Sintaxis de los Lenguajes de Programación

- Metalenguaje B.N.F.
  - Notación para especificar una gramática generativa: define el conjunto de cadenas que son programas del LP sujeto, junto con su estructura sintáctica
  - Permite describir lenguajes con una sintaxis "independiente del contexto"
- Gramáticas de atributos
  - Extensión de B.N.F. mediante atributos y reglas de evaluación de dichos atributos.
  - Permite describir lenguajes con hechos sintácticos "dependientes del contexto"

Metalenguaje B.N.F. (Forma de Backus-Naur)

- símbolos no-terminales (uno de ellos distinguido)
  - para describir los constructores sintácticos del LP sujeto
- símbolos terminales
  - para describir los símbolos (texto) del LP sujeto
- reglas de producción
  - una regla con (alternativas) para cada símb. no-terminal
  - cada alternativa: cadena de terminales y/o no-terminales

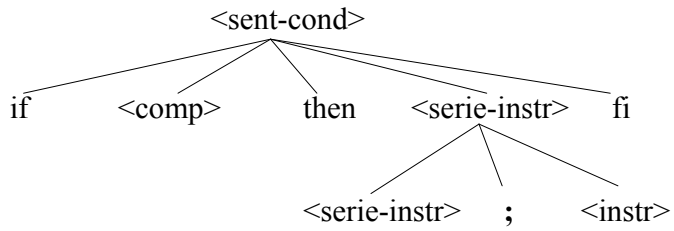
Ejemplos:

<sent-cond> ::= if <comp> then <serie-instr> fi |
if <comp> then <serie-instr> else <serie-instr> fi

<programa> ::= <serie-declar> <serie-instr>

- Se permite recursión en las reglas de producción
  - a izquierdas: <serie-instr> ::= <instr> | <serie-instr> ;<instr>
  - a derechas: <serie-instr> ::= <instr> | <instr> ; <serie-instr>
- Estructura sintáctica => árbol sintáctico

Ejemplo (parte de un árbol):



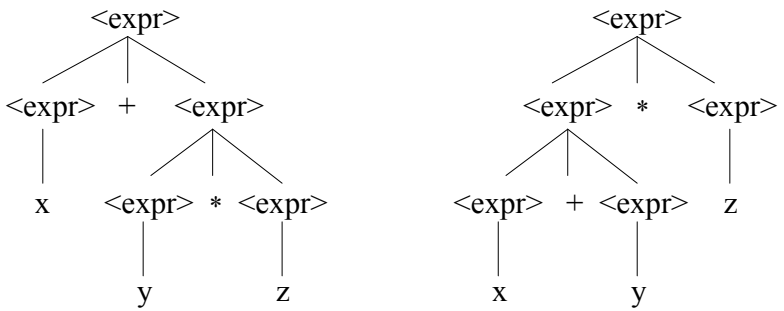
- Gramática ambigua:
  - Para una cadena terminal hay más de un árbol sintáctico

Ejemplo:

<expr> ::= x | y | z | (<expr>) | <expr> + <expr> | <expr> * <expr>

(recursión a izquierda y derecha para un mismo símbolo no-term.)

Dos árboles sintácticos para la cadena x + y * z :



- Evitar la doble recursividad:
  - introduciendo nuevos símbolos no-terminales (dejando * y + al mismo nivel)

<expr> ::= <elem> | <expr> + <elem> | <expr> * <elem>

<elem> ::= x | y | z | (<expr>)

➢ misma prioridad para * y + , y asociatividad a izquierdas

  - introduciendo nuevos símbolos no-terminales (dejando * y + a distinto nivel)

<expr> ::= <term> | <expr> + <term>

<term> ::= <elem> | <term> * <elem>

<elem> ::= x | y | z | (<expr>)

➢ prioridad de * respecto de + , y asociatividad a izquierdas para ambas operaciones.

- Características del lenguaje Pam:
  - aritmética entera
  - sentencias condicionales estructuradas y bucles
  - lectura, escritura, asignación
- Descripción sintáctica completa de Pam:
  - dada mediante la gramática BNF de la tabla 2.1.
- Asumiremos en cada programa escrito en Pam:
  - dos ocurrencias de un mismo identificador denotan la misma variable
  - una variable tendrá valor indefinido hasta que se le asigne algún valor

- Ejemplo de programa en Pam:

¿Es sintácticamente correcto?

```
read n;
to n do
  read x;
  if x>0 then
        y:=1; z:=1;
        while z<>x do
            z:=z+1;
            y:=y*z;
        end;
        write y
  fi
end
```


- Características del lenguaje Eva:
  - identificadores de distintos tipos (char, string, proc)
  - estructura de datos compuesta (string) con operaciones asociadas (head, tail, cons)
  - declaraciones
  - estructura de bloques
  - procedimientos (recursivos), paso de parámetros (por valor)
- Asumiremos en cada programa escrito en Eva:
  - se asociará un valor inicial por defecto (cadena vacía) a cada variable de tipo string en el momento de su declaración; mientras que las variables de tipo char quedan indefinidas
  - no hay conversiones de tipo (string ←→char)

– Descripción sintáctica completa de Eva:

- Gramática BNF de la tabla 2.2. +
- CONDICIONES de CONTEXTO:

1. Los <name> distintos dentro de cada <declaration sequence> en cada <block> y dentro de cada <parameter list>
2. Cada <name> contenido en una <statement> S tiene que haber sido declarado en un <block> que contenga a S o bien pertenecer a una <parameter list> de un **proc** que contenga a S.
   Si hay más de uno ➔ "el más interno"
3. En **input** <name> ➔ <name> debe ser de tipo **char**
4. En **call** <name> (<expression list>) ➔ <name> debe ser de tipo **proc** y el número y tipo de los argumentos de la llamada deben coincidir con lo declarado para <name>
5. En **cons** <char expression>,<name> ➔ <name> debe ser de tipo **string**
6. <name> debe ser de tipo **char** en <char expression> y de tipo **string** en <string expression>

EJEMPLO DE PROGRAMA en Eva

```
begin
 char c
 proc printword (string word)= (
        neq tail word,"": call printword (tail word)
        output head word )
 proc control =
  begin string w
        proc readword =   ( cons c,w
                            input c
                            neq c,space:call readword )
        call skipblaks
        call readword
        neq w,"zz" : ( call control
                       output space
                       call printword (w))
  end
 proc skipblancs =  ( input c
                      eq c, space: call skipblanks )
 call control
end
```


Variaciones de BNF

- herramientas notacionales añadidas para
  - reducir el tamaño de la gramática
  - incrementar la claridad
- no suponen mayor poder de expresión

Ejemplos:

[....] secuencia opcional

<sent-cond> ::= if <comp> then <serie-instr> [else <serie-instr>] fi

{...} repetición de 0 ó más veces

<serie-instr> ::= <instr> {; <instr> }

Gramáticas de Pam y Eva con BNF extendido ➔ tablas 2.3. y 2.4.