様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　元 年　 ６ 月　 ７ 日現在

機関番号：１２６１１
研究種目：基盤研究(C)（一般）
研究期間：2015～2018
課題番号：１５Ｋ００９０

研究課題名（和文）プログラム意味論に基づく先進的なプログラミング環境の構築

研究課題名（英文）Advanced Programming Environment Based on Program Semantics

研究代表者
浅井　健一（Asai, Kenichi）

お茶の水女子大学・基幹研究院・准教授

研究者番号：１０２６２１５６

交付決定額（研究期間全体）：（直接経費）　　3,600,000円

研究成果の概要（和文）：初学者にも利用可能な先進的なプログラミング環境を構築した。プログラムの静的な意味論（型システム）に関する支援として、型デバッガに型スライサを組み込み、型デバッグ時の質問の回数を減らすとともに、型デバッガの効率化を行なった。また、構文の支援としてブロックインタフェースを構築し、構文エラー、型エラーの起こらない環境を提供した。プログラムの動的な意味論（実行）に関する支援としては、例外処理やモジュールをサポートしたステッパを実装し、実行の様子を簡単に観察できるようにした。

研究成果の学術的意義や社会的意義
プログラミングには、構文エラーや型エラー、実行時のエラーなど、必ずしもプログラミングの本質には関係しない困難を伴う。本研究は、適切な環境を構築すれば、これらのエラーの多くは取り除くことができることを示している。特に、プログラミング初学者は、これらの困難に足をすくわれることが多く、このような環境は重要であると考えられる。実際、本研究で構築したステッパを使った授業では、理解の役に立ったという反応を得ている。今後、小学生のプログラミング教育が始まることなどを考えると、このような技術はさらに重要になっていくと予想される。

研究成果の概要（英文）：An advanced programming environment that is suitable for novice programmers is constructed. As support for static semantics (type system), a type debugger is equipped with a type slicer, which not only reduces the number of questions during type debugging, but also makes the type debugger more efficient. As support for syntax, a block interface is constructed, in which syntax errors and type errors would never arise. As support for dynamic semantics (program execution), a stepper is constructed that supports exception handling and modules, with which a user can observe execution sequences easily.

研究分野：プログラミング言語

キーワード：開発環境　関数型言語　型システム　型デバッガ　ステッパ　初心者プログラミング教育

１．研究開始当初の背景

プログラムの意味は、静的な意味論と動的な意味論によって定義される。これらはどちらもよく研究されており、プログラミング言語の理解につながっている。しかし、これらプログラミング言語の理論が実用的な言語で実際にプログラミングを行う際に果たす役割は限定的であった。

例えば、静的な意味論は種々の型規則を導入することでプログラムの実行前にプログラムの誤りを検出することができる。しかし、ユーザがプログラムを書いたときに静的な意味論から得られるフィードバックはほとんどの場合、与えたプログラムが型チェックに通るかどうかと、通らなかった場合は必ずしも適切とは言い難いエラーメッセージだけである。これは、文法理論に基づいた統合環境が「入力プログラムが文法に合っているか」を超えて、構文エラーの起こらないプログラムの作成を（構文の色分けや適切なインデントなど）いろいろな面から支援しているのとは対照的である。

また、動的な意味論はプログラムの実行を規定し、処理系作成の基礎を与えるものである。しかし、ユーザがコンパイルしたプログラムを実行する際に得られるフィードバックは、デバッガによるスタックトレースなど特定の実装の内部情報程度で、そこからプログラムの挙動を再構成するのは容易ではない。特にプログラミング初学者は、多くの処理系実装の元になっている big-step の意味論ではなく、項書き換えに基づく small-step の意味論に従ってプログラムの動きを理解する場合が多いと考えられるが、後者に基づくフィードバックは Racket 言語など一部の例外を除いて得ることができない。

両者が揃うと、型エラーなどを機械的に排除して大きなプログラムを作成し、より複雑な動的な挙動を検討することが可能となるが、これまではどちらも不十分な状態であった。

２．研究の目的

関数型プログラミング言語 OCaml を対象に、意味論に基づく高度な機能を備えたプログラミング環境を構築する。プログラム作成時、特に初学者やプログラミング言語の専門家には、プログラミングを静的な側面と動的な側面の両方から支援することが重要である。前者は型エラーなどの自明な誤りを排除し、後者でより複雑なプログラムの理解が可能となる。本研究では具体的に、静的な側面として、型主導で型の合ったプログラムをトップダウンに作成、デバッグできる環境を、動的な側面として small-step 意味論に基づくプログラムの実行過程の可視化環境を構築する。その過程で構築の際に必要な技術を明らかにするとともに、構築した環境の予備的な評価を行う。

３．研究の方法

本研究は【静的な意味論に基づくプログラミング環境】の構築と【動的な意味論に基づくプログラミング環境】の構築のふたつからなる。これらを同時並行的に進める。

【静的な意味論に基づくプログラミング環境】の構築については、構文木を歩き回る機能、未完成の部分を展開する機能、型主導で展開する機能などを実装することで、型情報を上手にユーザに見せながら、なるべく型エラーを起こさない環境を構築する。その上で、型エラーが起きた場合は、これまでに作成している型デバッガを最大限に活用して、その質問回数の削減や効率化に取り組む。また、実装した機能は可能な範囲で授業等で使用し、問題点等を洗い出す。

【動的な意味論に基づくプログラミング環境】の構築については、まずは文献の調査を行い、それをもとに単純な言語を対象に実装を行う。Racket 言語に対するステッパは継続マークというやや特殊なものを使って定式化されているが、これを通常の限定継続で定式化し直す。その上で、OCaml 初学者向けの構文を対象にして実装する。その実装を検討することで、より大きな言語に対して実装するにはどうすればよいかを検討する。特に、big-step 意味論と small-step 意味論との関係に着目して、実装を試みる。

４．研究成果

【静的な意味論に基づくプログラミング環境】の構築については、構文のサポートとしてブロックを使ったインターフェースを構築した。また、型のサポートとしては型デバッガに型スライサを組み込むことで質問の回数を減らすとともに、型デバッガの使われ方を解析し、時間のかかっている部分を特定して改良することで効率化を図った。

(1)　当初、予定していた構文木を歩き回る機能は、実装の手間が大きい割にあまり効果が得られなさそうな見通しとなったため、代わりに OCaml の基本的な構文をサポートするブロックインタフェースを作成した。これは Google が提供する Blockly を元に改変したもので、スコープの概念を新たに導入し、コネクタの部分には型情報を持つものである。この結果、プログラム作成時に構文エラーが起こらないばかりでなく、未定義の変数が使われることもなくなった。さらに、let 多相を含む型推論を実装し OCaml と同様の型推論が行われるようにしたため、型エラーも原理的に起こらないものを構築することができた。ここでの成果は国内のワークショップにて発表し、論文賞と発表賞を得た。

現在、このブロックインタフェースを使って大学２年生向けの授業を行なっている。詳しい

評価はこれからだが、昨年までと比べて非本質的なところに足をすくわれているケースが明らかに減っている印象である。

(2)　上で作成したブロックインタフェースを使うと型エラーは原理的に起こらなくなるが、ブロックインタフェースでは OCaml の構文を全てサポートできているわけではない。そのため、ブロックインタフェースを離れると型エラーに悩まされることになる。型エラーに対応するために、これまで型デバッガを使ってきたが、そこに型エラースライサを導入し、プログラム中で型エラーに関係ない部分をデバッグ対象から除外することで、型デバッグ時の質問の回数を減らすようにした。
　ところが、デバッグ対象のプログラムが大きくなってくると、型エラースライスを求める時間が無視できなくなってくる。そこで、まず型エラースライサの中のどこで時間がかかっているかのデータを取得できるようにした。それを解析した結果、型推論を何度も行なっているのが実行時間増加の主原因であることを突き止めた。これに対応するため、型エラースライサのアルゴリズムを変更し、２分探索と同様の手法を使うことで型推論の回数を削減し、それによって型エラースライサを効率化した。

　【動的な意味論に基づくプログラミング環境】の構築については、例外処理やモジュール、副作用命令などを扱うことのできるステッパを実装するとともに、実際に授業で使用し予備的な評価をおこなった。

(3)　プログラムの実行状況を見られるツールとしてステッパを作成した。ステッパは small-step の意味論に基づくが、small-step の意味論に基づいてステッパを作成すると保守が大変になる。そこで、よりわかりやすい big-step の意味論に基づき、限定継続と動的束縛を上手に使うことでステッパを実装した。big-step の意味論に基づいているため、部分式の実行をひとまとめに捉えることができ、それを使って関数呼び出しのステップを飛ばす機能などを簡単に追加実装することができた。
　上記のステッパを拡張する形で、より進んだ機能、具体的には例外処理、モジュール、プリント文、そしてセルや配列への副作用命令をサポートした。例外処理は、コンテキストを直近のハンドラまでで区分することで、プリント文はこれまでに出力された文字列を引き回すことで、また副作用命令はメモリの書き換わる部分をユーザに見せる形で実装した。これらの進んだ機能を実装したことで、対象として考えていた授業で使われる構文を一通りサポートすることができた。
　以上に加えて、さらに「次のステップ」を必要に応じて計算する形のステッパの実装も行なった。これまでのステッパは、全ステップを求めてから表示する形だったが、これだと長いプログラムのステップ実行を見るには多くの時間がかかってしまう。それを、必要に応じて次のステップを計算する形にすることで、長いプログラムでもすぐに次のステップが見られるようになった。この改良により、無限ループするプログラムでもステップ実行が可能となった。これは、プログラムの無限ループの原因を探る有効なデバッグ法となりうる。

(4)　実装したステッパは実際に授業で使用し、その効果を確認した。ステッパのようなツールの有効性を測定するのは簡単ではないが、ステッパを導入することで「学生が課題を解き終わった時間がどのくらい変化したか」を見た。解き終わった時間を p 値 0.05 未満の片側 t 検定を行った結果、ステッパを導入したことで解き終わった時間が有意に早くなったことがわかった。また、それ以外にも、ステッパがプログラム理解に役に立ったというアンケート結果が得られた。

５．主な発表論文等

〔雑誌論文〕（計　6　件）

① T. Furukawa, Y. Cong, and K. Asai "Stepping OCaml," Trends in Functional Programming in Education (June 2018). Post proceedings version to appear in Electronic Proceedings in Theoretical Computer Science, 18 pages (2019)、査読あり、再録決定済み
② 松本 晴香、浅井 健一「Blockly をベースにした OCaml ビジュアルプログラミングエディタ」、プログラミングおよびプログラミング言語ワークショップ論文集、15 ページ(March 2019)、査読あり、http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/
③ 石尾 千晶、山田 麗、浅井 健一「Agda による PHOAS を用いた CPS 変換の正当性の証明」、プログラミングおよびプログラミング言語ワークショップ論文集、17 ページ(March 2018)、査読あり、http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/
④ 叢 悠悠、浅井 健一「限定継続命令を持つ依存型付き言語の設計」、プログラミングおよびプログラミング言語ワークショップ論文集、17 ページ(March 2018)、査読あり、http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/
⑤ K. Asai, and C. Uehara "Selective CPS Transformation for Shift and Reset," ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM '18), pp. 40-52 (January 2018).

Best Paper Award 受賞、査読あり、DOI: 10.1145/3162069

⑥ Y. Cong, and K. Asai "Implementing a stepper using delimited continuations," In J. H. Davenport and F. Ghourabi editors, 7th International Symposium on Symbolic Computation in Software Science (SCSS '16), EPiC Series in Computing, Vol. 39, pp. 42-54 (March 2016)、査読あり、DOI: 10.29007/l2wb

〔学会発表〕（計　１０　件）

① 脇川 奈穂、浅井 健一、対馬 かなえ「実用的な型エラースライサーの評価およびデータ活用に向けた取り組み」、第２１回プログラミングおよびプログラミング言語ワークショップ、２０１９年３月、湯の杜ホテル志戸平（岩手県・花巻市）

② 古川 つきの、浅井 健一「Incremental な OCaml ステッパの開発」、第２１回プログラミングおよびプログラミング言語ワークショップ、２０１９年３月、湯の杜ホテル志戸平（岩手県・花巻市）

③ 北川 舞、浅井 健一「OCaml 初学者の syntax error 調査」、第２１回プログラミングおよびプログラミング言語ワークショップ、２０１９年３月、湯の杜ホテル志戸平（岩手県・花巻市）

④ 古川 つきの、浅井 健一「OCaml ステッパの拡張」、第２０回プログラミングおよびプログラミング言語ワークショップ、２０１８年３月、皆生グランドホテル天水（鳥取県・米子市）

⑤ 脇川 奈穂、対馬 かなえ、浅井 健一「型エラースライシングを利用した型エラーデバッガに関する実装と考察」、第１９回プログラミングおよびプログラミング言語ワークショップ、２０１７年３月、華やぎの章 慶山（山梨県・笛吹市）

⑥ 脇川 奈穂、対馬 かなえ、浅井 健一「型エラースライシングを利用した型エラーデバッガに関する実装と考察」、第５８回プログラミング・シンポジウム、２０１７年１月、ラフォーレ伊東（静岡県・伊東市）

⑦ K. Asai "Principle and Practice of OCaml Type Debugger," ACM SIGPLAN Programming Languages Mentoring Workshop at ICFP (September 2016), Nara, Japan.（招待講演）

⑧ 叢 悠悠、浅井 健一「動的束縛を用いた stepper の実装」、第１８回プログラミングおよびプログラミング言語ワークショップ、２０１６年３月、ダイヤモンド瀬戸内マリンホテル（岡山県・玉野市）

⑨ 石井 柚季、浅井 健一「強い型付けを利用したプログラミング初学者のための開発環境」、第１８回プログラミングおよびプログラミング言語ワークショップ、２０１６年３月、ダイヤモンド瀬戸内マリンホテル（岡山県・玉野市）

⑩ C. Uehara, and K. Asai "Cross validation of the universe teachpack of Racket in OCaml," 4th International Workshop on Trends in Functional Programming in Education (June 2015), Sophia-Antipolis, France

６．研究組織

(1)研究分担者　なし

(2)研究協力者　なし