Java プラットホーム上での Coq 検証済みコードの実行について

湯浅 能史　田辺 良則

定理証明法によるソフトウェア検証は、数学的なモデリングが可能なら、原理的には必ず適用できるという意味で、非常に強力な検証手法である。しかしながら、実用的な適用例に関しては、残念ながら現状では限定的と評価せざるを得ない。その原因の一つは、開発現場で主流となっている言語で開発されたシステムとの親和性にあると思われる。現在、我々は、Java プラットホームを対象に、証明支援系 Coq による検証済みコードを、他のプログラムとシームレスに結合して利用する方法の研究を進めている。本講演では、Coq 言語で書かれたプログラムを、Java 仮想マシン上のインタープリタで直接実行する方法や、Java 言語の関数型への拡張である Scala 言語に変換する方法などを紹介し、それぞれの評価を元に議論したい。

1　はじめに

定理証明の方法によるソフトウェア検証は、モデル検査と並ぶ代表的な形式手法による検証方法である。数学的なモデリングが可能なら、原理的には必ず適用できるという意味で、潜在的には極めて強力な方法ではあるが、その適用事例は決して多くはない。モデル検査手法がかなり普及していることと比較すると大きな差がある。

普及を妨げる要因として、よく挙げられるのは、検証に要する工数が比較的大きいことや、この手法を扱える技術者が少ないなど、開発体制やコストに関わることである。その一方で、プログラミング言語をはじめとするシステムの開発環境や、想定している動作環境に関する要因も無視できない。実際のシステム開発においては、ソフトウェアのすべての部分が検証の観点から同等の重要性を持つわけでは無く、高度な信頼性を期待されるごく一部分にのみ検証手法を適用するのが普通である。しかし、そのような場合でも検証手法として、定理証明法が用いられることはあまり無い。技術者不足を脇に置けば、これは、作成される検証済みコードが、システムの他の部分や全体の実行環境などとの親和性が低いことに原因があると思われる。定理証明法が得意とするのは、主に関数型プログラムの検証であり、また最終的に生成される実行コードも、特定の関数型言語のものに限られていることが多い。一方、システム全体をこのような言語で開発・運用するケースは、現状では稀である。

しかし、これに関しては、近年、若干の状況の改善が見られる。Java 言語に関数型言語の特徴を統合した Scala 言語が提案され、Web サービスの開発現場などで徐々に普及してきている。このような背景を踏まえ、我々は、証明支援系 Coq で検証した関数プログラムを Java プラットホーム上で、他のプログラムとシームレスに結合して動作させる方法を研究している。我々が検討しているのは、主に以下の 2 つの方法である。

- Coq 言語の証明式からの Scala プログラム抽出
- Java 仮想マシン上で動作する Coq 言語処理系の開発

本稿では、この研究のこれまでの経緯や現状、また今後の計画について概要を述べる。

Yoshifumi Yuasa, 神奈川大学総合理学研究所, Research Institute for Integrated Science, Kanagawa University.

Yoshinori Tanabe, 鶴見大学文学部ドキュメンテーション学科, Department of Library, Archival and Information Studies School of Literature, Tsurumi University.

2 プログラム抽出

Coq システムは元々、証明からプログラムを抽出する仕組みを備えている [3]。抽出できるプログラミング言語は OCaml(ML), Haskell, Scheme の 3 つである。これらは全て関数型言語である。

Scala 言語は、その関数型言語としての側面においては、 ML に類似したものと理解できる。Java から引き継いで拡張したクラス機構は ML の型システムと凡そ自然な対応関係があり、これを踏まえれば、Coq システムの OCaml プログラム抽出ツールを Scala プログラム向けに改造することができる。今井 [2] はこれを実際に行い、抽出先言語として Scala を追加する拡張を行っている。しかし、これは Coq システムの特定の実装に強く依存しているため、そのバージョンアップに追随するコード修正が頻繁に必要となるという難点がある。

逸見ら [1] は、この点に鑑み、既存の Coq システムの実装変更の影響を受けにくい別の方法を試みている。Coq システムにおいては、証明から対象言語のプログラムを直接抽出しているわけではない。簡易 ML 言語である Mini-ML のプログラムをまず抽出し、後にこれをそれぞれの対象言語へ変換しているのである。逸見らの方法では、この中間的な Mini-ML プログラムを元に Scala コードを生成する。Mini-ML の言語仕様は シンプルで普遍性のあるものなので、Coq システムのバージョンアップ等でも、変更されることはまず無いと思われる。

一方、この方法の難点は Coq による Mini-ML コード抽出の際に多くの型情報が失われてしまう点にある。その結果、Scala コンパイラが必要とする型情報を復元することができず、変換が失敗に終わこるとがある。通常の ML 言語であれば完全な型推論が可能だが、ここで用いている Mini-ML は、任意の型キャストを行う特殊関数を含んでおり、またプログラム抽出アルゴリズムもこれに大きく依存しているため、状況の改善には相応の困難が伴うと思われる。

3 Scala による Coq 言語処理系の実装

本節で述べるのは Coq 言語を Java 仮想マシン上で直接処理する方法である。まだ研究は初期段階であり、システムの基本部分を作成しているところである。ここでは設計/実装の方針について、現時点で考えているものを簡単に説明する。

実装言語は Scala である。システムは Coq の宣言文列を読み、それらが形成する関数評価環境を Scala のオブジェクトとして作成する。他のオブジェクトは、これにメッセージとして Coq 式を送り、戻り値として評価値を得る、という形で Coq コードとの連携を図ることが可能になる。また、効率的連携を可能にするため、以下の 2 点をシステム設計上の課題として掲げた。

1. 無駄のない関数評価戦略
2. Scala 関数の Coq 言語への埋め込み

Coq 言語の証明式には、処理の記述とその正当性証明が混在している。評価戦略の如何によっては、簡約化が後者に対しても適用されてしまい、大きなオーバーヘッドになる場合もある。最終的な出力に不必要な計算はできるだけ行わずに済ませるために、遅延評価 (lazy evaluation) を基本的な評価戦略としたい。課題 2 は、既存の Scala ライブラリ等を有効に利用する上で特に重要である。これをどのような形で実現するかについては、現在検討中である。

なお、Coq 言語処理系の独自実装には、別の応用も考えられる。それは、前節で述べたた Scala コード抽出を Java 仮想マシン上で行えるようにすることである。これにより、特定の Coq システム実装への依存が無くなり、今井の方法の難点を克服することができる。

4 まとめと今後の展望

本稿では Java プラットホーム上での、Coq 検証済コードを利用する方法についての、我々のこれまでの研究を紹介した。現在、前二節に述べたツールや処理系の改善/開発を進めている。

最後に今後の展望について述べたい。本稿で述べた手法は、検証 “済み” コードをの利用を目指したも

のであるが、今後はこれまで得た知見を踏まえつつ、より発展的な研究テーマとして Java オブジェクト指向プログラミングにおける検証 “付き” コード (Proof Carrying Code) の利用についても視野に入れていきたい。関数プログラミングにおいては、基礎となる型理論を依存型に強化することで、関数定義と定理の証明を、同一の枠組みで扱うことができるようになることが知られている。Coq 等の定理証明系はこの原理 (Curry-Howard 対応) を応用したものである。型理論に対する同様の強化を、適切な形で Scala 言語でも行うことができれば、システムが提供するサービスの正当性証明を、関数オブジェクトとして表現して保持できるようになると思われる。また、状態遷移の確証を、オブジェクトの受け渡しによって他者（サービスを受ける側）に伝えることもできる。このような考え方は、契約プログラミング (Design by Contract) における形式手法や、それに基づくシステム検証等に、応用できるのではないかと予想してる。

参 考 文 献

[1] 逸見港，田辺良則，今井宜洋，萩谷昌己: 検証済みのコードによる Coq から Scala へのコード抽出． 第 31 回ソフトウェア科学会大会．(2014)

[2] Yoshihiro Imai, Coq2scala for coq8.3.(2012) http://proofcafe.org/wiki/Coq2Scala

[3] Pierre Letouzey, Certified functional programming - Program extraction within Coq proof assistant. PhD thesis, University Paris-sud (2004)

[4] The Coq Development Team. The coq proof assistant. http://coq.inria.fr

[5] The Scala Development Team. The scala programming language. http://scala-lang.org