Semântica de IMP

Construção de compiladores I

Objetivos

Objetivos

- Apresentar a semântica operacional para a linguagem IMP.
- Implementar um interpretador de IMP.

Introdução

Introdução

- Na última aula, apresentamos uma semântica operacional para uma linguagem de expressões.
- Implementamos um interpretador a partir da semântica desta linguagem.

Introdução

- Porém, o que vimos não é suficiente para lidar com linguagens imperativas.
- Como lidar com a manipulação de variáveis?

Introdução

- Para lidar com variáveis, vamos adicionar uma abstração para representar a memória.

Sintaxe de Imp

Sintaxe de Imp

- Recapitulando: Sintaxe de Imp

$$
\begin{array}{lcl}
\mathit{Program} & \rightarrow & \mathit{Stmts} \\
\mathit{Stmts} & \rightarrow & \mathit{Statement}\ \mathit{Stmts} \mid \lambda
\end{array}
$$

Sintaxe de Imp

- Recapitulando: Sintaxe de Imp

$$
\begin{array}{lcl}
\mathit{Statement} & \rightarrow & \texttt{skip};\ \mid\ \mathit{Type}\ \text{id}\ \mathit{Init}; \\
& \mid & \text{id} := \mathit{Expr}; \\
& \mid & \texttt{read}\ \text{id};\ \mid\ \texttt{print}\ \mathit{Expr}; \\
& \mid & \texttt{if}\ \mathit{Expr}\ \texttt{then}\ \mathit{Block} \\
& \mid & \texttt{if}\ \mathit{Expr}\ \texttt{then}\ \mathit{Block}\ \text{else}\ \mathit{Block} \\
& \mid & \texttt{while}\ \mathit{Expr}\ \mathit{Block}
\end{array}
$$

Sintaxe de Imp

- Recapitulando: Sintaxe de Imp

$$
\begin{array}{lcl}
\mathit{Expr} & \rightarrow & \mathit{Expr}\ \mathit{Op}\ \mathit{Expr} \\
& \mid & -\ \mathit{Expr} \mid (\mathit{Expr}) \\
& \mid & !\ \mathit{Expr} \\
& \mid & \text{number} \mid \text{id} \\
& \mid & \text{true} \mid \text{false}
\end{array}
$$

Sintaxe de Imp

- Recapitulando: Sintaxe de Imp

$$
\begin{array}{lcl}
\mathit{Op} & \rightarrow & + \mid - \mid * \mid / \\
& \mid & \&\& \mid < \mid == \\
\mathit{Type} & \rightarrow & \text{int} \mid \text{bool} \\
\mathit{Block} & \rightarrow & \{\mathit{Statement}^*\} \\
\mathit{Init} & \rightarrow & := \mathit{Expr} \mid \lambda
\end{array}
$$

Sintaxe de Imp

- Representação da AST

```
newtype Var a = Var { unVar :: a }

data Value
  = EInt Int
  | EBool Bool
```

Sintaxe de Imp

- Representação da AST

```
data Exp a
  = EValue Value
  | EVar (Var a)
  | (Exp a) :+: (Exp a)
  | (Exp a) :*: (Exp a)
  | (Exp a) :-: (Exp a)
  | (Exp a) :/: (Exp a)
  | (Exp a) :==: (Exp a)
  | (Exp a) :<: (Exp a)
  | ENot (Exp a)
  | (Exp a) :&: (Exp a)
```

Sintaxe de Imp

- Representação da AST

```
data Stmt a
  = Skip
  | Def Ty (Var a) (Maybe (Exp a))
  | (Var a) := (Exp a)
  | If (Exp a) (Block a) (Block a)
  | Print (Exp a)
  | SRead (Var a)
  | While (Exp a) (Block a)
```

Sintaxe de Imp

- Representação da AST

```
type Program a = Block a

newtype Block a
  = Block { unBlock :: [Stmt a] }
```

Semântica de Imp

Semântica de Imp

- Principal diferença: inclusão de **estado**

  - Variáveis.

- Representamos o estado usando uma **função finita**.
  - Tabela de variáveis para valores.

Semântica de Imp

- Representaremos o estado por σ.
- σ(x): obter o valor de x em σ.

Semântica de Imp

- σ [x $\mapsto$ v]: incluir o valor v para a variável x.
  - Sobreescreve valores anteriores de x
  - Insere, caso x não esteja presente no domínio de σ.

Semântica de Imp

- Semântica big-step

$$\frac{v \text{ é um valor}}{\sigma; v \Downarrow v} \quad \frac{\sigma(x) = v}{\sigma; x \Downarrow v} \quad \frac{\sigma; e_1 \Downarrow v_1 \quad \sigma; e_2 \Downarrow v_2 \quad v_3 = v_1 \bullet v_2}{\sigma; e_1 \bullet e_2 \Downarrow v_3}$$

Semântica de Imp

- Semântica big-step

$$\frac{\sigma; e_1 \Downarrow v_1 \quad \sigma; e_2 \Downarrow v_2 \quad v_3 = v_1 \bullet v_2}{\sigma; e_1 \bullet e_2 \Downarrow v_3}$$

Semântica de Imp

- Definição de uma mônada

```
type Env = Map (Var String) Value

type ExecM a = ExceptT String (StateT Env IO) a
```

Semântica de Imp

- Operações sobre a mônada

```
insertEnv :: Var String -> Value -> ExecM ()
insertEnv v val = modify (insert v val)

removeEnv :: [Var String] -> ExecM ()
removeEnv vs = modify (\ env -> foldr delete env vs)
```

Semântica de Imp

```
lookupEnv :: Var String -> ExecM Value
lookupEnv v
  = do
       val <- gets (lookup v)
       case val of
         Nothing -> throwError $ "Variable undefined:" ++ (unVar v)
         Just val' -> return val'
```

Semântica de Imp

- Definição de duas funções.
  - Semântica de expressões
  - Semântica de comandos.

Semântica de Imp

- Semântica de expressões
  - Valores

```
semanticsExpr :: Exp String -> ExecM Value
semanticsExpr (EValue v) = return v
```

Semântica de Imp

- Semântica de expressões
  - Variáveis

```
semanticsExpr (EVar v) = lookupEnv v
```

Semântica de Imp

- Semântica de expressões
  - Operadores

```
(.+.) :: Value -> Value -> Value
(EInt n) .+. (EInt m) = EInt (n + m)
_ .+. _ = error "Impossible! Type Error"

semanticsExpr (e1 :+: e2)
  = do
       v1 <- semanticsExpr e1
       v2 <- semanticsExpr e2
       return $ v1 .+. v2
```

Semântica de Imp

- Semântica de comandos

```
semanticsStmt :: Stmt String -> ExecM [Var String]
semanticsStmt Skip = return []
semanticsStmt (Def ty v e)
  = do
      val <- maybe (return $ defaultValue ty) semanticsExpr e
      insertEnv v val
      return [v]
```

Semântica de Imp

- Semântica de comandos

```
semanticsStmt (v := e)
  = do
      val <- semanticsExpr e
      insertEnv v val
      return []
semanticsStmt (If e blk1 blk2)
  = do
      val <- semanticsExpr e
      if val == EBool True then semanticsBlock blk1
        else semanticsBlock blk2
      return []
```

Semântica de Imp

- Semântica de comandos

```
semanticsStmt (Print e)
  = do
      val <- semanticsExpr e
      liftIO (putStr (render (pprint val)))
      return []
semanticsStmt (SRead v)
  = do
       inp <- liftIO getLine
       case parse parserValue "" inp of
         Left _ -> throwError "invalid input!"
         Right val -> insertEnv v val >> return []
```

Semântica de Imp

- Semântica de comandos

```
semanticsStmt (While e blk)
  = do
      val <- semanticsExpr e
      if val == EBool True then do
        semanticsBlock blk
        semanticsStmt (While e blk)
      else return []
```

Semântica de Imp

- Semântica de blocos

```
semanticsBlock :: Block String -> ExecM ()
semanticsBlock (Block blk)
  = do
      vs <- concat <$> mapM semanticsStmt blk
      removeEnv vs
```

Concluindo

Concluindo

- Apresentamos a semântica operacional da linguagem Imp.

- Apresentamos como esta é a especificação de um interpretador desta linguagem.

Concluindo

- Apresentamos a implementação do interpretador em Haskell.

Exercícios

Exercícios

- O interpretador de Imp não provê suporte a valores de String. Modifique a implementação de Imp de forma a permitir Strings e também adicione a operação de concatenação de strings.