# Gramáticas de Atributos

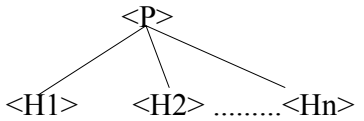
❖ Más poderosas que B.N.F.
- formalizan también los aspectos dependientes del contexto

❖ Consisten en una gramática BNF + atributos + reglas de evaluación + condiciones
- a cada símbolo BNF ← asocia → conj. de atributos
- a cada atributo ← asocia → un dominio de valores
- cada nodo (del árbol sintáctico) lleva etiquetado:
  - símbolo de la gramática BNF
  - conjunto de pares (atributo, valor)
  - y, opcionalmente, una condición lógica



❖ Reglas de evaluación (de atributos)
- para calcular el valor de los atributos en cada nodo del árbol
- cada alternativa de cada regla de producción ← asocia → un conjunto de reglas de evaluación + posible condición.

❖ Tipos de atributos
- sintetizado: el valor del atributo en un nodo del árbol depende de los valores de atributos asociados a sus "nodos hijos"
- heredado: el valor del atributo en un nodo del árbol depende de valores de atributos asociados a su "nodo padre".






❖ Reglas de evaluación asociadas
- para cada alternativa de cada regla de producción

  <P> ::= <H1> <H2> .....<Hn>

se tendrá:
- una r. evaluación para cada atributo sintetizado de <P>
- una r. evaluación para cada atributo heredado de <Hi>, para cada i=1..n
- una posible condición lógica asociada

❖ Forma de evaluación
- en los nodos del árbol sintáctico, se rellenan los valores de los atributos (en el orden adecuado) y finalmente se evalúan las condiciones. Si todas ellas son True, la cadena producida es sintácticamente correcta.



❖ Especificación de un lenguaje sujeto con G.A.
- Consiste en especificar los siguientes apartados:

1. Atributos y dominios de valores para cada atributo
2. Atributos asociados a cada símbolo de la gramática BNF junto con la clase de atributo: heredado ó sintetizado.
   - *un atributo puede estar asociado a más de un símbolo BNF y*
   - *puede ser heredado para un símbolo y sintetizado para otro.*
3. Reglas de producción BNF, junto con las reglas de evaluación de atributos asociadas a (cada alternativa de) cada regla de producción, y la condición lógica asociada (si es el caso)
4. Definición de funciones auxiliares.

## ❖ Dominios de valores para los atributos

enumeraciones con constantes escritas entre comillas.
Ejs: 'full','empty'
conjuntos con los símbolos de operación usuales: { }, ∈ , ∪
tuplas por ejemplo (17,'a') con funciones primitivas:
field$_i$ ($v_1$, ..., $v_i$ , ... $v_n$) = $v_i$
secuencias con valores del mismo tipo: <17,8,9>, < >
y con funciones primitivas:
append(s,v) (añade v al final de s),
concat($s_1$,...,$s_n$) (concatena las secuencias $s_i$),
length(s), first(s), last(s),
tail(s) y allbutlast(s).



## G. de Atributos de la especificación sintáctica de Eva

- Idea principal:

Mediante el atributo sintetizado **Decs** se va obteniendo la información del "nombre" y "tipo" de las variables que han sido declaradas en las secuencias de declaraciones y/o en las listas de parámetros (de los procedimientos). Dicha información la heredan las instrucciones mediante el atributo **Nest**.

| **Atributo** | **Valor** |
|---|---|
| Nest | secuencia de valores Decs |
| Decs | conj. de triples (Type, Tag, Params) |
| Type | 'char', 'string', 'proc' |
| Tag | secuencia de letras ('a',...,'z') |
| Params | secuencia de valores Type |



## Ejemplo 1

```
begin
      char x                    bloque interno
      proc p =  begin
                      string s
                      --- A ---- (cuerpo de p)
                end
      ----- B ------- (cuerpo del programa principal)
end
```

- **Nest** heredado para el bloque interno y para B :
  < {('char',<'x'>,<>),('proc',<'p'>,<>)}>
- **Nest** heredado para A :
  < {('char',<'x'>,<>),('proc',<'p'>,<>)},{('string',<'s'>,<>)}>



## Ejemplo 2

```
begin
      proc p (char a)=   -- S1 -- (cuerpo de p)
      char b
      ----- S2 ------- (cuerpo del programa principal)
end
```

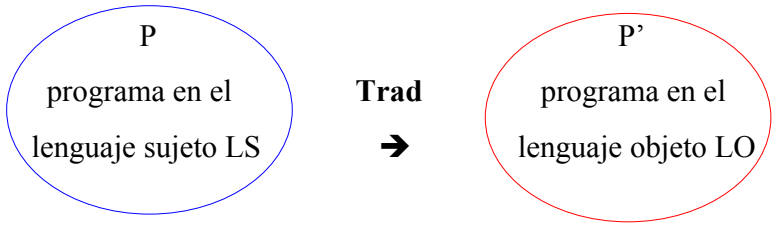
- **Nest** heredado para el bloque externo: < >
- **Nest** heredado para <declaration sequence> y para S2 :
  <{('proc',<'p'>,<'char'>),('char',<'b'>,<>)}>
- **Nest** heredado para S1 :
  <{('proc',<'p'>,<'char'>),('char',<'b'>,<>)}, {('char',<'a'>,<>)}>

- **Condiciones de contexto de Eva** - reflejadas en la tabla 2.5
  - **Condición de <namelist>:** no se duplican nombres en la misma lista de nombres (de una declaración y/o lista de parámetros)
  - **Condición de <declaration sequence>:** no hay variables con el mismo nombre en la secuencia de declaraciones
  - **Condición de <char expression>:** el tipo de la variable es 'char'
  - **Condición de <string expression>:** el tipo de la variable es 'string'
  - **Condición de la instrucción input:** el tipo de la var. leida es 'char'
  - **Condición de la instrucción cons**: el tipo de <name> es 'string'
  - **Condición de la instrucción call <name>:** el tipo de <name> es 'proc' sin parámetros
  - **Condición de la instrucción call <name>(<exp list>):** el tipo de <name> es 'proc' y los parámetros se pasan heredados a <exp list>
  - **Las dos condiciones de <exp list>** afirman que la longitud de esta lista coincide con la de la lista de parámetros y que el tipo de cada expresión coincide con el tipo del parámetro correspondiente.



## Semántica Traslacional (usando G. Atributos)

- Técnica para especificar formalmente la aplicación (o traducción) de un lenguaje (sujeto) a otro lenguaje (objeto)




- Esta aplicación constituye una especificación semántica completa del lenguaje sujeto LS en términos de la semántica del lenguaje objeto LO. Definimos:

$$\text{significado}_{LS}(P) = \text{significado}_{LO}(P'), \text{ siendo } P'=\textbf{Trad}(P)$$



## Ilustración del método traslacional

Tomamos como (parte del) Lenguaje Sujeto:

```
<programa> ::= <decl>;<instr>
<instr> ::=  <instr>;<instr>
           | while <expr> do <instr> od
           | <ident>:= <expr>
<ident>::= A | B | ... | Z
```

y como Lenguaje Objeto, el lenguaje con las siguientes instrucciones:

| | |
|---|---|
| LOAD dir | carga el contenido de **dir** en el acumulador |
| STORE dir | almacena el contenido del acumulador en **dir** |
| SALTO dir | produce un salto a la dirección **dir** |
| SALTOC dir | produce un salto condicional a la direcc. **dir** |
| NOP | operación nula |



Idea de la traducción:

- Atributo sintetizado **Trad** que define la traducción de las estructuras del L.Sujeto en términos de las del L.Objeto.
- Dos atributos para construir y consultar "la tabla de símbolos"
  **TabSimb1** sintetizado (se calcula en las declaraciones)
  **TabSimb2** heredado (se calcula en las instrucciones)
  cuya relación viene dada por las siguientes reglas de evaluación:

```
<programa> ::= <decl>;<instr>

TabSimb1(<programa>) <- TabSimb1(<decl>)
TabSimb2(<instr>) <- TabSimb1(<programa>)
```

(similares a Decs y Nest vistos para el lenguaje Eva)

| Atributo | Comentario |
|---|---|
| **Trad** | lleva la traducción (sintetizado para instr y expr) |
| **DirRes** | dir. para guardar el resultado (sintetizado para expr) |
| **Dir** | dir. del identificador (sintetizado para ident) |
| **DirInst1** | dir. donde comienza la trad. (heredado para instr y expr) |
| **DirInst2** | dir. donde termina la trad. (sintetizado para instr y expr) |

Reglas de evaluación de atributos

```
<instr> ::= <ident>:= <expr>
Trad(<instr>)     <-   Trad(<expr>)
                       LOAD DirRes(<expr>)
                       STORE Dir(<ident>)
DirInst1(<expr>) <-    DirInst1(<instr>)
DirInst2(<instr>)<-    DirInst2(<expr>)+2
```



```
<instr>1 ::=  <instr>2;<instr>3
Trad(<instr>1)      <-  Trad(<instr>2)
                        Trad(<instr>3)
DirInst1(<instr>2) <-   DirInst1(<instr>1)
DirInst1(<instr>3) <-   DirInst2(<instr>2)+1
DirInst2(<instr>1) <-   DirInst2(<instr>3)
```

```
<instr>1 ::= while <expr> do <instr>2 od
Trad(<instr>1)     <-  Trad(<expr>)
                       SALTOC DirInst1(<instr>2)
                       SALTO DirInst2(<instr>2)+2
                       Trad(<instr>2)
                       SALTO DirInst1(<expr>)
DirInst1(<expr>)  <-   DirInst1(<instr>1)
DirInst1(<instr>2) <-  DirInst2(<expr>)+3
DirInst2(<instr>1) <-  DirInst2(<instr>2)+1
```



## Ejercicio

Hallar la traducción (parcial) del siguiente texto de programa al lenguaje objeto dado:

```
A:=5;
B:=1;
while A≠0 do
     B:= B*A;
     A:= A-1
od
```

Suponiendo: hay un único acumulador, los identificadores A..Z tienen posiciones de memoria asociadas 1..26 (Dir), las posiciones 27..99 se reservan para resultados de expresiones (DirRes), el programa comienza a traducirse en la posición 100, y la traducción de cualquier expresión ocupa 10 posiciones de memoria.