Ayudantía 1: Introducción y Fundamentos de los Lenguajes de Programación

Profesores: José Luis Martí Lara, Roberto Díaz Urra
Ayudantes:
Hugo Sepúlveda Arriaza
Gabriela Acuña Benito
Lucio Fondón Rebolledo
lucio.fondon@sansano.usm.cl

Universidad Técnica Federico Santa María
Departamento de Informática

Contacto

Ayudantes de Cátedra:

- Hugo Sepúlveda Arriaza ⇒ `hugo.sepulvedaa@sansano.usm.cl`
- Gabriela Acuña Benito ⇒ `gabriela.acuna@usm.cl`
- Lucio Fondón Rebolledo ⇒ `lucio.fondon@sansano.usm.cl`

Resumen Capítulo I

Un poco de historia

- **Lenguajes de Máquina**: Primeras pseudocomputadoras (Babbage, 1837). Ada Lovelace escribió los primeros programas que utilizaban la máquina de Babbage (1842). Turing (1936), Zuse (1941), ENIAC (1946).
- **Lenguajes de Assembly**: IBM 704 (1954).
- **Primeros Lenguajes de Alto Nivel**: FORTRAN (1957), LISP (1958), COBOL (1959).
- **Lenguajes Estructurados**: PASCAL (1970), C (1972) y ADA (1979).
- **Lenguajes Orientados a Objeto**: Smalltalk (1980), C++ (1983), Java (1995).
- **Lenguajes de Scripting** : PERL (1987), PYTHON (1991), JavaScript (1995), PHP (1995), Ruby (1995).

Paradigmas de Programación

- **Imperativo**: Basado en la máquina de Von Neumann. Ejecución secuencial, variables de memoria, asignación y E/S. Típicamente procedural.
  - C, Pascal, Fortran
- **Funcional**: Basado en cálculo Lambda. Usa funciones y recursión.
  - LISP, Scheme, Haskell.
- **Declarativo**: Se declara lo que se quiere hacer, no cómo. Ej: SQL.
  - **Lógico**: Basada en cálculo de predicados (lógica simbólica). Está fundamentalmente basado en reglas.
    - PROLOG
- **Orientado a Objetos**: Conjunto de objetos (piezas) que interactúan controladamente intercambiando mensajes.
  - Smalltalk, C++ y Java.

Abstracciones

Datos:

- Primitivos
- Simples
- Estructurados

Control:

- Sentencias
- Estructuras de Control
- Abstracción Procedural
- Concurrencia

Tipos de Datos Abstractos (TDA): Tipo definido por el usuario, que agrupa datos y operaciones. Abstrae el tipo de dato con sus operaciones, de la implementación del tipo.

Metodos de Implantación

1. **Compilación**:
   - Se traduce a lenguaje de máquina para su posterior ejecución
   - Ej: C, C++, Go
2. **Interpretación**:
   - Una máquina virtual interpreta directamente el código fuente durante la ejecución.
   - Ej: LISP, Python, Javascript
3. **Híbrido**:
   - Se compilan a un lenguaje intermedio, que luego es interpretado.
   - Ej: C#, Java

Resumen Capítulo II

Conceptos básicos

- **Lenguaje**
- **Sintaxis**
- **Semántica**
- **Alfabeto**
- **Gramática**

Gramática (TALF Posting)

Definición

Se define **gramática** (formal) como una 4-tupla $G = (V, \Sigma, P, S)$, en donde:

- V: Conjunto finito de símbolos no terminales o variables.
- Σ: Conjunto finito de símbolos terminales.
- P: Relación finita $V \Rightarrow (V \cup \Sigma)^*$, donde cada miembro de P se denomina regla de producción de G.
- S: Símbolo de partida, donde $S \in V$.

Las gramáticas permiten expresar formalmente la sintaxis de un lenguaje.

Ejemplo de Gramática

Sea $G = (V, \Sigma, P, S)$, con:

- $V = \{S\}$
- $\Sigma = \{\epsilon\}$ (donde ϵ es la cadena vacía).

Donde P tiene sólo una regla de producción con 3 posibles opciones:

$$S \rightarrow SS \mid (S) \mid \epsilon$$

Ejemplo de Gramática

Sea $G = (V, \Sigma, P, S)$, con:

- $V = \{S\}$
- $\Sigma = \{\epsilon\}$ (donde ϵ es la cadena vacía).

Donde P tiene sólo una regla de producción con 3 posibles opciones:

$$\boxed{S \rightarrow SS \mid (S) \mid \epsilon}$$

¿Qué lenguaje representa esta gramática?

Ejemplo de Gramática

Sea $G = (V, \Sigma, P, S)$, con:

- $V = \{S\}$
- $\Sigma = \{\epsilon\}$ (donde ϵ es la cadena vacía).

Donde P tiene sólo una regla de producción con 3 posibles opciones:

$$\boxed{S \rightarrow SS \mid (S) \mid \epsilon}$$

¿Qué lenguaje representa esta gramática?

El lenguaje de los paréntesis balanceados o Lenguaje de Dyck!

BNF

Definición

Notación BNF es un metalenguaje que permite especificar formalmente gramáticas libres de contexto.

```
<number> ::= <digit> | <number><digit>
<digit> ::= 0 | 1 | 2 | 3 | ··· | 9
```

BNF

Definición

Notación BNF es un metalenguaje que permite especificar formalmente gramáticas libres de contexto.

```
<number> ::= <digit> | <number><digit>
<digit> ::= 0 | 1 | 2 | 3 | ··· | 9
```

BNF que genera una cantidad arbitraria de números

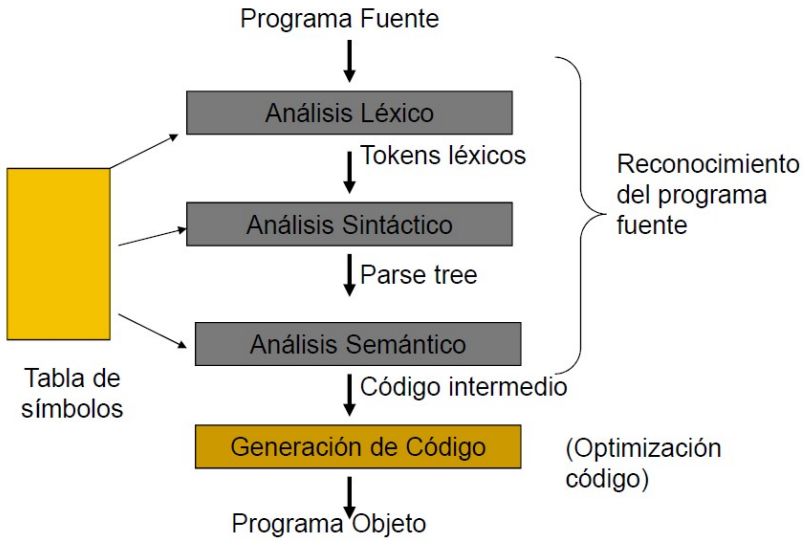
EBNF: Extensión de un BNF

Para facilitar la escritura de las BNF, se extienden a las EBNF, en donde se incorporan las siguientes notaciones:

- **Elementos opcionales**: Presentados entre corchetes → $[\cdots]$
- **Agrupaciones**: Paréntesis redondos agrupan elementos →(<exp>)
  - Se pueden usar agrupaciones con alternación → (<exp>|<exp>)
- **Repetición (0 ó más)**: Se indican con paréntesis de llaves → $\{\cdots\}$

```
<if-stmt> ::= if <condition> then <stmts> [ else <stmts>]
<identifier> ::= <letter> { <letter> | <digit> }
<for-stmt> ::= for <var> := <expr> (to | downto) <expr> do <stmt>
```

Proceso de Compilación (Análisis Léxico y Sintáctico)



- **Scanning (análisis léxico)**: El traductor lee secuencia de caracteres de programa de entrada, y se reconocen tokens con **expresiones regulares**.
- **Parsing (análisis sintáctico)**: El traductor procesa los tokens y se construye un Árbol Sintáctico (parse tree), que permite reconocer si el programa es sintácticamente correcto.

OJO: Ambigüedad se resuelve mediante reglas de asociatividad y precedencia.

Expresiones Regulares

Definición

- Permiten describir patrones de cadenas de carácteres dentro de un texto.
- Cada ER tiene asociado un autómata finito.

Símbolo	Significado
x^*	Frecuencia de 0 ó más veces
x+	Frecuencia de 1 ó más veces
x?	Aparece una vez o ninguna
xy	Concatenación
[xyz]	Captura cualquiera de los caracteres que esté dentro (x o y o z)
[0-9]	Rango de carácteres entre 0 y 9
(xy)	Agrupa elementos (se puede combinar con alternación)

Ejercicios

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. En la fase de parsing se construye el árbol sintáctico.
II. Una gramática, como aquella libre del contexto, permite expresar formalmente la semántica de un lenguaje.
III. C, C++ y C# utilizan compilación como método de implantación.
IV. Scheme es un lenguaje que utiliza el paradigma funcional.

a) I y II
b) II, III y IV
c) I y IV
d) I, II, III y IV

Ejercicio 1

¿Cuál(es) de la(s) siguiente(s) afirmación(es) es(son) correcta(s)?

I. En la fase de parsing se construye el árbol sintáctico.
II. Una gramática, como aquella libre del contexto, permite expresar formalmente la semántica de un lenguaje.
III. C, C++ y C# utilizan compilación como método de implantación.
IV. Scheme es un lenguaje que utiliza el paradigma funcional.

a) I y II
b) II, III y IV
c) I y IV
d) I, II, III y IV

R: **Alternativa c**

Ejercicio 2

Considere el siguiente BNF, en donde la operación ϕ es la de mayor precedencia, seguido de $\ominus$. La asociatividad para los dos operaciones son desde la izquierda.

<assign> ::= <id> := <expr>

<expr> ::= <expr> $\ominus$ $< expr >$ | $< expr > \phi < expr >$ | $< id >$

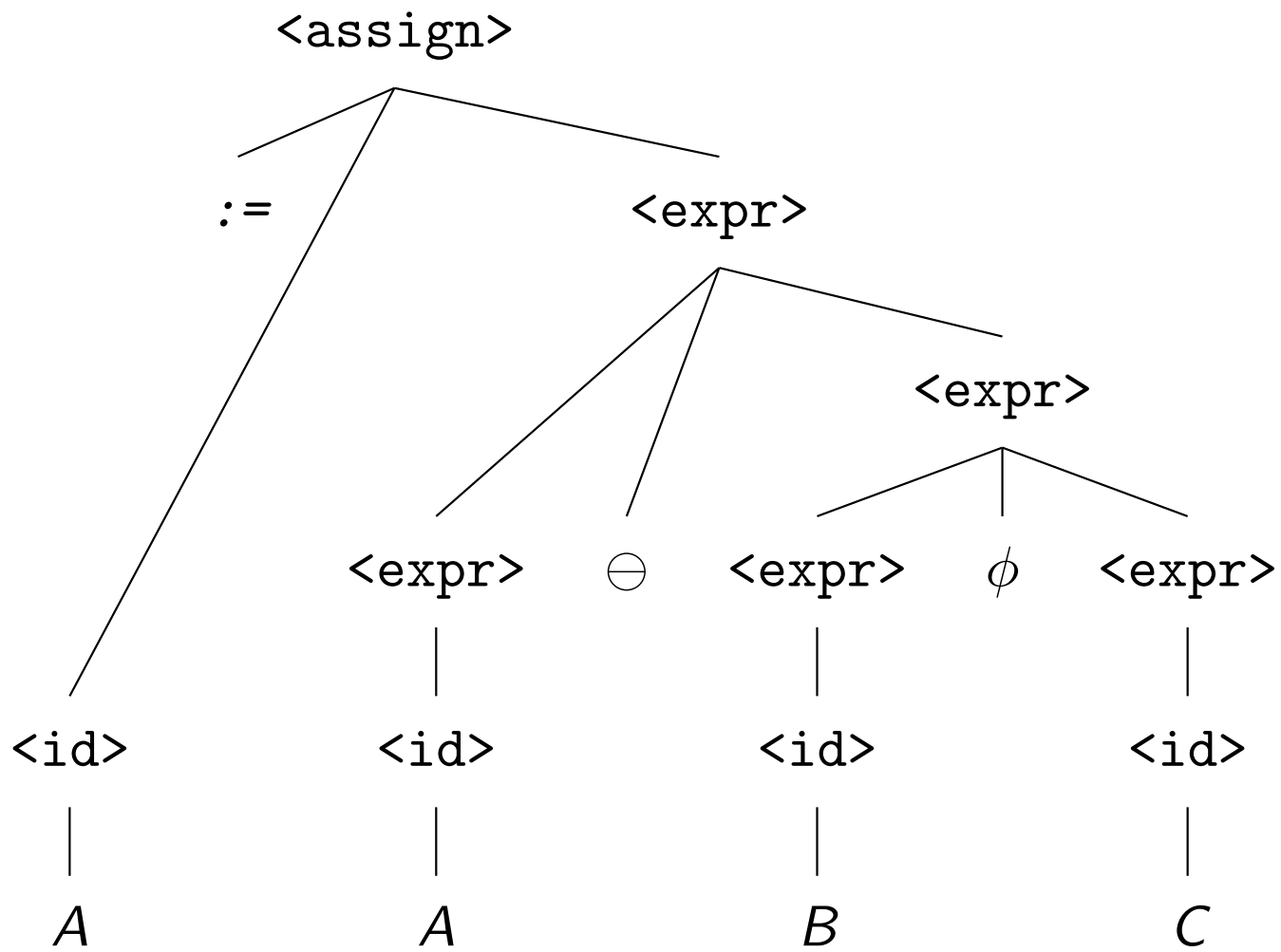
<id> ::= A | B | C

Realice el árbol sintáctico de las siguientes sentencias:

1. A:= A $\ominus$ B ϕ C
2. A:= A ϕ C $\ominus$ B ϕ A

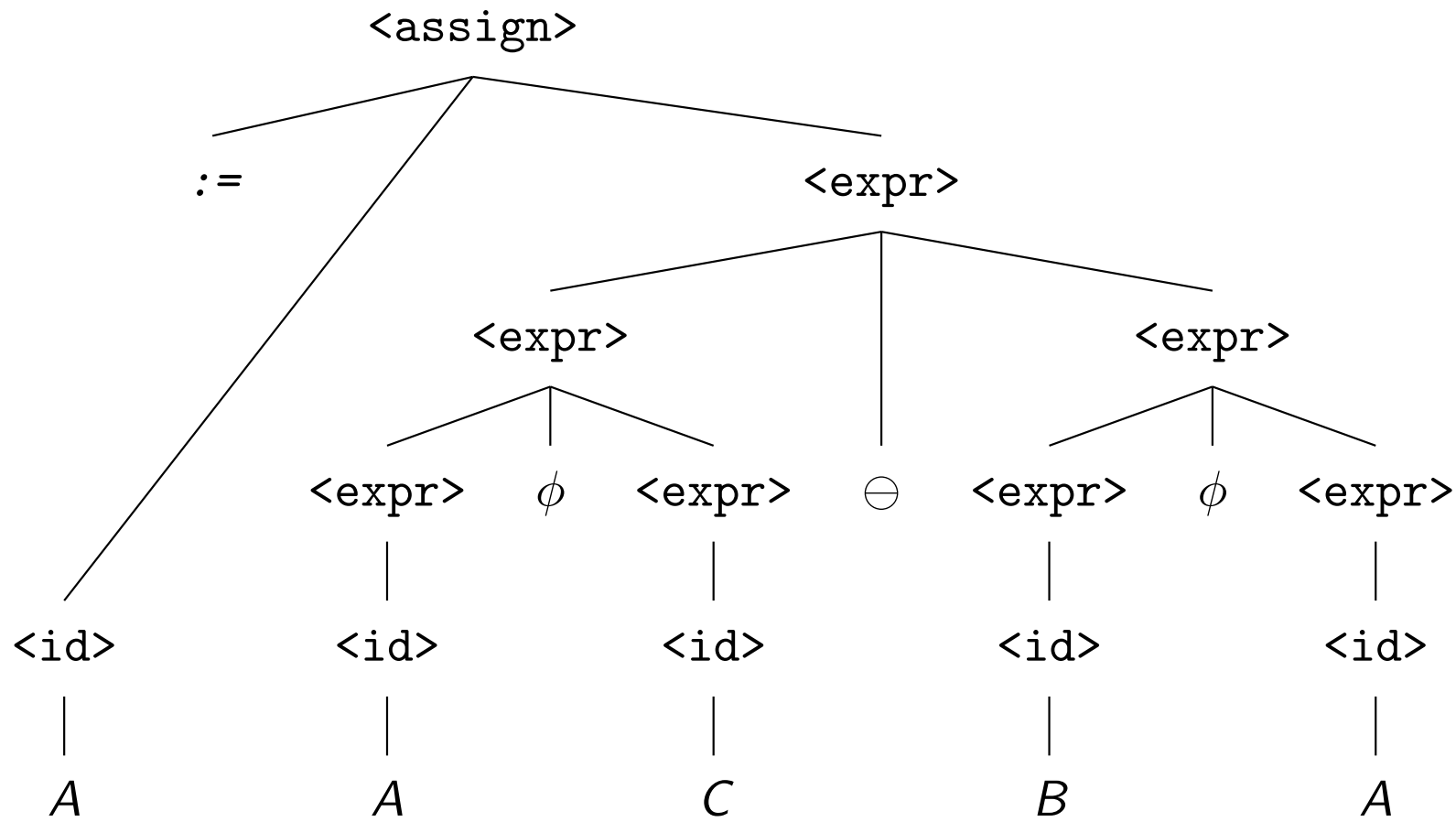
Solución

1.



Solución

2.


<assign>
:=
<expr>
<expr>
<expr>
<expr>
ϕ
<expr>
$\ominus$
<expr>
ϕ
<expr>
<id>
<id>
<id>
<id>
<id>
A
A
C
B
A

Ejercicio 3

Cree expresiones regulares que puedan identificar los siguientes patrones:

1. Fechas (ej: 03-04-1999, 07-08-1998, etc.)
2. Una dirección de correo electrónico

Ejercicio 3

Cree expresiones regulares que puedan identificar los siguientes patrones:

1. Fechas (ej: 03-04-1999, 07-08-1998, etc.)
2. Una dirección de correo electrónico

Respuestas (no son únicas):

1. `(0[1-9]|1[0-9]|2[0-9]|3[01])-(0[1-9]|1[0-2])-([0-9]+)`
2. `([a-z0-9\.\-_+])@([a-z]+)(\.[a-z]+)?(\.[a-z]+)`