Реалізація лямбда-числення в Haskell

Implementation of lambda calculus in Haskell.

Студент 2-го курсу МП Комп'ютерних Наук Соболєв Владислав

Науковий керівник: Проценко Володимир Семенович

Основні елементи λ-числення:

Змінні
(x, y)

Лямбда-абстракції
(λx.expr – визначення функції)

Застосування
(expr1 expr2 – виклик функції)

Правила редукції:

α-редукція: Перейменування зв'язаних змінних для уникнення конфліктів.

β-редукція: Механізм застосування функції шляхом підстановки аргументу в її тіло.

η-редукція: Спрощення функцій вигляду λx.(f x) до f.

Стратегії обчислення

Нормальний порядок: Завжди редукується найлівіший, найбільш зовнішній редекс.

- **Переваги:** Гарантовано знаходить нормальну форму, якщо вона існує; дозволяє працювати з потенційно нескінченними виразами.

Сильне обчислення: Редукує вираз повністю, до його нормальної форми, включаючи внутрішні редекси.

- **Переваги:** Надає повністю спрощений результат, ідеально підходить для покрокової візуалізації.

Кодування Черча: Дані як Функції

Логічні значення:

true ≡ λxy.x, false ≡ λxy.y

Натуральні числа:

zero ≡ λsz.z, n ≡ λsz.s^n(z)

Пари Черча:

PAIR ≡ λxyf.fxy

Haskell як мова реалізації

haskell/**parsec**

A monadic parser combinator library

- Чиста функціональність
- Статична типізація та типобезпека
- Алгебраїчні типи даних та зіставлення зі зразком (Pattern Matching)
- Підтримка рекурсії та підстановки
- Бібліотека Parsec

Проектування та Реалізація

```haskell
-- Evaluate expression with a maximum step limit (returns only computed steps)
evalStepsLimited :: Int -> Expr -> [Expr]
evalStepsLimited maxSteps = go 0 []
  where
    go n acc e
      | n >= maxSteps = acc ++ [e]  -- return partial result up to maxSteps
      | otherwise = case step e of
          Just e' -> go (n + 1) (acc ++ [e]) e'
          Nothing -> acc ++ [e]     -- normal f
```

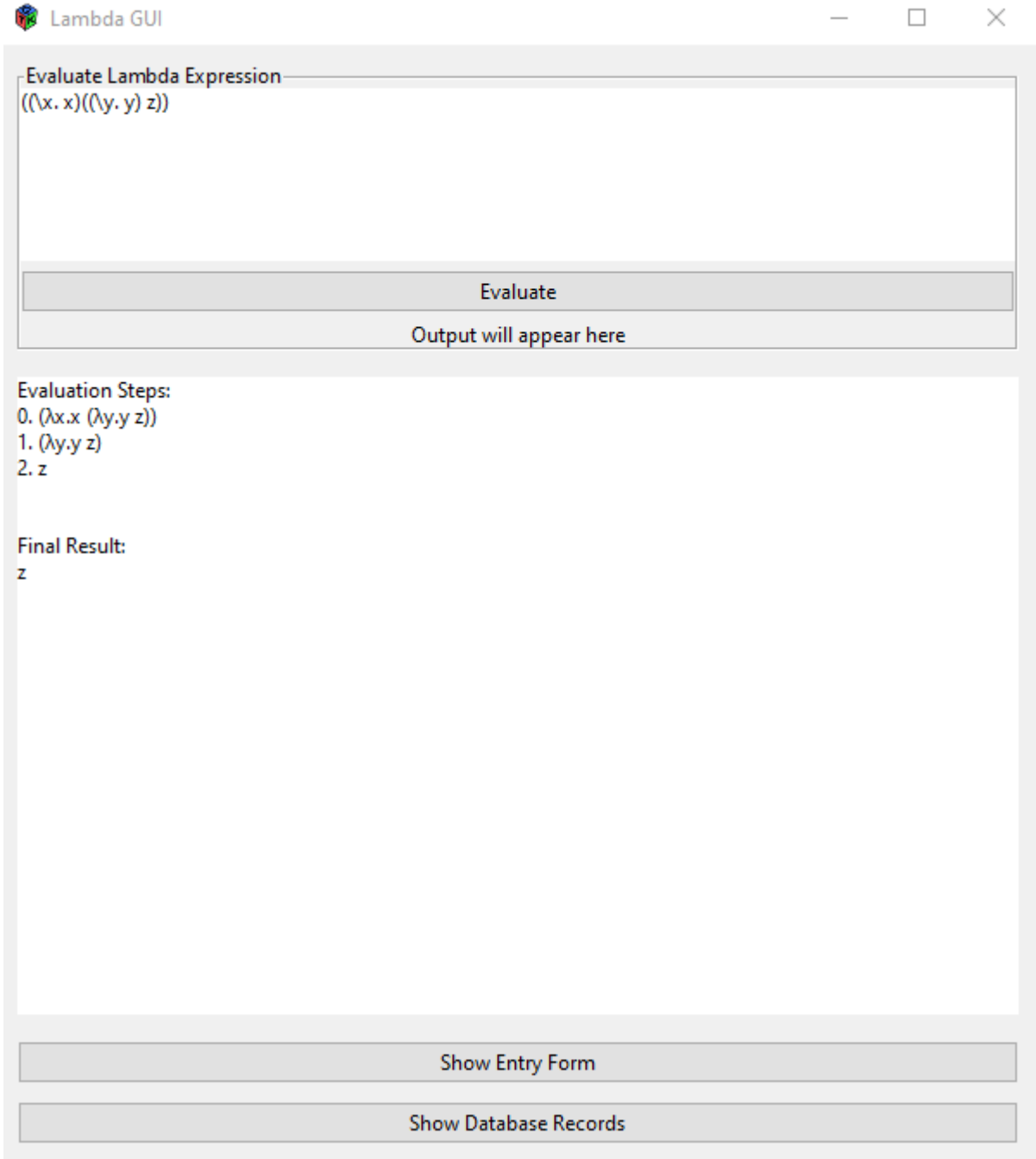
```haskell
-- Lambda calculus expression AST
data Expr
  = Var String
  | Lam String Expr
  | App Expr Expr
  deriving (Eq, Show)
```

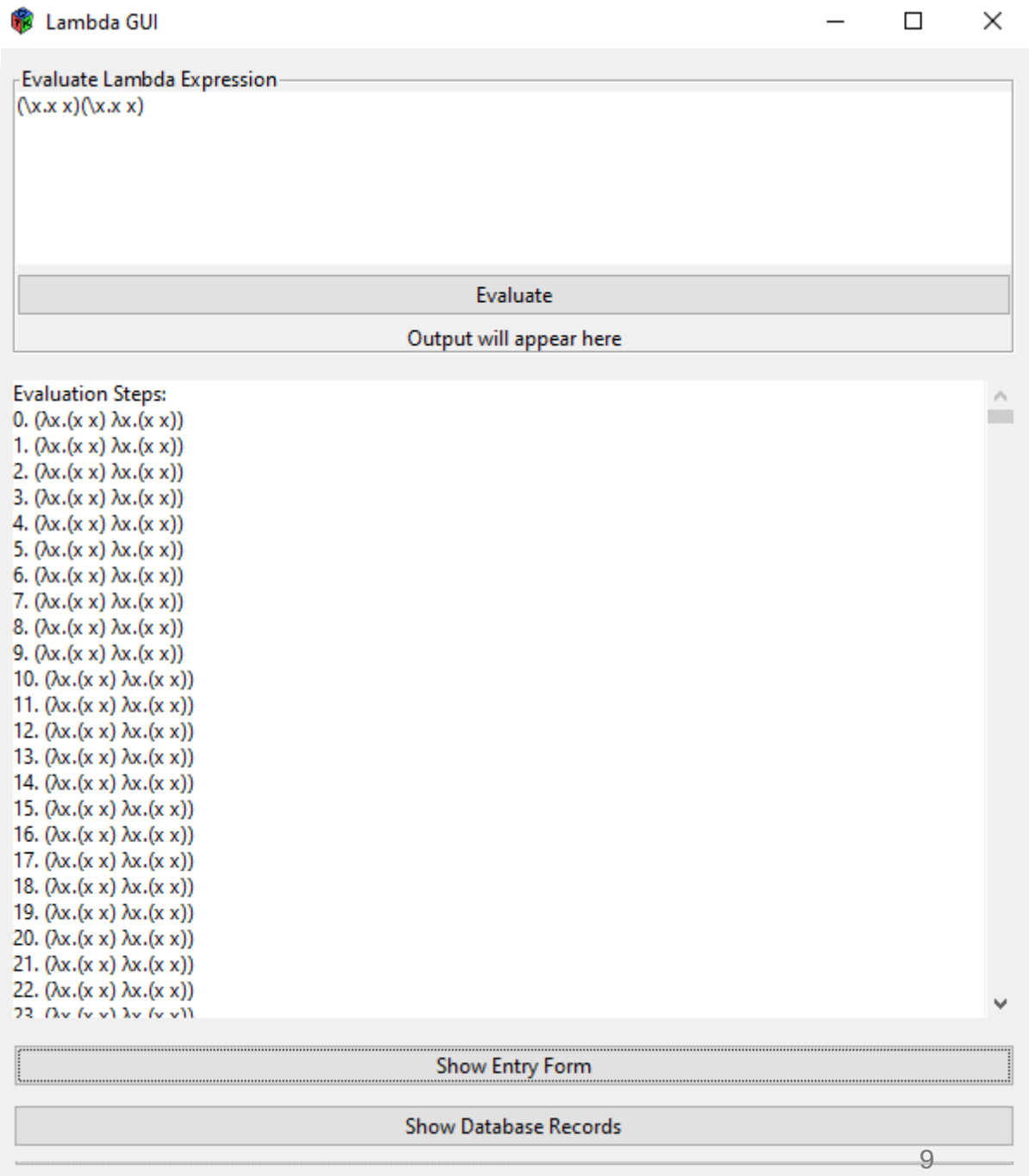
```haskell
-- Evaluate a program with an environment
evalProgram :: Env -> [Statement] -> Eith
evalProgram env stmts =
  case splitAtAssignments stmts of
    (defs, ExprOnly e) ->
      let newEnv = foldl insertDef env defs
          resolved = resolveEnv newEnv e
          steps = evalStepsLimited 1000 resolved
      in Right (steps, last steps)
    _ -> Left "No expression to evaluate."
  where
    insertDef acc (Assign name expr) = Map.insert name expr acc
    insertDef acc _ = acc

    splitAtAssignments xs = (init xs, last xs)
resolveEnv env (App e1 e2) = App (resolveEnv env e1) (resolveEnv env e2)
resolveEnv env (Lam x e) = Lam x (resolveEnv env e)

-- Extract name-body map from DB records
buildEnvFromRecords :: [Record] -> Map String Expr
buildEnvFromRecords records = Map.fromList [ (name, parseUnsafe body)
                                           | Record _ _ _ name body _ <- records ]
```

Функціональність та Архітектурні Рішення

Підтримка зовнішніх середовищ (SQLite)

Інтеграція з графічним інтерфейсом (GTK+)


Lambda GUI
Evaluate Lambda Expression
((\x. x)((\y. y) z))
Evaluate
Output will appear here
Evaluation Steps:
0. (λx.x (λy.y z))
1. (λy.y z)
2. z
Final Result:
z
Show Entry Form
Show Database Records


Lambda GUI
Evaluate Lambda Expression
(\x.x x)(\x.x x)
Evaluate
Output will appear here
Evaluation Steps:
0. (λx.(x x) λx.(x x))
1. (λx.(x x) λx.(x x))
2. (λx.(x x) λx.(x x))
3. (λx.(x x) λx.(x x))
4. (λx.(x x) λx.(x x))
5. (λx.(x x) λx.(x x))
6. (λx.(x x) λx.(x x))
7. (λx.(x x) λx.(x x))
8. (λx.(x x) λx.(x x))
9. (λx.(x x) λx.(x x))
10. (λx.(x x) λx.(x x))
11. (λx.(x x) λx.(x x))
12. (λx.(x x) λx.(x x))
13. (λx.(x x) λx.(x x))
14. (λx.(x x) λx.(x x))
15. (λx.(x x) λx.(x x))
16. (λx.(x x) λx.(x x))
17. (λx.(x x) λx.(x x))
18. (λx.(x x) λx.(x x))
19. (λx.(x x) λx.(x x))
20. (λx.(x x) λx.(x x))
21. (λx.(x x) λx.(x x))
22. (λx.(x x) λx.(x x))
Show Entry Form
Show Database Records

Порівняння з Аналогічними Роботами

Більшість існуючих лямбда-інтерпретаторів на Haskell є консольними

Наявний повноцінний GUI та інтеграція з SQLite

Акцент на фундаментальних механізмах обчислення

Висновки

Ключові досягнення:

Повноцінний інтерактивний GUI	**Детальна покрокова візуалізація β-редукції**	**Система управління користувацькими визначеннями**

Розроблений інтерактивний інтерпретатор нетипізованого лямбда-числення на Haskell є не просто функціональним обчислювачем, а **освітнім інструментом,** що дозволяє користувачам зрозуміти фундаментальні концепції обчислень та функціонального програмування.

Дякую за увагу