Departamento de Ciência da Computação
Haskell II
Trabalho

Implemente em *Haskell* o *parser* (analisador sintático) e o inferidor de tipos (similar ao algoritmo *W*) para a linguagem definida pela gramática abaixo, sendo que *x* representa um identificador de variável que inicia com letra minúscula e *Cons* um construtor de dados que inicia com letra maiúscula. As duplas são representadas pelo construtor (,).

Represente as expressões em *Haskell* por meio do ADT *Expr* e os tipos por meio do ADT *SimpleType*:

```
<Expr >        → \x.<Expr>
                 | <Expr> <Expr>
                 | x
                 | let x = <Expr> in <Expr>
                 | case <Expr> of {<PatExpr>}
                 | if E then E else E
                 | (<Expr>, <Expr>)
                 | (<Expr>)
                 | Cons
                 | literal Inteiro
                 | literal Booleano

<Pat>          → Cons <Pats>
                 | (<Pat>, <Pat>)
                 | x

<Pats>         → <Pat>  <Pats> | ε

<PatExpr>      → <Pat> -> <Expr>

<LPat>         → <PatExpr> <MaybePat>

<MaybePat>     → ; <LPat> | ε
```

```
data Pat = PVar Id
         | PLit Literal
         | PCon Id [Pat]
         deriving (Eq, Show)

data Literal = LitInt Integer | LitBool Bool deriving (Show, Eq)

data Expr =  Var Id
           | Const Id
           | App Expr Expr
           | Lam Id Expr
           | Lit Literal
           | If Expr Expr Expr
           | Case Expr [(Pat, Expr)]
           | Let (Id, Expr) Expr
           deriving (Eq, Show)

data SimpleType  = TVar Id
                 | TArr  SimpleType SimpleType
                 | TCon String
                 | TApp SimpleType SimpleType
                 | TGen Int
                 deriving Eq
```

Os construtores para dados booleanos e duplas devem ser adicionados ao contexto inicial:

```
iniCont = ["(,)" :>: (TArr (TGen 0) (TArr (TGen 1) (TApp (TApp (TCon "(,)") (TGen 0))
          (TGen 1)))), "True" :>: (TCon "Bool"), "False" :>: (TCon "Bool")]
```