Title: **Haskelite: A Step-By-Step Interpreter for Teaching Functional Programming**

Vasconcelos, P. (2023). Haskelite: A Step-By-Step Interpreter for Teaching Functional Programming. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Authors: Pedro Vasconcelos

Published: 4th International Computer Programming Education Conference (ICPEC 2023)

Introduction

- Functional programming has been taught in universities since the 1980s. Textbooks and interest in functional languages show its importance.
- The ACM CS curriculum increased the functional programming component in 2022.
- However, educators report difficulties students face learning functional programming:
    - Understanding the evaluation model
    - Type error messages
    - Perception functional languages are only academic
- This paper describes Haskelite, a step-by-step interpreter for a subset of Haskell.
- It is designed to help students understand:
    - Evaluation by rewriting
    - Pattern matching and recursion
    - Higher order functions
    - On-demand evaluation
- Haskelite focuses on these core concepts using a subset of Haskell.

- It is implemented in Elm and JavaScript to run client-side in the browser without installation.
- Source code and demo available online.

Design Goals and Motivation

- Bird and Wadler describe functional programming as building definitions and using the computer to evaluate expressions.
- The evaluation model should be familiar from high school algebra, but students struggle:
    - Complexity going from numbers to algebraic data types
    - Missing cases or redundant cases when pattern matching
    - Thinking recursively and inductively
    - Reasoning about termination and lazy evaluation
- Haskelite aims to automate evaluation steps to expedite learning.
- It allows quickly trying different expressions and definitions.
- Showing evaluation steps connects computations to code.
- This prepares students for equational reasoning and proofs.

User Interface

- Haskelite runs embedded in a web page.
- The editor provides feedback on errors.
- Instructor can provide pre-filled code.
- Clicking "evaluate" switches to evaluation mode.
- Each step shows evaluation of an expression.
- Steps apply student functions or built-in functions.
- Previous steps are shown.
- Tooltips explain the justification for each step.
- Student can move backwards and forwards through steps.

Examples

- Sum function over lists - illustrates structural recursion
- Product function over lists - illustrates base case issues
- Intersperse function - illustrates complex recursion patterns

- Mapping over infinite lists - illustrates higher order functions and lazy evaluation

Implementation

- Implemented in Elm, compiled to JavaScript.
- Parsing uses parser combinators.
- Type checking uses Hindley-Milner.
- Evaluation uses naive rewriting for simplicity.
- Performance is adequate - 70KB JS bundle, fast enough for small examples.
- Runs client-side so scales to many users.

Limitations

- Limited Haskell subset, no user-defined types, list comprehensions etc.
- Call-by-name evaluation rather than call-by-need.
- No persistence of programs across sessions.

Related Work

- Helium - Haskell teaching compiler/interpreter with good error messages. No longer maintained.
- GHCi - advanced Haskell debugger for experienced programmers.
- Python Tutor - visualizes execution of imperative programs.
- DrRacket - IDE for Scheme with debugger.
- Lambda Lessons - inspiration for Haskelite, reimplemented in Elm.

Experience and Future Work

- Used since 2022, positive experience but not empirically validated.
- Students keen to try examples and clarify understanding.
- Future work:
    - Expand language features
    - Improve UI, especially for mobile

    - Add persistent state
    - Empirical evaluation

Conclusion

- Haskelite aims to help students learn core concepts of functional programming by visualizing evaluation steps.
- Implemented in Elm/JavaScript for web delivery with no installation required.
- Initial classroom experience is positive but needs formal evaluation.
- Many possibilities exist for extending the prototype.