Interpretadores

Linguagem Imperativa 1

Prof. Vander Alves

Linguagem Imperativa 1

- LI1 = LE 1 estendida com variáveis e comandos de atribuição e de controle de fluxo
- Durante a interpretação, surge a necessidade de um contexto:
  - Mapeamento dinâmico entre identificadores e valores
- Um programa é um comando

Exemplos

```
x = 1;
```

Exemplos

```
{
   x = 1;
   y = 2;
   z = x + y;
}
```

Exemplos

```
{
      x = 1;
      soma = 0;
      c = 10;
      while (c) {
            soma = soma + c;
            c = c - 1;
      }

}
```

Gramática (Sintaxe Concreta)

```
Prog. Program  ::= Stm ;
SAss.    Stm ::= Ident "=" Exp ";" ;
SBlock.  Stm ::= "{" [Stm] "}" ;
SWhile.  Stm ::= "while" "(" Exp ")" Stm ;
terminator Stm "" ;
EAdd. Exp1  ::= Exp1  "+" Exp2 ;
ESub. Exp1  ::= Exp1  "-" Exp2 ;
EMul. Exp2  ::= Exp2 "*" Exp3 ;
EDiv. Exp2  ::= Exp2 "/" Exp3 ;
EInt. Exp3  ::= Integer ;
EVar. Exp3  ::= Ident ;
coercions Exp 3 ;
```

Tipos Algébricos (Sintaxe Abstrata)

```
data Program = Prog Stm


data Stm = SAss Ident Exp | SBlock [Stm]
         | SWhile Exp Stm
data Exp
    = EAdd Exp Exp
    | ESub Exp Exp
    | EMul Exp Exp
    | EDiv Exp Exp
    | EInt Integer
    | EVar Ident


data Ident = Ident String
```

Interpretador

```
executeP :: RContext -> Program  -> RContext

executeP context (Prog stm) = execute context stm

execute :: RContext -> Stm -> RContext

execute context x = case x of

  SAss id exp -> update context (getStr id)
                                (eval context exp)

  SBlock [] -> context

  SBlock (s:stms) -> execute (execute context s)
                             (SBlock stms)

  SWhile exp stm -> if ( (eval context exp) /= 0)

                    then execute (execute context stm)
                                 (SWhile exp stm)

                    else context
```

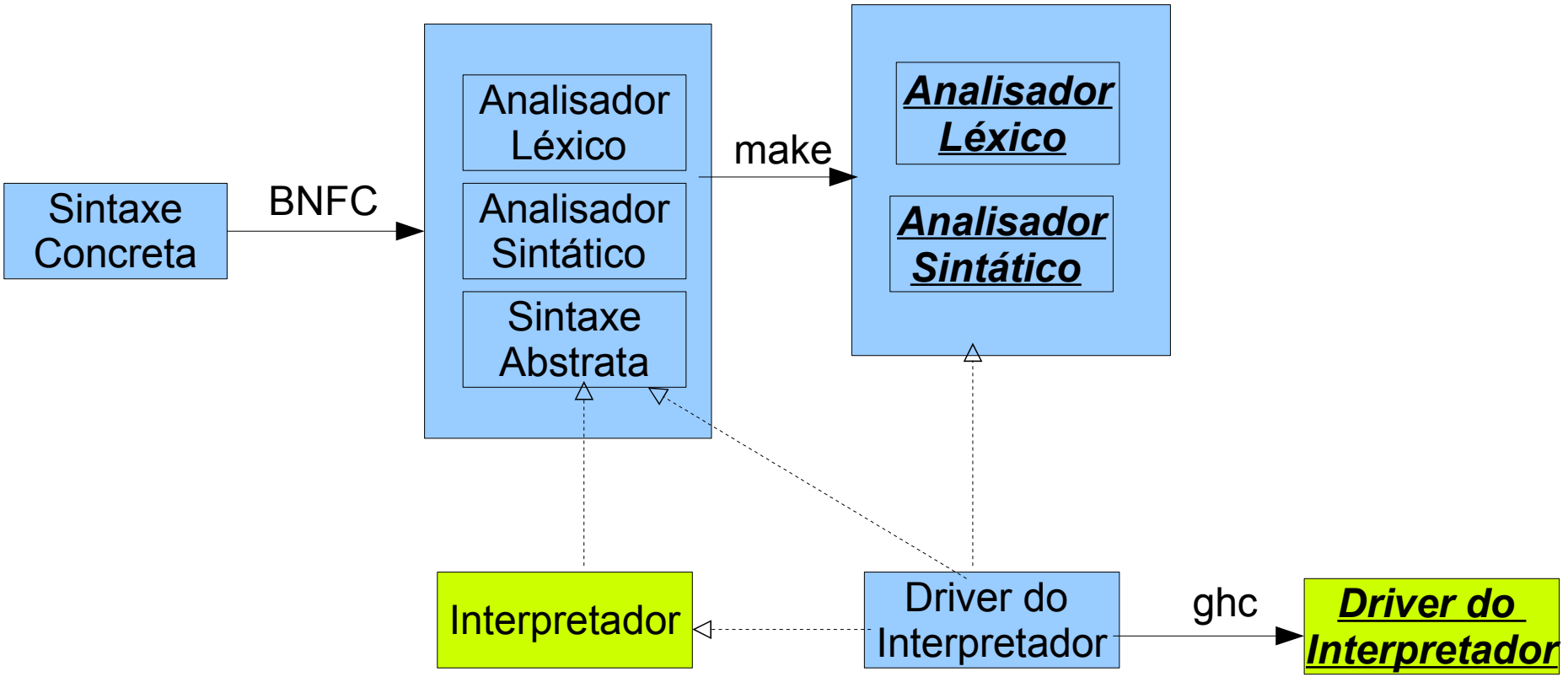
Interpretador

```
eval :: RContext -> Exp -> Integer
eval ctx x = case x of
   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

- Obtenha o valor do tipo algébrico representando cada programa apresentado como exemplo da LI1 nos slides anteriores
  - Você pode checar o resultado usando o programa **`TestLI`**, que faz parte do arquivo **`LI1.rar`**
- Estenda a definição de tipos algébricos e do interpretador para implementar os comandos ***`for`*** e ***`if`***.