



Facultad de Ingeniería

MENOS ERRORES. MÁS CONFIANZA. CÓDIGO QUE SE VALIDA SOLO

Equipo 06

Arias Hernández Javier
Gallegos García Alejandro
Miranda Galván Erick Santiago
Ortíz Briseño Emiliano
Torres Delgadillo Samuel Mixcoatl

Ing. Rene Adrián Dávila Pérez

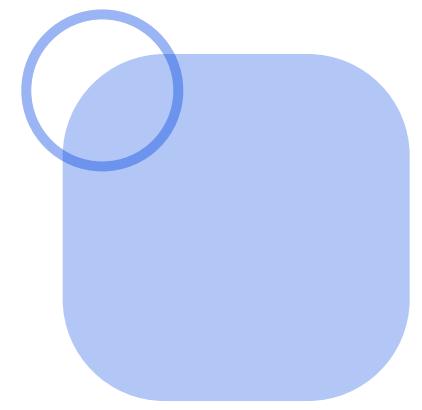
```
    requests.get(url)

    response.raise_for_status()
    if response.status_code != 200:
        raise Exception(f"Status: {response.status_code} - {response.text}")

    soup = BeautifulSoup(response.content, 'lxml')
    images = soup.find_all("img", attrs={"src": True})
    for image in images:
        print(image['src'])
```



EL PROBLEMA REAL HOY



- 01** Integraciones fallan por inconsistencias de tipos y declaraciones tardías
 - 02** Equipos pierden horas depurando faltas de “;” y divisiones entre cero
 - 03** La validación manual es lenta y poco repetible
- 

¿QUIÉN USARÁ EL PARSER+SDT?

T

Profesionales y estudiantes de ingeniería en computación, desarrollo y QA que buscan reducir errores.
Integrar herramientas de automatización, validación temprana y estandarización en sus flujos CI/CD

Perfil del cliente



Docentes e investigadores de compiladores; estudiantes de programación; equipos de desarrollo y QA en startups y pymes.

- Universidades y bootcamps que imparten análisis sintáctico/semántico.
- Startups y áreas de TI con pipelines CI/CD y cultura de automatización.
- Pymes tecnológicas que requieren mensajes de error claros y control temprano de calidad.

Segmento de adopción



UN ANALIZADOR QUE ENTIENDE LO QUE LEES



01

Parsing determinista + SDT en una sola etapa

El sistema realiza análisis sintáctico y semántico de forma simultánea. Cada regla de la gramática ejecuta acciones SDT que validan diversas reglas.

02

Alineado a categorías léxicas clave

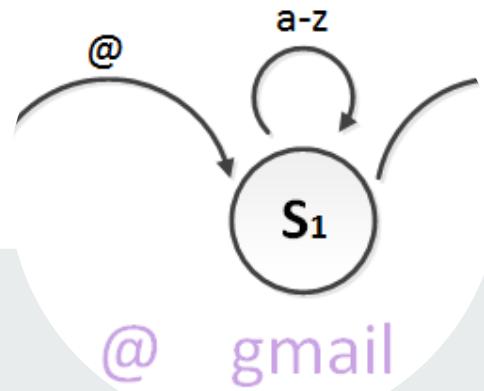
Opera sobre las categorías KEYWORD, IDENTIFIER, OPERATOR, CONSTANT y PUNCTUATION, garantizando que cada token sea reconocido.

03

Comprobación inmediata con tabla de símbolos

Durante el parseo, el sistema consulta la tabla de símbolos para validar tipos y confirmar compatibilidad.

¿QUÉ HACE NUESTRO PARSER+SDT?



Valida declaraciones, asignaciones, expresiones y print

Soporta $S \rightarrow D \mid P$, $D \rightarrow T$
id = E ;,
 $P \rightarrow \text{print}(\text{STRING})$; y
expresiones con jerarquía.



Aplica precedencias y detecta errores clave

Falta de ;, type mismatch, división entre cero y carácter ilegal, con clasificación explícita.



Devuelve mensajes claros de resultado

Confirma Parsing Success! y SDT Verified! cuando la verificación sintáctica y semántica es correcta.

DIMENSIÓN	DE
0	0
0	1
1	2
2	0

Opera sobre categorías léxicas y tabla de símbolos

Reconoce KEYWORD, IDENT, OP, CONST, PUNCT y actualiza símbolos durante el parseo.

GRAMÁTICA BASE IMPLEMENTADA

Producciones principales

$S \rightarrow D \mid P$	-Asignación o impresión
$D \rightarrow T \ id \ = \ E ;$	-Asignación
$T \rightarrow \text{int} \mid \text{float}$	-Tipo de dato
$P \rightarrow \text{print} \ (\text{STRING}) ;$	-Impresión
$E \rightarrow E + M \mid E - M \mid M$	-Operaciones menor jerarquía
$M \rightarrow M * F \mid M / F \mid F$	-Operaciones mayor jerarquía
$F \rightarrow \text{number} \mid \text{decimal} \mid id \mid (\ E \)$	

Operadores y precedencia

E agrupa + y - (menor jerarquía).

M agrupa * y / (mayor jerarquía).

Paréntesis en F fuerzan el orden.

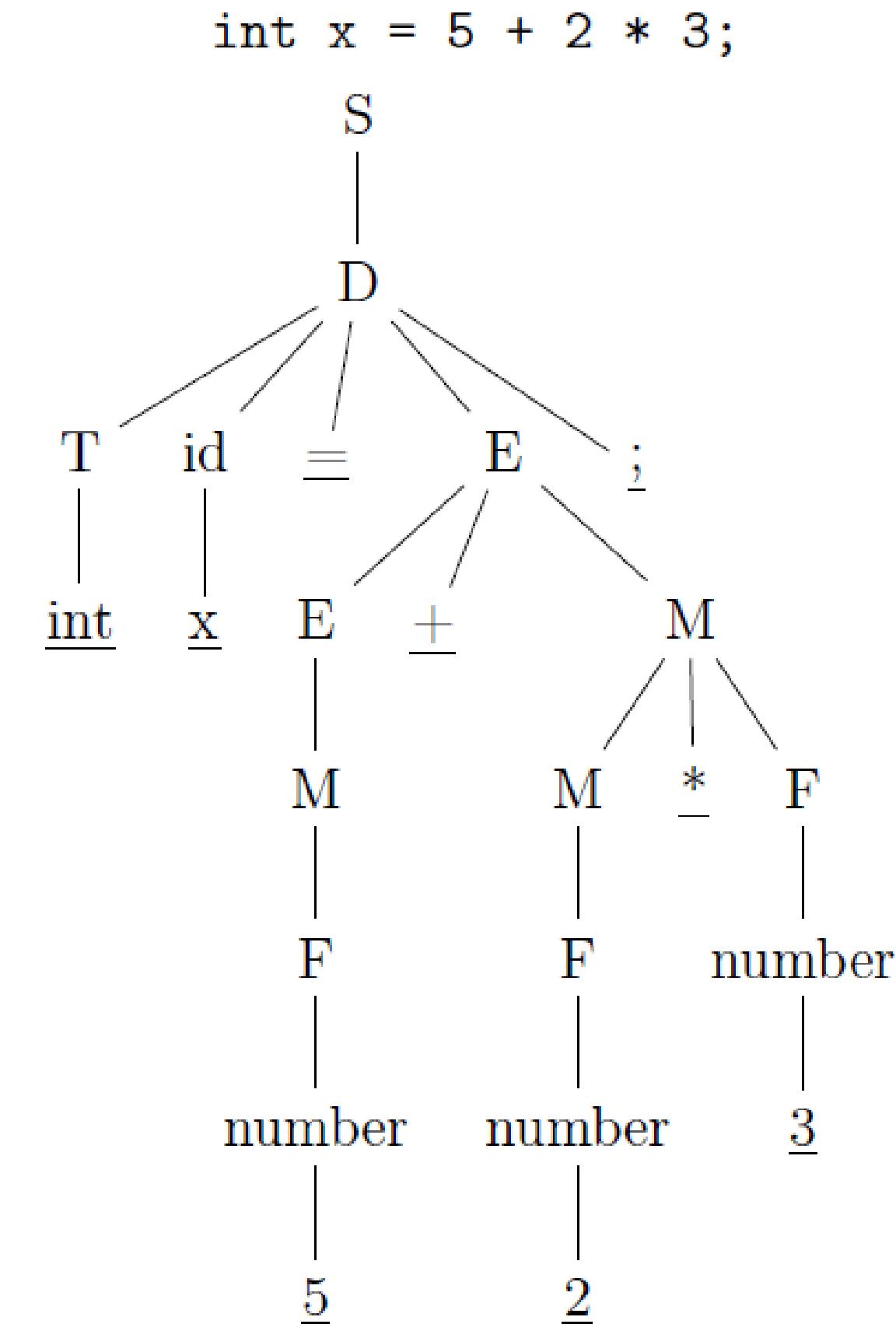
EJEMPLO PARSEADO

Como se desarrolla:

La gramática genera el árbol donde $2 * 3$ se resuelve antes que $5 + (...)$.

Orientación del diseño

Gramática simple apta para LL/LR, con tokens: KEYWORD, IDENTIFIER, OPERATOR, CONSTANT, PUNCTUATION. Optimizada para acciones SDT durante el parse.



SDT: REGLAS CLAVE QUE DAN “INTELIGENCIA”



Comprobación de tipos en asignaciones real y creciente

$$D \rightarrow T \ id \ = \ E ; \ \{ \text{if } (T = \text{int}) \wedge (E.type = \text{int}) \text{ then } ok \text{ else error} \}$$

Valida tipo declarado vs valor calculado antes de aceptar la asignación.

Acción semántica en print

$$P \rightarrow \text{print(string)} ; \ \{ \text{print(string.value)} \}$$

Ejecuta la acción al reducir la producción y garantiza el uso correcto del literal.

SDT: REGLAS CLAVE QUE DAN “INTELIGENCIA”



Propagación de valores y tipos en expresiones

$$E \rightarrow E_1 + M \quad \{E.value = E_1.value + M.value\}$$

$$E \rightarrow E_1 - M \quad \{E.value = E_1.value - M.value\}$$

$$M \rightarrow M_1 * F \quad \{M.value = M_1.value * F.value\}$$

Evalúa y propaga atributos durante el parseo para resultados consistentes.

Guardas semánticas: división entre cero

$$M \rightarrow M_1 / F \quad \{\text{if } F.value = 0 \text{ then error else } M.value = M_1.value / F.value\}$$

Evita ejecuciones inválidas y reporta el error de forma inmediata.

DETECCIÓN DE ERRORES EN TIEMPO DE PARSEO



Error Type	Example	Description
Lexical Error	<code>int \$x = 10;</code>	Invalid character detected by the lexer. The system reports Illegal character '\$'.
Syntax Error	<code>int x = 10</code>	Missing semicolon or incorrect token order. Reported as Parsing error in token '10'.
Semantic Error. Type mismatch	<code>int x = 3.14;</code>	Type mismatch between declared variable and assigned value. Reported as SDT error....
Semantic Error. Division by zero	<code>float y = 10 / 0;</code>	Arithmetic evaluation with division by zero. Detected by SDT rules and reported as a semantic error.

RESULTADOS: EJECUCIÓN 1 - INICIO

Compilación de tablas LALR

```
PS C:\Users\echav\OneDrive\Documentos\Compiladores\unam.fi.compilers.g5.06-main\parser> python src/main.py
Generating LALR tables
Parser & SDT
Escribe 'exit' para salir.
```

Al correr `python src/main.py` aparece “Generating LALR tables”. La gramática se compila a tablas LR usadas por el motor de parsing.

Listo para operar en modo interactivo

Generating LALR tables
Parser & SDT

Se muestran “Parser & SDT” y “Escribe ‘exit’ para salir.” El sistema queda en espera de sentencias, sin parsear nada todavía.

RESULTADOS: EJECUCIÓN 2 – ESTRUCTURA DEL PROYECTO

01

lexer.py

Realiza el análisis léxico del código fuente, detectando tokens válidos y reportando símbolos ilegales.

02

****archivos generados automáticamente por PLY**

Contienen las tablas LR y los estados de análisis. Confirman que el build del proyecto es totalmente reproducible.

03

main.py

Actúa como interfaz entre Parser + SDT. Permite validar y probar código directamente desde la consola.

pycache

lexer.py

main.py

parser.out

parser.py

parsetab.py

RESULTADOS: EJECUCIÓN 3 – CASO CORRECTO

T

Entrada reconocida correctamente

```
>> int x = 10;
```

El analizador ejecuta el ciclo completo léxico-sintáctico-semántico, confirmando que la sentencia cumple las reglas de la gramática y del SDT.

Salida del sistema

```
>> int x = 10;  
Parsing Success!  
SDT Verified!
```

Indica que todos los tokens fueron válidos y las acciones semánticas se ejecutaron sin conflicto de tipos.

RESULTADOS: ERRORES TÍPICOS

Falta “;” – error sintáctico

```
>> int x = 10  
Parsing error...
```

El flujo termina en CONSTANT y viola la producción $D \rightarrow T \text{id} = E ;$
El driver reporta Parsing error... antes de cualquier acción SDT.

Type mismatch – error semántico

```
>> int x = 10.5;  
Parsing Success!  
SDT error... (Type mismatch)
```

El parseo es correcto, pero la regla SDT
 $D \rightarrow T \text{id} = E ; \{ \text{if } (T = \text{int}) \wedge (E.\text{type} = \text{int}) \text{ then ok else error} \}$
detecta inconsistencia y emite Parsing Success! seguido de SDT error... (Type mismatch).

RESULTADOS: EJECUCIONES COMPLEMENTARIAS

Declaración correcta con valor decimal

```
>> float i6=7.2;  
Parsing Success!  
SDT Verified!
```

El analizador reconoce la variable, ejecuta acciones SDT.

Demuestra la consistencia entre tipo declarado y valor numérico.

Sentencia con instrucción de salida (print)

```
>> print("Example")  
Output: Example  
Parsing Success!  
SDT Verified!
```

Se ejecuta la acción SDT y muestra el resultado.

Confirma la correcta evaluación semántica de la producción

FINALIZACIÓN DEL PROCESO INTERACTIVO

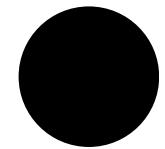
T

Declaración correcta con valor decimal

```
exit
C:\Users\echav\OneDrive\Documentos\Compiladores\unam.fi.compilers.g5.06-main\parser>
```

El sistema cierra el modo interactivo y regresa al entorno del intérprete.
Se conserva el estado de símbolos y finaliza de forma segura.

IMPACTO ESPERADO



Menos tiempo en depuración

Impacto: Se acorta el ciclo de corrección.

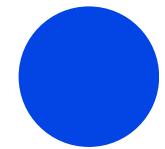
Cómo se logra: Validación léxica-sintáctica-semántica



Menos incidencias por errores

Impacto: Baja la tasa de fallos como ; faltante, type mismatch y división entre cero.

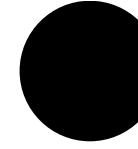
Cómo se logra: Reglas SDT y mensajes específicos.



Estandarización en equipos mixtos

Impacto: Menos retrabajo en handoffs

Cómo se logra: Reglas SDT, tabla de símbolos compartida y mensajes deterministas



Más previsibilidad y calidad

Impacto: Builds reproducibles y fácil integración a CI/CD.

Cómo se logra: Artefactos y CLI para pruebas automatizadas.



Equipo 6 Compiladores

GRACIAS

POR SU ATENCIÓN

Noviembre 2025



updateScr
type:finishSb
is},draggable
stopPropagation
s.context.cli
ext.scrollHeight
et{value}func
(left:this.co
prototype={st
urn this.ver
addClass("bo
rn newm(a,b
heck")pp=0=!