Web ブラウザ上で動作する拡張可能な Haskell 処理系

情報科学科　山田 航　　指導教員：大久保 弘崇

1　はじめに

純粋関数型プログラミング言語 Haskell を用いたプログラミングを行うには，基本的には Haskell の代表的なコンパイラである GHC[1] をインストールする必要がある．しかし，インストール方法は OS によって異なり，プログラミングの経験がない初学者にとって容易なことではない．インストールの手間を省き，手軽に Haskell を試すことができる環境はプログラミング初学者にとって有用である．

また，Haskell には型クラスや遅延評価などの複雑な特徴があり，プログラムの理解の妨げになっている．式の評価途中のトレースなどの支援機能があると，初心者が Haskell を学習する際に大いに役立つと考えられる．しかし，GHC に対して機能を追加することは困難であり，初学者への Haskell プログラミングを支援するための機能等を実装するためには，容易に拡張可能な Haskell の処理系が必要である．

本研究は，プログラミング初心者でも手軽に Haskell を試すことが可能であり，拡張可能な研究基盤となる処理系を実装することを目的とする．また，実装には内山による先行研究 [2] にて実装された Haskell のサブセット言語の処理系 Hachu を基に拡張をする形で開発する．

2　Hachu

Hachu は TypeScript [6] で実装された Haskell の サブセット言語の処理系である．デザインパターンとして Visitor パターンが用いられており，処理系はパーサ，構文木を表すデータ構造，評価や表示を行う Visitor から構成されている．パーサがプログラムを構文木に変換し，Visitor が構文木を処理する．構文は ラムダ式，値の定義，データ型の定義，case 式などが実装されている．また，論文内に Hachu に遅延評価を追加する手法が提案されていたが，Hachu には含まれていなかったため，遅延評価を行うには検証が必要である．

3　処理系の拡張

3.1　構文の追加

本研究は Hachu の拡張により Haskell98 [5] と同等の構文に対応することを目標に実装を進めた．図 1 に Haskell の代表的な機能の実装状況を示す．

表 1　対応している機能

機能	Hachu	拡張後の Hachu
インデント構文	未実装	未実装
型検査器	未実装	未実装
ラムダ式	実装済み	実装済み
case 式	実装済み	実装済み
レコード構文	未実装	未実装
関数定義	未実装	実装済み
リスト内包表記	未実装	実装済み
タプル	未実装	実装済み
ガード	未実装	実装済み
セクション	未実装	実装済み

インデント構文やレコード構文などは未実装であるものの，関数定義やリスト内包表記の処理が可能となったため，Hachu よりも Haskell らしい表現が可能となった．また，Hachu に実装されていた単体テストを行うための関数を用いて新たに追加した構文のテストを実装し，動作を確認した．

3.2　実行環境

Hachu の実行環境は Web ブラウザ上のテキストエリアを用いた必要最低限の環境であった．初学者にとって使いやすいユーザインタフェースを提供するために実行環境の改良も行う．本研究では処理系のユーザインタフェースとして，関数を定義するためのテキストエリアと，入力した式の評価結果を表示する対話環境を提供する．関数を定義するためのテキストエリアの実装には，クライアントサイドで動作する Web ブラウザ上のエディタ である Ace [3] を使用し，対話環境の実装にはエスケープシーケンスによるターミナル制御が可能である Xterm.js [4] を用いた．エディタ内のプログラムを文字列として取得し，コンパイルにより環境を構築することによって，定義した関数をインタプリタで扱えるようにした．

3.3　拡張性

本研究で実装した処理系における各要素の行数及びソースコード全体に対する割合を表 2 に示す．

表 2　各要素の行数と割合

要素	行数 (行)	割合 (%)
パーサ	443	38.1
構文木	144	12.4
表示する Visitor	88	7.6
正格評価する Visitor	297	25.5
その他	192	16.5
合計	1164	100

処理系全体の行数は約 1200 行であり，コンパクトな処理系となっている．パーサや構文木，Visitor の実装がそれぞれ分かれて実装されているため，プログラムの見通しが良く，容易に拡張することが可能である．

拡張性の確認のために，処理系に追加する機能の例として式の評価回数を求める機能を実装した．変数 `evalCounter` を用意し，特定の構文木を評価した際にインクリメントすることで，最終的に評価した回数を求めた．この機能を処理系に追加するために，プログラムに対して 25 行の変更と 43 行の追加を行うことで実現できた．

4　おわりに

本研究は，プログラミング初学者でも手軽に Haskell を試すことが可能であり，拡張可能な研究基盤となる処理系を実装した．Hachu の拡張により，ガード，二項演算子，リスト内包表記などの初学者が学習する範囲における基本的な構文に対応した．また，実行環境は Ace や Xterm.js を用いることにより初学者向けのユーザインタフェースを実現した．

今後の課題として，未実装である構文への対応や，インデント構文のパース，型検査器の実装，遅延評価など，Haskell において重要な機能を実装する必要がある．その後，プログラミング学習を支援する機能を追加する必要がある．

参考文献

[1] The Glasgow Haskell Compiler, https://www.haskell.org/ghc/, 2019 年 1 月 17 日閲覧.
[2] 内山鷹介, JavaScript を実装言語とした Haskell のサブセット言語の実装, 卒業研究, 愛知県立大学 情報科学部, 2016 年.
[3] Ace - The High Performance Code Editor for the Web, https://ace.c9.io/, 2019 年 1 月 17 日閲覧.
[4] Xterm.js, https://xtermjs.org/, 2019 年 1 月 17 日閲覧.
[5] The Haskell 98 Language Report, https://www.haskell.org/onlinereport/, 2019 年 1 月 17 日閲覧.
[6] TypeScript - JavaScript that scales. , https://www.typescriptlang.org/, 2019 年 1 月 17 日閲覧.