Tacit programming, návrh a implementace DSL

Obhajoba bakalářské práce

Oleg Musijenko

Tacit programming

- Funkce implicitně pracují s jejími argumenty

Ukázka 1.1.: Javascript

```
fetch("APIURL")
.then(x => fancyFunction(x))
.then(x => console.log(x))
.catch(e => console.error(e))
```

Ukázka 1.2.: Javascript

```
fetch("APIURL")
.then(fancyFunction)
.then(console.log)
.catch(console.error)
```

Currying a kompozice funkcí

- f:: (a, b) -> c po curryingu f:: a -> (b -> c) nebo-li f:: a -> b -> c
- Currying je zabudovaný v GHC
- Kompozice (.):: (b -> c) -> (a -> b) -> a -> c

```haskell
dupe:: a -> (a,a)
dupe a = (a,a)

applyToTuple :: (t1 -> a) -> (t2 -> b) -> (t1, t2) -> (a, b)
applyToTuple f g (a,b) = (f a, g b)

equalToOne :: Double -> Double
equalToOne = uncurry (+) . applyToTuple ((^ 2) . sin) ((^ 2) . cos) . dupe
```

f(x) = (sin x) ^ 2 + (cos x) ^ 2

Tacit programming

Ukázka 1.6.: Haskell

```haskell
sumCustom:: (Traversable t, Num a) => t a -> a
sumCustom = foldr (+) 0
```

Ukázka 1.8.: Haskell

```haskell
-- Haskell je lenivý jazyk a proto je možné vytvořit nekonečnou
-- fibonnacciho posloupnost a z té si vzít jen potřebný počet čísel
fibonacci:: Num a => Int -> [a]
fibonacci = (flip take) fibonacciInfinite
  where
    fibonacciInfinite:: Num a => [a]
    fibonacciInfinite = scanl (+) 0 (1:fibonacciInfinite)
```

Co je DSL

- Doménově specifický jazyk (doman specific language), který by měl řešit pouze problém domény.
- Ghetto language
- CSS a HTML, XAML, GLSL, SQL, Verilog a VHDL, Makefile, cmake file

Proč haskell? Konkurence?

Applications and libraries/Concurrency and parallelism

< Applications and libraries

Concurrent and Parallel Programming

Haskell offers a broad spectrum of tools for developing parallel or concurrent programs. For parallelism, Haskell libraries enable concise high-level parallel programs with results that are guaranteed to be deterministic, i.e., independent of the number of cores and the scheduling being used. Concurrency is supported with lightweight threads and high level abstractions such as software transactional memory for managing information shared across threads. Distributed programming is still mainly

Applications and libraries/Concurrency and parallelism - HaskellWiki

Návrh tacitního DSL

Ukázka 3.1.: Haskelyzer

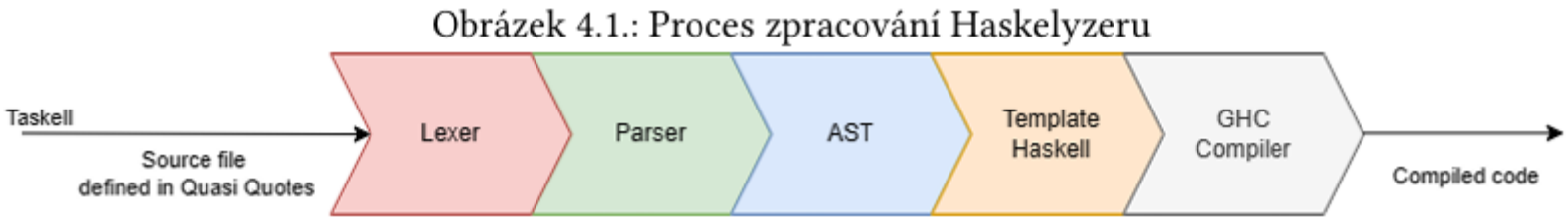
```
[CompileTime]
{
  let exampleCSV = "example.csv" :
    (a,Int)
    (b,Float)
    (c,String)
}

let concurrentProcess = exampleCSV | kalmanFilter
                                   | gaussianFilter


let nestedConcurrentProcess = exampleCSV | kalmanFilter | sum
                                                        | product
                                         | gaussianFilter
```

Návrh tacitního DSL

Ukázka 3.2.: Haskelyzer

```
let guiMainLoop = | gatherMainState -> writeEventQueue -> render
                  | gatherEventQueue -> fireEventsToMainState
```

Jak obecně implementovat DSL



Obrázek 4.1.: Proces zpracování Haskelyzeru

Zdroj: Vlastní zpracování

Lexer, parser a AST

- Parsec

```
-- file: ch16/csv6.hs
import Text.ParserCombinators.Parsec

csvFile = endBy line eol
line = sepBy cell (char ',')
cell = many (noneOf ",\n\r")

eol =   try (string "\n\r")
    <|> try (string "\r\n")
    <|> string "\n"
    <|> string "\r"

parseCSV :: String -> Either ParseError [[String]]
parseCSV input = parse csvFile "(unknown)" input
```

[Chapter 16. Using Parsec (realworldhaskell.org)]

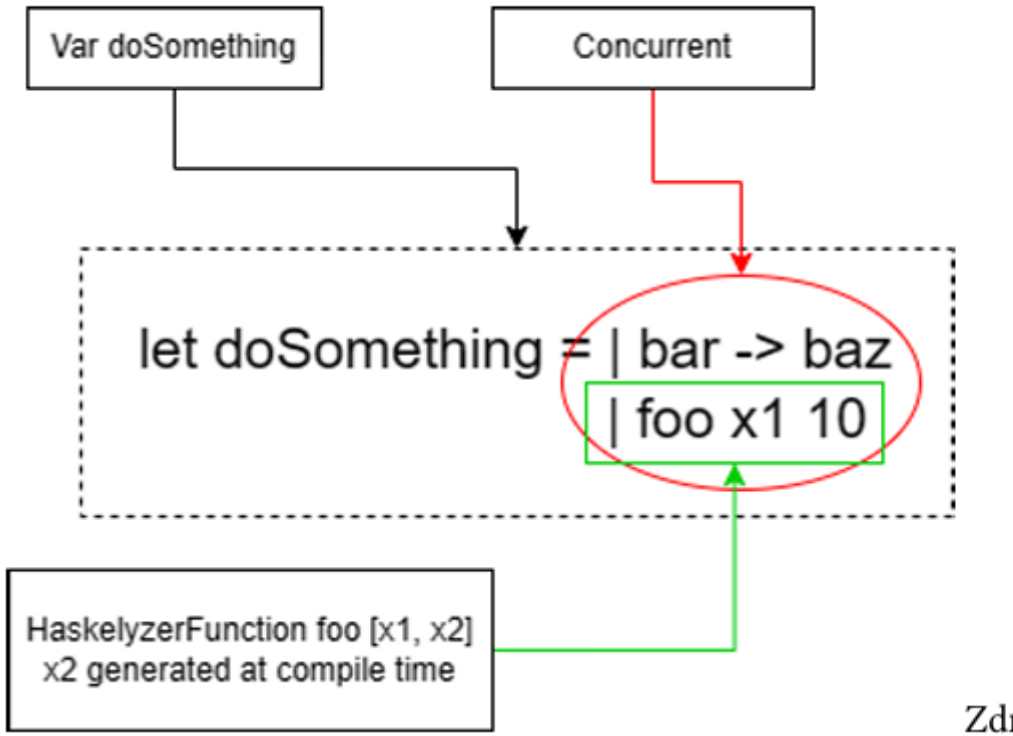
Lexer, parser a AST

- AST = [Expr]

```haskell
data Expr
  = BinOp BinOp Expr Expr
  | CsvDataType CsvDataType
  | UnaryOp UnaryOp Expr
  | Var Name [HaskelyzerFunction]
  | SchemaExpr Schema
  | LiteralExpr Literal
  deriving (Eq, Ord, Show)

data HaskelyzerFunction =
  HaskelyzerFunction Name [Name] -- Function name args
  | Concurrent [[HaskelyzerFunction]]
  deriving (Show, Ord, Eq)
```

Lexer, parser a AST

Obrázek 4.2.: Lexer v kontextu jazyka



Zdroj: vlastní zpracování

Lexer, parser a AST

- TDD – Test Driven Development
- Bez testů nevznikne žádný spolehlivý parser

Z AST do LLVM

- Jedná se o front end kompilátorů – clang, Jai, Odin
- Není vhodné na DSL
- Na programovací jazyk ano
- U DSL generování bindings mezi jazyky? Moc náročné.

Z AST do Template Haskell

- Template Haskell = Template metaprogramming
- Generování funkcí při kompilaci, protože je umožněn přístup k samotným tokenům jazyka

Ukázka 4.5.: Haskelyzer

```
let loadModels =
| loadModel1 -> rotate 0.0 45.0 90.0 -> scale 4.0
| loadModel2 -> translate 60.0 0.0 0.0 -> scale 3.0
```

Ukázka 4.6.: Haskell

```
loadModels:: IO [Model]
loadModels = runListConcurrently [_loadModel1, _loadModel2]
    where
      _loadModel1 =
        ((scale 4.0) . (rotate 0.0 45.0 90.0)) loadModel1
      _loadModel2 =
        ((scale 3.0) . (rotate 0.0 45.0 90.0)) loadModel2
```

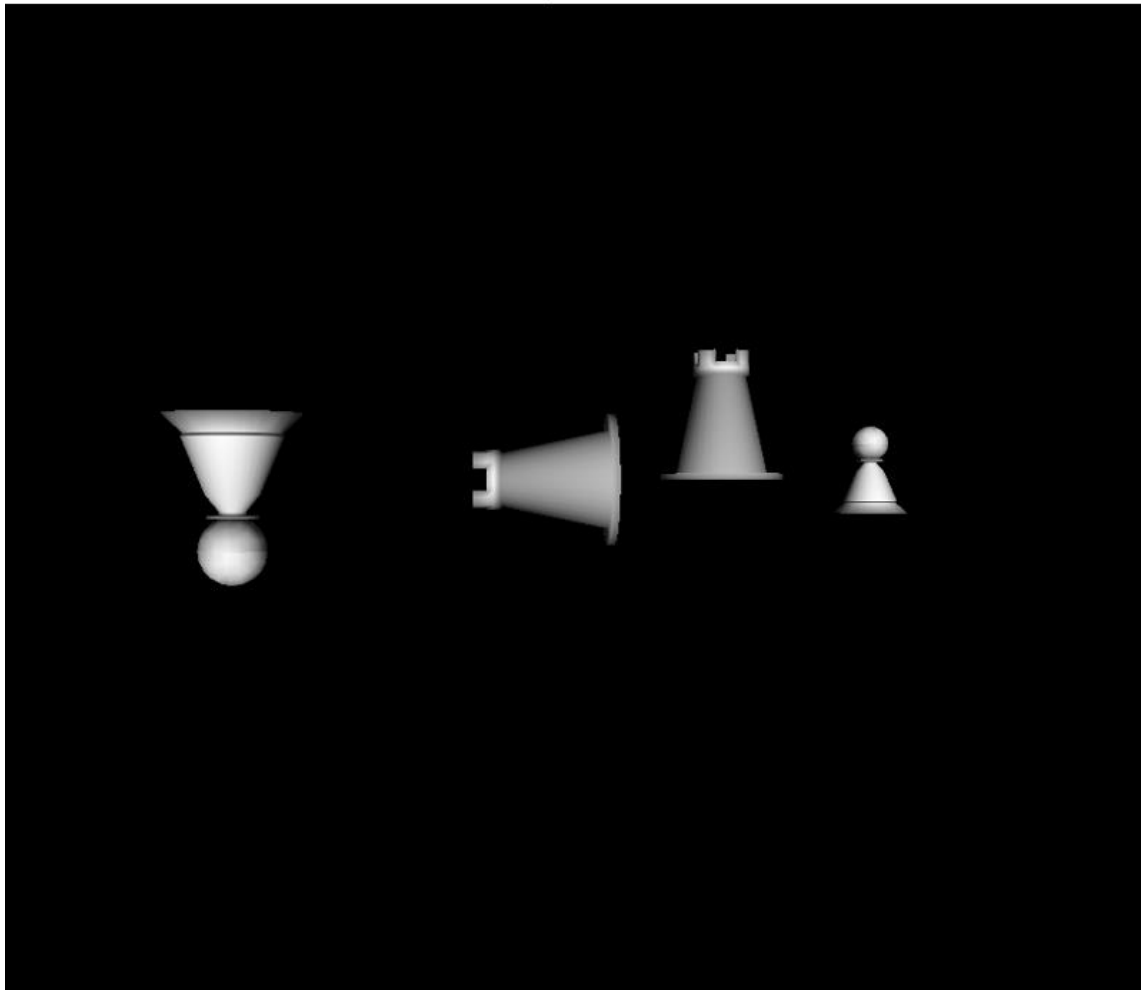
Příklady užití – 3D

- Zpracování scén v 3D editoru

Ukázka 5.5.: Haskell

```
let loadModels =
| loadRook -> scale 10 -> translate 20 0 20
| loadRook -> scale 10 -> rotate (-90) 0 0-> translate 40 0 0
| loadPawn -> scale 10 -> translate 0 0 0
| loadPawn -> scale 20 -> rotate 180 0 0-> translate 100 0 0
```

Obrázek 5.1.: Výstřižek z rendereru



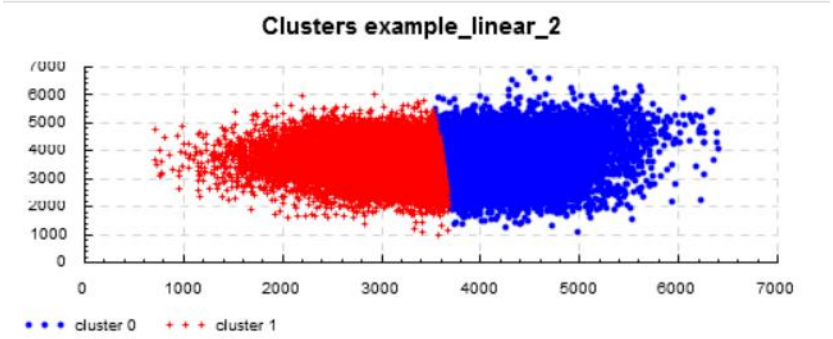
Příklady užití – web scraping

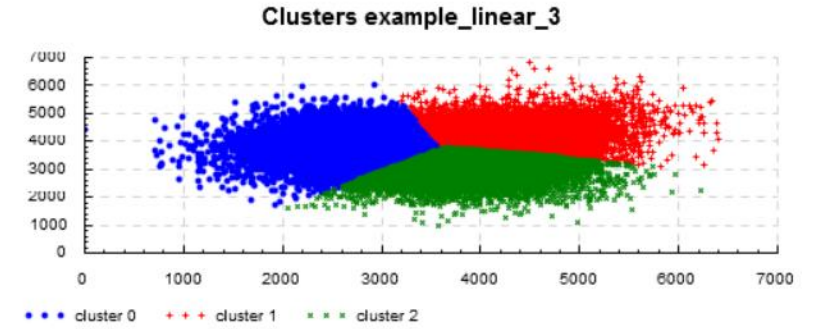
```
sourcesFromWhichToMine =
  [
    ("urlToNewsSource1",
     "urlToDifferentNewsSource1",
     "urlToWeather1"
    ),
    ("urlToNewsSource2",
     "urlToDifferentNewsSource2",
     "urlToWeather2"
    ),
    ("urlToNewsSource3",
     "urlToDifferentNewsSource3",
     "urlToWeather3"
    ),
    ("urlToNewsSource4",
     "urlToDifferentNewsSource4",
     "urlToWeather4"
    ),
  ]

-- Predefine mining with algorithms here,
-- using scalpel is strongly advised
-- also define saveToPostgreSQL and saveToMongoDB
createDBConnections:: IO(MongoDBConnection,PostgreConnection)
createDBConnections = do
  x <- createMongoConnection
  y <- createPostgreConnection
  return (x, y)
```

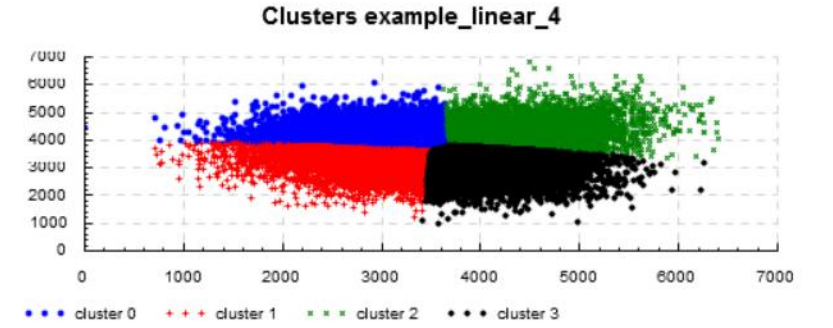
Ukázka 5.7.: Haskelyzer

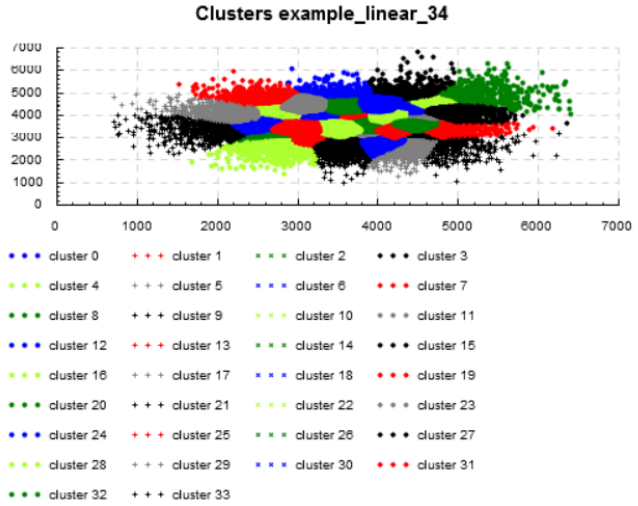
```
let scrapeData a b c postgreConnection mongoConnection =
  | scrapeNewsSource a -> saveToPostgreSQL postgreConnection
  | scrapeDifferentNewsSource b -> saveToMongoDB mongoConnection
  | scrapeDifferentNewsSource b -> saveToPostgreSQL mongoConnection
  | scrapeWeatherSource c -> saveToPostgreSQL postgreConnection
```

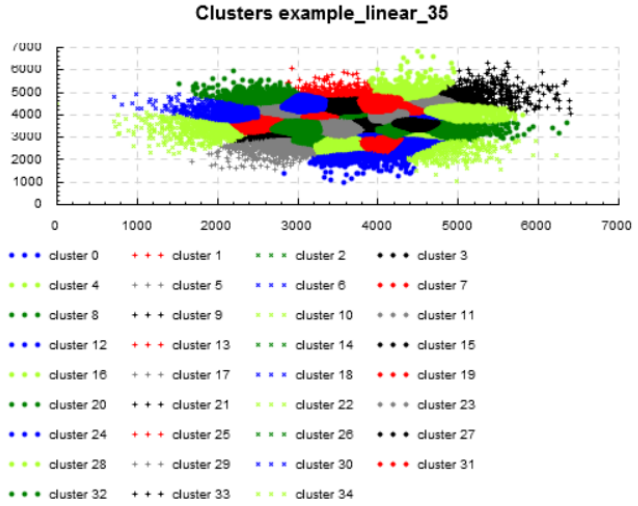
Příklady užití – clustering





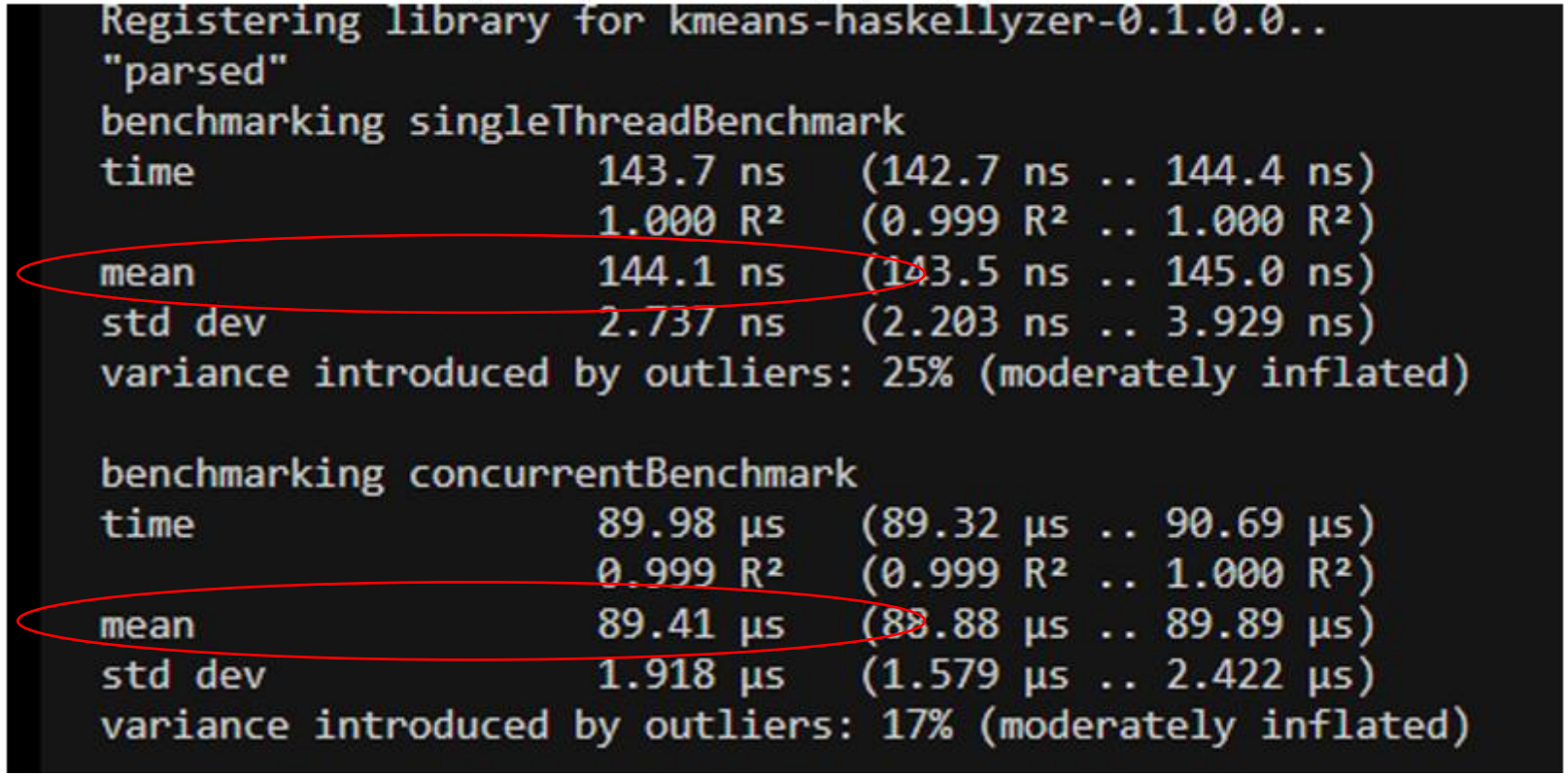






Výsledek?

Obrázek 5.2.: Výsledky benchmarkingu, kde je jedno vlákno rychlejší



Vylepšení do budoucna

- Tvorba threadů na začátku programu a udržet je na živu
- Haskelyzer momentálně nepodporuje základní konstanty
- Caching a duplikace stejných funkcí

Děkuji za pozornost