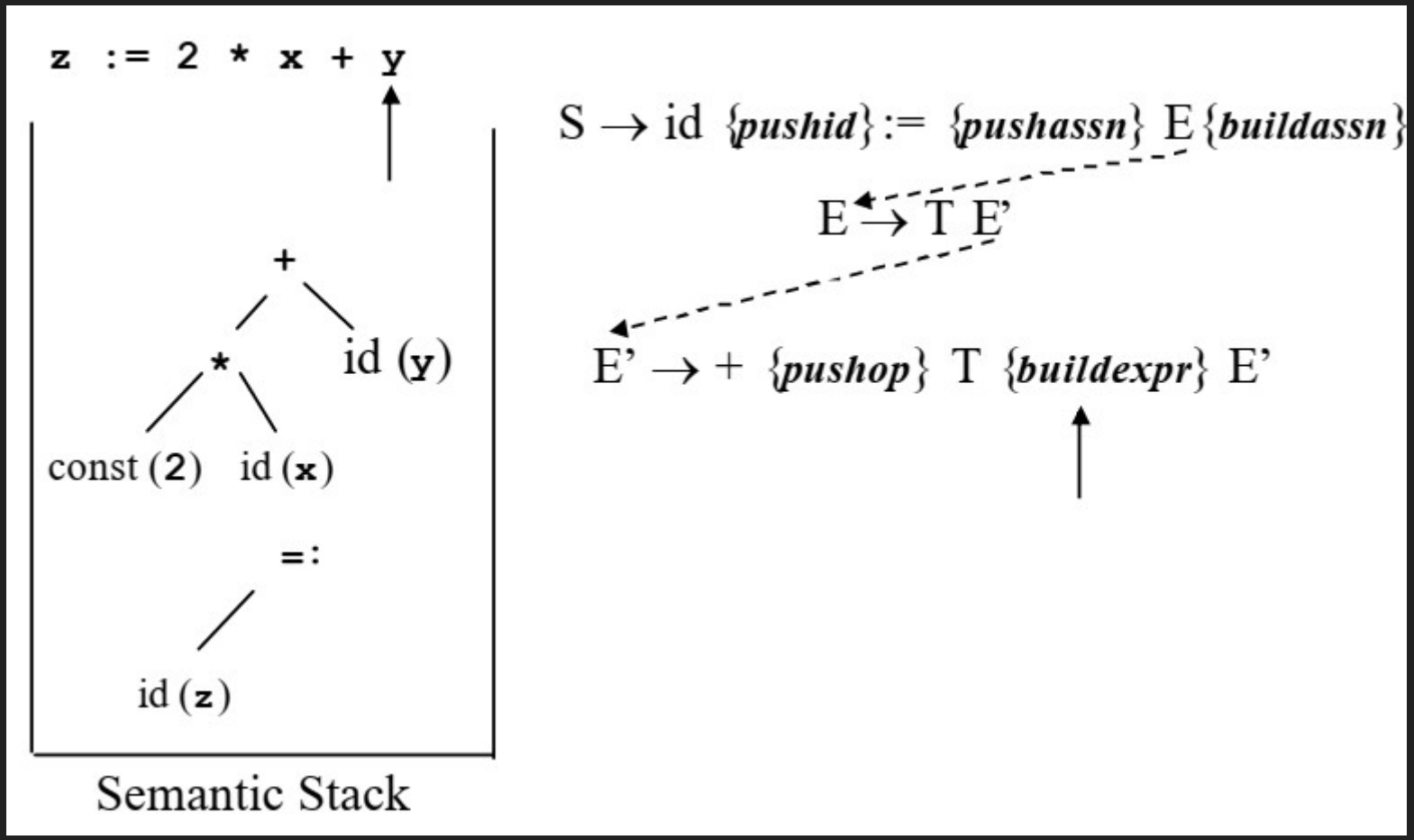

z := 2 * x + y
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
T → FT'
T' → * {pushop} F {buildterm} T'
*
const (2)
:=
id (z)
Semantic Stack


z := 2 * x + y
id (x)
*
const (2)
:=
id (z)
Semantic Stack
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
T → FT'
T' → * {pushop} F {buildterm} T'
F → id {pushid}


z := 2 * x + y
S → id {pushid} := {pushassn} E {buildassn
E → T E'
T → FT'
T' → * {pushop} F {buildterm} T'
*
const (2)
id (x)
:=
id (z)
Semantic Stack


z := 2 * x + y
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
E' → + {pushop} T {buildexpr} E'
+
*
const (2)
id (x)
=:
id (z)
Semantic Stack


z := 2 * x + y
id (y)
+
*
const (2)
id (x)
=:
id (z)
Semantic Stack
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
E' → + {pushop} T {buildexpr} E'
T → FT'
F → id {pushid}


z := 2 * x + y
+
*
id (y)
const (2)
id (x)
=:
id (z)
Semantic Stack
S → id {pushid} := {pushassn} E {buildassn}
E → T E'
E' → + {pushop} T {buildexpr} E'

z := 2 * x + y

S → id {*pushid*} := {*pushassn*} E {*buildassn*}