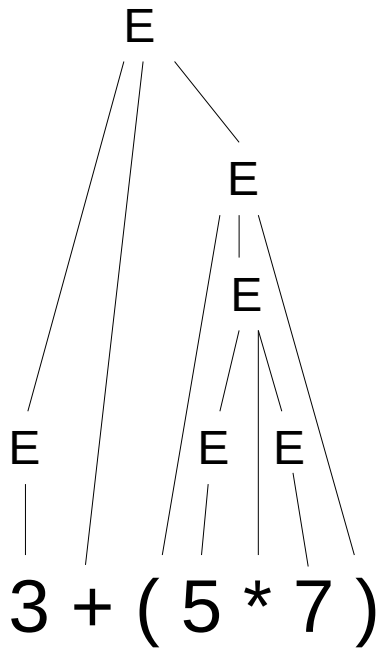
Expressões Aritméticas

Árvore de Sintaxe Concreta

```
E → E ‘+’ E
E → E ‘*’ E
E → n
E → ‘(‘ E ‘)’
```

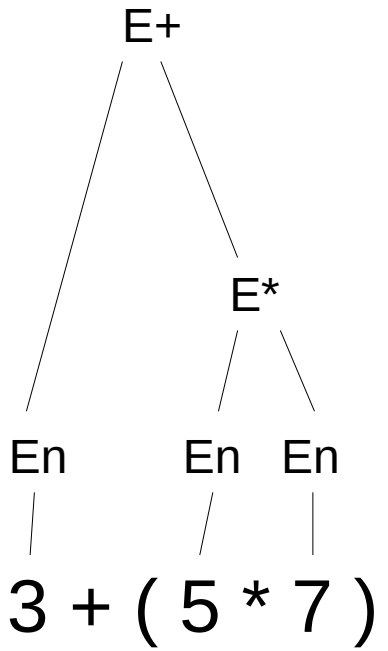


Árvore de Sintaxe Abstrata

- Colocamos rótulos nas regras
- Tiramos terminais fixos das regras
- Tiramos regras puramente sintáticas

```
E → E '+' E
E → E '*' E
E → n
E → '(' E ')'
```

```
E → E E     (E+)
E → E E     (E*)
E → n       (En)
E → '(' E ')'
```



Árvore de Sintaxe Abstrata em Haskell

```
E → E '+' E
E → E '*' E
E → n
E → '(' E ')'
```

```
E → E E     (E+)
E → E E     (E*)
E → n       (En)
E → '(' E ')'
```

```
Data E = Eadd E E
       | Emul E E
       | Enum [Char]
```

```
3+(5*7)
```

```
Eadd (Enum "3")
     (Emul (Enum "5")
           (Enum "7"))
```

Árvore de Sintaxe Abstrata em Haskell

```
data Exp = ExpN [Char]              -- constants
         | ExpAdd Exp Exp           -- e1 + e2
         | ExpSub Exp Exp           -- e1 - e2
         | ExpMul Exp Exp           -- e1 * e2
         | ExpDiv Exp Exp           -- e1 / e2
         | ExpNeg Exp               -- -e
```

Semântica

- Qual a semântica de uma expressão aritmética?
- E/ou, qual seu significado?
- O que “significa” a expressão abaixo?
  - `234 * 27 – 6276`

(`the answer to life, universe and everything?`)

Semântica

- Uma semântica possível para expressões aritméticas é seu valor!
  - Inteiro? Real? Imaginário?

```
Type Value = Integer

EvalExp :: Exp -> Value
```

```
evalExp (ExpN num) = val num 0
  where val [] acc = acc
        val (x:xs) acc = val xs (acc * 10 + dig x)
        dig x = toInteger (fromEnum x - fromEnum '0')
evalExp (ExpAdd e1 e2) = evalExp e1 + evalExp e2
evalExp (ExpSub e1 e2) = evalExp e1 – evalExp e2
evalExp (ExpMul e1 e2) = evalExp e1 * evalExp e2
evalExp (ExpDiv e1 e2) = evalExp e1 `div` evalExp e2
evalExp (ExpNeg e)     = -evalExp e
```

Composicionalidade

Em uma definição semântica, usamos apenas o *significado* de cada sub-termo. Termos com significados iguais são completamente indistinguíveis semanticamente.

```
evalExp (ExpAdd e1 e2) = evalExp e1 + evalExp e2

...
```

Observações

- Traduzimos o tipo inteiro e suas operações da nossa linguagem diretamente para seus correspondentes em Haskell.
  - E se quisermos complemento a dois?
  - O que ocorre na divisão por zero?
- Usamos recursão apenas *estrutural.*
  - Semântica “bem comportada”