形式的手法を広めるためには

河　野　真　治[†1]

How to promote formal method?

Shinji KONO[†1]

1. 形式手法を広めていくためには

ソフトウェアの信頼性の基本は、それがどういう性質を満たすかを理解することであり、それは本質的に論理を理解することになる。論理式による仕様記述、仕様を満たしていることの証明、仕様を満たしているかどうかを調べる様々なツールが研究開発されてきた。実用レベルの大きさのソフトウェアに適用できるモデル検証ツールや、正しさを証明された OS などできることは広がっているが、その基本は、それほど変わっていない。形式手法が広く使われない理由もあまり変わっていない。

形式手法は数学であり、それを学ぶことは難しい。書いてすぐ実行できるプログラムと異なり、その入力と出力は複雑である。それが実際のプログラミングでどう役に立つかには、さまざまな準備が必要になる。特に「形式手法ツールが何をするのか」を学ぶコストが大きい。

教育コストを含めて、実際の研究開発で使えるようなワークフローを示すことが形式手法を広げていく鍵だと考える。

2. 大学で教えていること

琉球大学情報工学科で使用している形式手法は、まず UML がある。UML はそれ自体が検証手段ではないが、さまざまな図と、その目的を理解し、図を書くことにより、自分のやろうとしていることを記号として表すことを学ぶ。形式手法の基本は、ずべてのことを記号で表すことであり、UML から始めるのは図を好む日本人に合っているようである。図としてはシーケンス図、ユースケース図、クラス図、協調図を使用している。

プログラミングでは並行実行を理解することが重要であり、それを理解するためのツールとして、Java のモデル検証系である PathFinder[3] を使用している。スレッドの競合は再現するのが難しいが、PathFinder により競合、デッドロックを見つけることができる。また、可能な実行を列挙することにより、並列実行がどういうものかを具体的に理解することができる。PathFinder は CTL(Computational Tree Logic) のような仕様記述言語を持っていないので、モデル検証ツールとしては限界があるが、逆に、CTL を教えなくてすむのは良い。

ソフトウェア工学では、Haskell を使用し、関数型言語の基本を学ぶ。λ式からカーリハワード対応を使って簡単な論理式の証明を示す。そこから Agda[5] へ誘導する。証明の例としては、圏の関手と自然変換を用いる。最後に Monad を導入する。学生はそれを理解し問題を解くことはできるが、それを実際にプログラミングに応用するところまでの道筋は遠い。

3. 研究室では

論理や関数型言語、そして証明やモデル検証を学ぶには、結局は、本や論文を読むしかない。そして、その問題を一つ一つ解いていく必要がある。これに付き合うのは長く手間のかかる作業になる。しかし、これは大学の本業であり避けて通ることはできない。すべての学生がこれに付いてこれるわけではない。一方で、形式手法は「誰がやっても同じになる」手法なので、基本は簡単である。問題を解く作業も単純なルールの適用を確認していくに過ぎない。難しいのは、その簡単なルールを膨大に組み合わせて意味のある結果を得

†1 琉球大学
University of the Ryukyus

るところにある。そこを想像できないと「わからない」ということになってしまう。つまり、わからないのではなくて、わかることに価値を見出せなくなる。

実際の研究開発では、形式手法そのものを研究にする場合では、自明に形式手法が重要となる。広い範囲をカバーするのではなく、特定の問題に集中することになる。研究対象となる並列分散プログラム自体に形式手法を適用しようとすると、企業での適用と同じような問題が生じる。

一番簡単なのは UML であり、それは有効である。UML は学生の卒論や修論にそのまま使えるものなので、積極的に書かせている。しかし UML は例示に過ぎず、仕様記述ではないので、それを使ったテストや証明はできない。

実際の形式ツールを使用するには、どういうツールがあるのかを調べる必要がある。開発環境をそれに合わせて用意し、そのツールを理解し、それを使ってバグが取れたとしても、その利益がどこにあるのか。信頼性が上がったとしても、それは目に見えにくい。形式手法を使うことにより、研究開発が早くなるのか、質が良くなるのか。それを理解して使う必要がある。

Agda による証明も積極的に導入しているが、Agda は一行書くのに何分、あるいは数時間かかるものである。その使いどころは限られている。一方で、形式手法自体を学び、論文を書くには便利なツールであるとも言える。証明には見落としがつきものであり、それを調べるのには最適なツールである。

4. ワークフローと形式手法

研究開発は分散版管理を中心に行われていて、Trac などのプロジェクト管理ツールや、github[2]/bitbucket[1] などの pullreq (プログラムへの変更を review してもらう request) などを利用することもある。しかし、研究室ではグループ作業よりは「指導教員と学生」という一対一の環境になるので、グループ管理ツールの必要性は少ない。しかし、研究状況を認識するツールとして分散版管理は重要である。

分散版管理は形式手法とは異なるものだが、それ自体を形式的に定義することができると考えて、Delta monad というのを提案している。

プログラミングを学び、形式手法を学んで理解するという教育手順そのものを学科や研究室のワークフローとして確立したいと考えている。様々なツールを実際の研究開発手順にワークフローとして組み込み、その有効性を示すことが形式手法を広げていく鍵だと思われる。

最初の一歩は UML であり、教育コストも実プロジェクトへの影響も少ない。しかも、効果は高い。

Continous Integration[4] では pullreq をワークフローの起点としている。そこに形式手法、例えばモデル検証などを組み込むことはテストを行うこととの差が小さいので受け入れられやすい。

証明を研究開発の過程に組み込むことは難しいが、Monad などを理解することにより、その方向性を示すのには有効だと思われる。将来的にプログラムのかなりの部分は証明されることになる。実際、現在使われているライブラリのほとんどは論文的には、 証明を持つのであり、それを形式手法で示すことはライブラリの信頼性を上げることになる。その範囲をプログラム全体に広げるのには時間がかかるし、可能かどうかも不明である。しかし、SSL などのセキュリティに関わる部分には証明が要求されるようになる。その時に、それに用意ができているような人材を育てる必要があると考えている。

形式手法に詳しくなくても、基本を理解し、いくつかのツールを使える。そういう人材を企業に送り込むことが遠回りではあるが、結局は、近道なのではないかと考えている。

参 考 文 献

1) Bitbucket. http://bitbucket.org/.
2) Github. http://github.com/.
3) K.Havelund and T.Pressburger. Model checking java programs using java pathfinder, 1998.
4) Jenkins: An extendable open source continous integration server. http://jenkins-ci.org/, 2014.
5) Yoshiki Kinoshita. Agda language. Technical Report PS-2008-014, 独立行政法人 産業技術総合研究所, 2008.