# Interpretadores

# Linguagem de Expressões 2

Prof. Vander Alves

# Linguagem de Expressões 2

- LE 2 = LE 1 + identificadores (variáveis) que possuem um valor (constante)
- Durante a interpretação, surge a necessidade de um contexto:
  - mapeamento entre identificadores e valores
- Na avaliação de uma expressão, a ocorrência de um identificador é substituída pelo valor associado ao identificador
- Um programa é uma expressão

# Exemplos

- `22 * (2+3)`
- `(123 + 47 - 6) * 222 / 4`
- `let x = 1 in x+2`
- `let  x = 1 in let x = 2 in x + 1`
- `let  x = 2, y = 3 in x * y`
- `let  x = 2, y = x in (x+5) * y`
- `let  x = 2, y = x * 2 in (x+5) * y`

# Exemplos

```
let  x = 2,
     y = x * 2,
     z = let x = 1 in x*10
  in (x+5) * y + z
```

# Exemplos

```
let   x = 2,
      y = x * 2,
      z = let x = 1 in x*10,
      x = 3
  in (x+5) * y + z
```

# Gramática (Sintaxe Concreta)

```
ELet. Exp  ::= "let" [Def] "in" Exp ;
EAdd. Exp  ::= Exp  "+" Exp1 ;
ESub. Exp  ::= Exp  "-" Exp1 ;
EMul. Exp1 ::= Exp1 "*" Exp2 ;
EDiv. Exp1 ::= Exp1 "/" Exp2 ;
EInt. Exp2 ::= Integer ;
EVar. Exp2 ::= Ident ;

Def.  Def ::=  Ident "=" Exp ;

separator  Def "," ;
coercions Exp 2 ;
```

# Tipos Algébricos (Sintaxe Abstrata)

```
data Ident = Ident String

data Exp
    = ELet [Def] Exp
    | EAdd Exp Exp
    | ESub Exp Exp
    | EMul Exp Exp
    | EDiv Exp Exp
    | EInt Integer
    | EVar Ident


data Def = Def Ident Exp
```

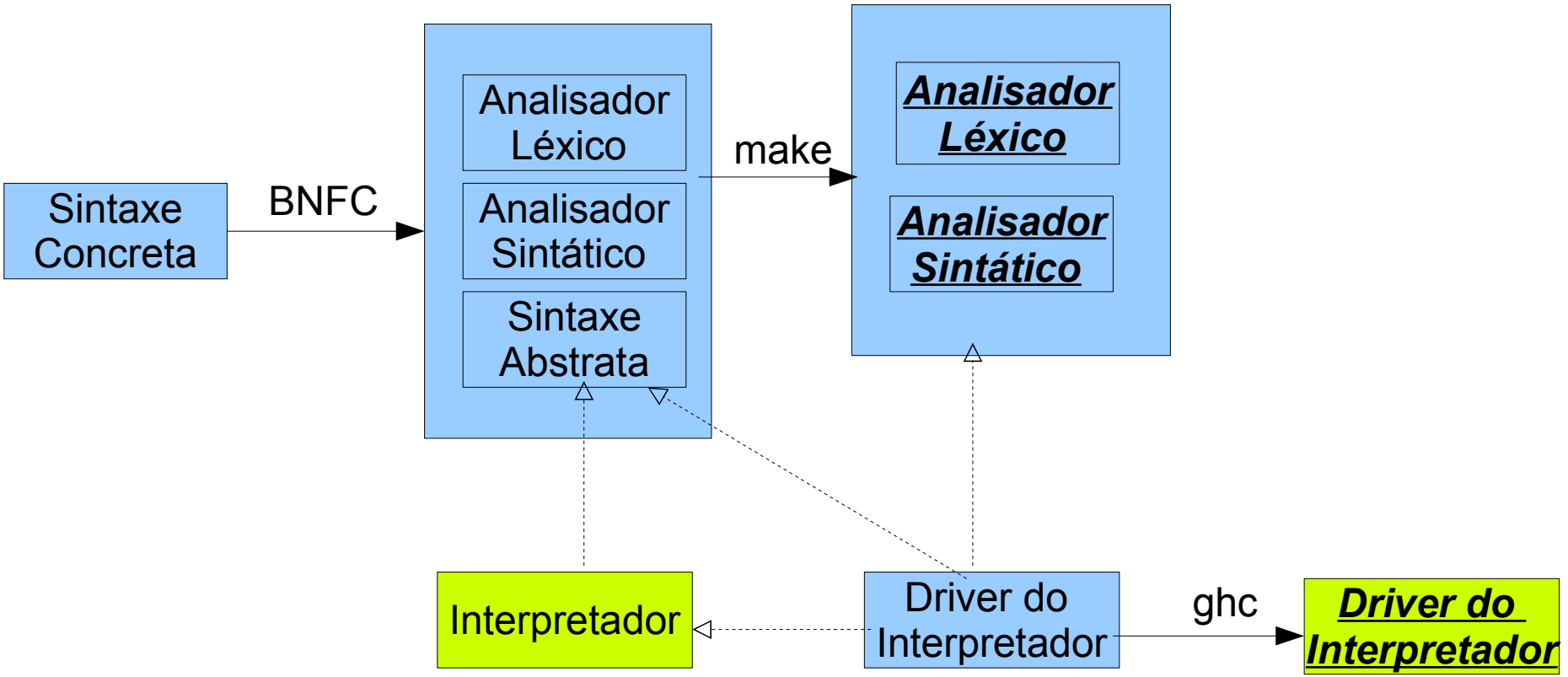
# Interpretador

```
eval :: RContext -> Exp -> Integer
eval ctx x = case x of
   ELet [] exp  -> eval ctx exp
   ELet ((Def id expr):ds) exp ->
           eval (update ctx (getStr id)(eval ctx expr))
                (ELet ds exp)
   EAdd exp0 exp -> eval ctx exp0 + eval ctx exp
   ESub exp0 exp -> eval ctx exp0 - eval ctx exp
   EMul exp0 exp -> eval ctx exp0 * eval ctx exp
   EDiv exp0 exp -> eval ctx exp0 `div` eval ctx exp
   EInt n  -> n
   EVar id  -> lookup ctx (getStr id)
```

# Interpretador

```
type RContext = [(String,Integer)]
getStr :: Ident -> String
getStr (Ident s) = s
lookup :: RContext -> String -> Integer
lookup ((i,v):cs) s
   | i == s = v
   | otherwise = lookup cs s
update :: RContext -> String -> Integer -> RContext
update [] s v = [(s,v)]
update ((i,v):cs) s nv
  | i == s = (i,nv):cs
  | otherwise = (i,v) : update cs s nv
```

# Processo


Sintaxe Concreta
BNFC
Analisador Léxico
Analisador Sintático
Sintaxe Abstrata
make
Analisador Léxico
Analisador Sintático
Interpretador
Driver do Interpretador
ghc
Driver do Interpretador

# Exercício

- Implemente contexto colateral na LE2, ou seja, a ordem das declarações das variáveis no contexto não deve afetar o resultado da execução do programa (avaliação da expressão)