Ayudantía Lenguajes de Programación

(E)BNF y REGEX

Rodolfo Castillo Mateluna Benjamín Gautier Ortiz Andrés Tapia Olguín

U. Técnica Federico Santa María
Departamento de Informática

Segundo semestre 2017

Contenidos

BNF

- Definición
- Extended BNF
- Parse Tree
- Ambigüedad

Expresiones regulares

- Python: módulo `re`

Backus-Naur Form

Definición

- Es un lenguaje de marcado (no guarda estados) o metalenguaje.
- Se compone por elementos terminales y no terminales.
- Usualmente se utiliza una regla inicial principal, representado por un S_0
- Los elementos no terminales se expresan encerrados por <>.
- Se utilizan "|" para indicar opciones.

EBNF, a.k.a., Extended Backus-Naur Form

Agrega dos metacaracteres extra = "More Fun":

- [ ] Hace un símbolo terminal o no terminal opcional, es decir, puede estar o no estar.
- { } Indica repetición de elementos, 0 o mas veces, la cantidad no se indica de manera explícita.
- ejemplo:

```
<if_stmt> ::= if <condition> then <stmts> [
    else <stmts>]
```

Lo que nos convoca: *Parse Tree*

Nos permite detectar 2 cosas fundamentales del uso de (e)BNF, ambiguedad y determinar si una expresión pertenece o no a la regla definida. Es necesario conocer la notación primeramente:

- ◯ : Terminales
- □ : No Terminales

Ejemplo

Ejemplo

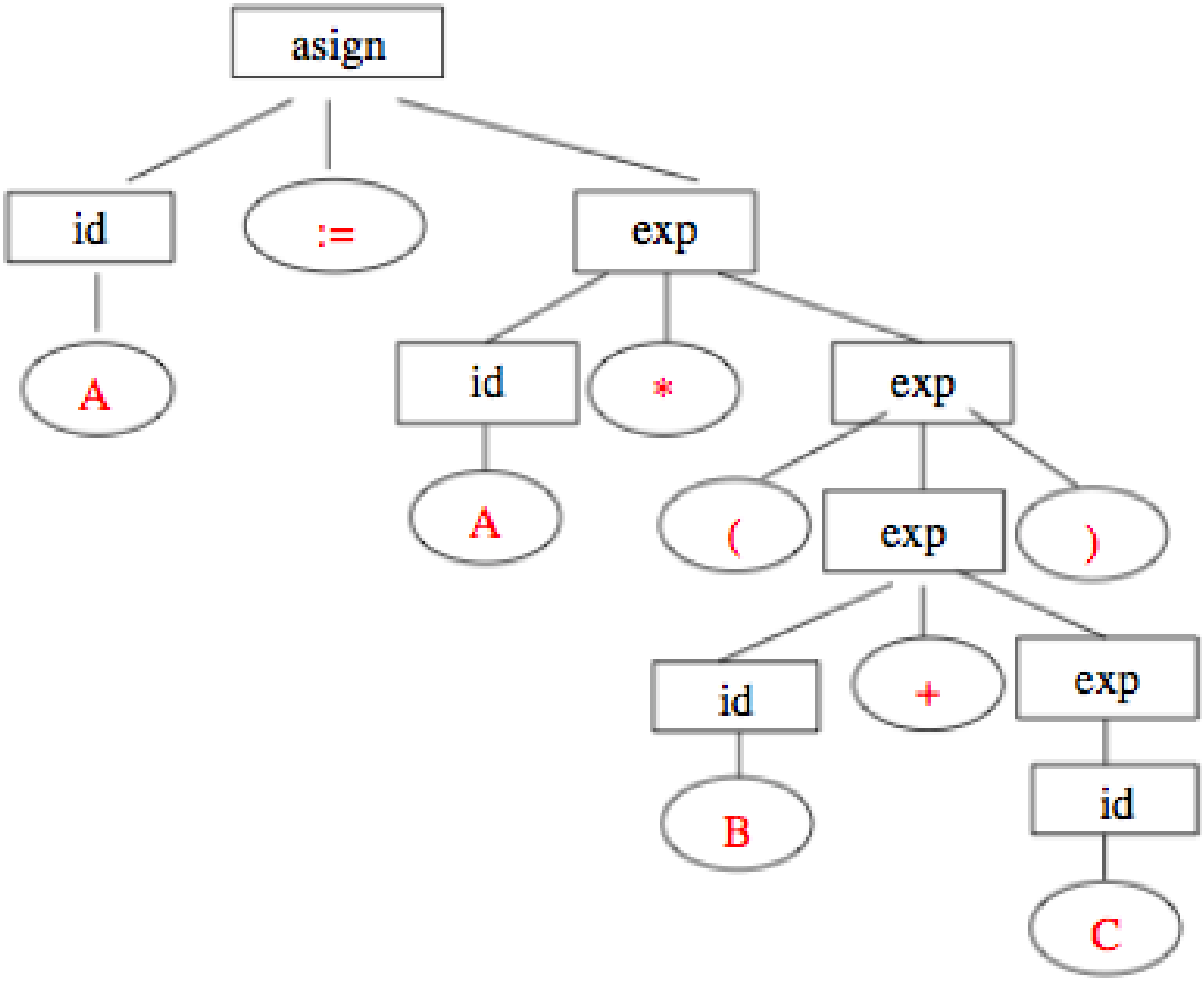
Expresión:

```
A := A * (B + C)
```

Gramática:

```
<S0>   : <id> := <exp>
<id>   : A | B | C
<exp>  : <id>+<exp> | <id>*<exp> | (<exp>)
       | <id>
```

Ejemplo

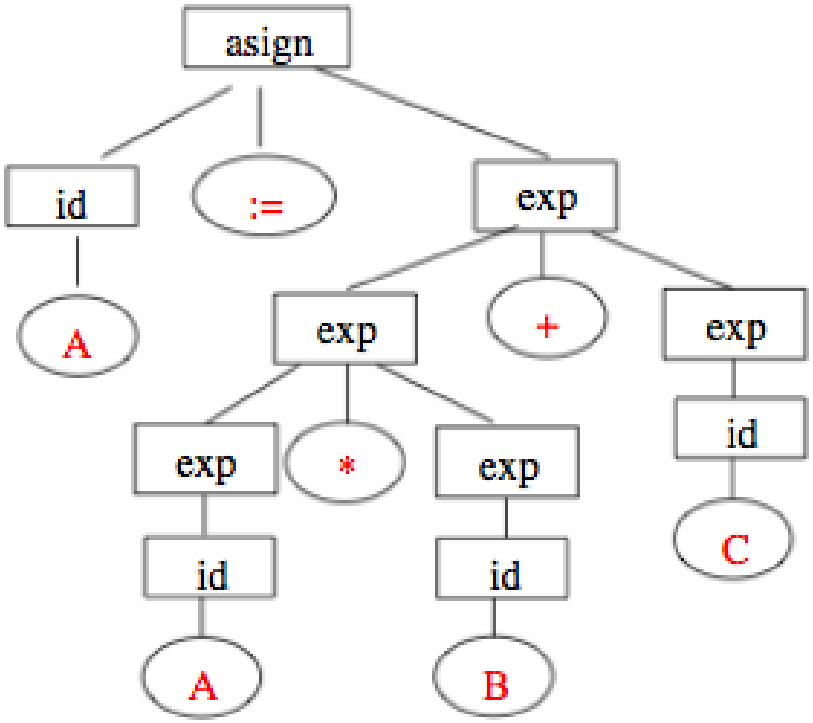


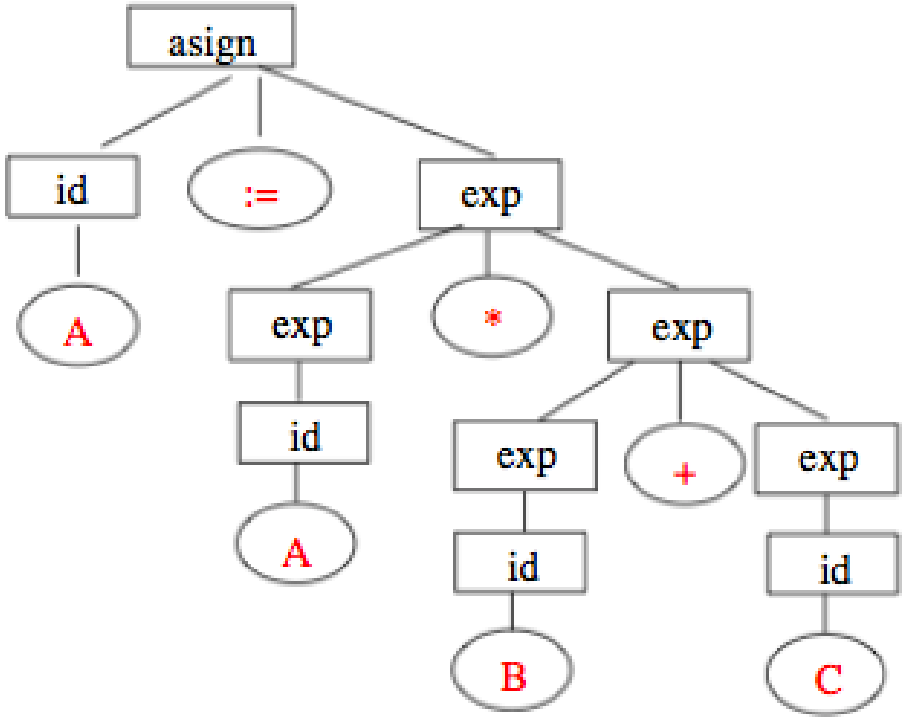
Ambigüedad

Ejemplo

```
<S0>   : <id> := <exp>
<id>   : A | B | C
<exp>  : <exp>+<exp> | <exp>*<exp> | (<exp>)
       | <id>
```

Ambigüedad


asign
id
:=
exp
A
exp
+
exp
exp
*
exp
id
id
C
id
A
B


asign
id
:=
exp
A
exp
*
exp
id
exp
+
exp
A
id
id
B
C

Expresiones regulares en python, modulo `re`

- El módulo re posee funciones para trabajar con expresiones regulares y secuencias de caracteres en python.
- Con este modulo podemos crear objetos de tipo "patrón" o objetos de tipo *matcher*. Estos últimos son los que contienen la información de la coincidencia con el patrón en alguna secuencia.
- Algunos metacaracteres de importancia son:
  - $ calza el fin de string
  - [ ] indica un conjunto, solo se elige un elemento.
  - Rangos: `[a-z]`
  - Rango negado: `[^a-z]`
  - "." coincide con cualquier caracter.
  - ? es opcional, es decir, cero o una vez.
  - + es una o mas veces.

- ...
  - * es cero o mas veces.
  - Ademas podemos crear grupos con paréntesis, o tambien podemos nombrar estos grupos:
    - r'(a)([3-5]+)'
    - (?P <letra>[a])(?P<numero>[3-5]+)
    - Lo anterior nos servirá para obtener los grupos mediante la funcion `group()`

...Uso de re en Python

- Lo primero es definir una expresion regular o un patron:

```
patron = re.compile('(a)([3-5]+)')
```

- Luego tenemos un objeto de tipo patron, usado para manejar expresiones regulares.
- Existe una serie de funciones como `match, search, findall`, que requieren de un patron compilado para ser usados sobre alguna entrada de texto. Por ejemplo:

```
string = 'a345'
objeto = patron.match(string)
objeto.group(0) -> 'a345'
objeto.group(1) -> 'a'
objeto.group(2) -> '345'
```

- En `objeto` se encuentra el matching del patron con una entrada o string.