National Institute of
Advanced Industrial Science
and Technology
AIST

AIST-PS-2008-008

フォーマルメソッドのフィールドワーク

木下 佳樹　高井 利憲

独立行政法人 産業技術総合研究所
システム検証研究センター

算譜科学研究速報
Programming Science
Technical Report

Research Center
for Verification and Semantics

独立行政法人 産業技術総合研究所 システム検証研究センター 発行
Published by Research Center for Verification and Semantics
National Institute of Advanced Industrial Science and Technology

フォーマルメソッドのフィールドワーク

木下 佳樹　　高井 利憲

1. フィールドワークとは

フィールドワークという語はもともと民族学などでの専門用語で､野外研究と翻訳される。村落などの社会を観察し、それを説明する枠組をつくる研究、といってよいだろう。産業技術総合研究所システム検証研究センター（AIST/CVS）では、この言葉にかなりの拡大解釈をくわえて、科学者が自らの専門知識と技能をもちいて一般社会ではたらく意味にもちい、フィールドワークと科学研究の二本立てで、研究活動をすすめている。フィールドワークをとおして社会に貢献するだけでなく、その経験から科学研究の新しい豊かな対象をうみだすことができるかもしれないとの期待をもって活動している。

参加者に資するもの　我々研究者側がフィールドワークにもとめるものは、具体的な事例の獲得である。検証技法にかぎらず、ソフトウェア開発技法を大規模な開発に適用するためには、研究室内の実験では十分でなく、どうしても、開発現場での事例にふれた体験と、事例の分析が必要である。フィールドワークはまた、我々の科学研究におけるテーマ選択にもよい影響をあたえるだろうと期待される｡学術の世界にとじこもるだけではなく、社会と相互作用しつつすすむ科学研究を目指す我々にとって、フィールドワークは、外界との接点として重要である。

他方、共同研究のパートナーが、フィールドワークからえるものは、自らの仕事に適応させた形での先端技術の移転である。数理的技法を開発現場に導入するには、教科書にかいてあるような知識だけではたりない。基礎的な知識を、その場その場に適応した形に定型化する必要がある。また、現場の技術者が新しい技法をまなび、新たに発生する作業を身につける必要があるかもしれない。研究所は、フィールドワークをとおして、最新の技術をその適用現場に沿った形にして提供することができる。

具体的な課題提示　数理的検証法に関係しそうな具体的な課題をフィールドがAIST/CVS に提示するところから、フィールドワークがはじまる｡数理的検証法がその課題に有効なのかどうか、前もってフィールドの側でわかるわけではない。それが判断できるくらいなら、はじめから研究所の助けを借りずに技術導入していたはずである。この段階では、検証に直接関係しない開発現場の専門用語や概念を数理的検証法の研究者側が理解する必要がある｡これは一般には容易ではない｡フィールド自身が研究所を持っておれば、開発部門と AIST/CVS の間の通訳としてはたらくことができて好都合だろうと思われる。

課題が具体的かつ詳細に AIST/CVS の研究者の前に提示されたら、モデル検証法、対話型

検証法など、多数ある数理的検証技法のうちのどれをもちいるべきかを研究者が検討してきめ、それをフィールドの現場に適用する。現場への適用が肝心だから、現場が新しいテクニックを利用することに対して肯定的にならないと、この段階がすすまない。

縁側から奥座敷へ　大抵は、AIST/CVS が縁側に入れてもらうことからはじまる。つまり、初めての適用では、既に開発が終了しているソフトウェアや、昔作ったプロトタイプなどに対する検証をおこなってみせることからはじまることがおおいのである。記録にのこっている不具合が、数理的検証法でみつかるかどうかをみたりもする。

これをくりかえすうちに､AIST/CVS の研究者および数理的検証法へのフィールドの人々からの信頼が増し、次第に縁側から奥座敷へととおしてもらえるようになる。奥座敷とはすなわち、フィールドにおける本格的開発計画、企業であれば、待ったなしの納期が設定されているような開発案件である。そこでフィールドワークを行えるようになって初めて、技法を現実の大規模開発に適用するときの問題点が明らかになり、必要な研究を追加したり、技術者の教育方針を修正したりといったアクションをとることができる。

成果の発表　フィールドワークからえた知見は、知的財産ではなく学術的な知識だから、学術誌などに発表して研究コミュニティで共有すべきである。しかし、フィールドワークによる野外科学の研究のすすめ方は、実験科学や書齋科学とは異なるので、発表の手順や問題点に自ずから異なる点がいくつかある。

まず、共同研究先の開発過程あるいはそこで稼動しているシステムを題材に使う研究なので、題材自体に共同研究先の重要な秘密情報が含まれている場合が多い。そこで、発表前に共同研究先との十分な打ち合わせが必要である。また、この打ち合わせをスムーズにすすめるには、共同研究開始時に必要な条項を契約書に記しておくことが有効である。例えば、開示情報についての合意を発表のたびにおこなうのかどうか、また、秘密情報に関しても、永久に秘密とするのか、一定の年月の後には開示してよいのかどうか、開示してよいとしても合意の上で開示するのか合意が必要ないのか、などという点である。

題材に秘密情報が含まれるといっても、それはたいていの場合、遂行中のフィールドワークとは直接関係ない情報である場合がほとんどである。たとえば、DVD プレーヤーの組込みシステムを検証するフィールドワークにおいて､再生画像の美しさに関するノウハウは、プレーヤーを開発するメーカーにとっては重大な秘密情報になりうるが、プレーヤーの動作検証をおこなう立場からは、あまり興味のない情報である。検証に直接関係のない企業秘密をうまく隠蔽しながら検証の様子だけを発表することは、容易な場合が多い。いっぽう、不具合そのものは、大抵の共同研究パートナーにとって、重大な秘密情報である。製品情報をうまく隠蔽しながら、不具合の本質を浮き彫りにするようなプレゼンテーション

には工夫が必要だが、不具合の実例の本質的な部分を産業界および学界で共有することができるとすれば、社会全体のシステムの信頼性向上にとって大きな利益となる。

臨床研究としてのフィールドワーク　フィールドワークは医学でいう臨床研究と同様で、短期間で成果を得ること、とくに論文の出版に至ることが難しい。学位を早急に取得しなければならない学徒には困難な道といえよう。しかし、産学協同を真剣に求めるとすれば、フィールドワークあるいはこれと似たような活動が、産業界などの一般社会における問題の解決に最も効果的なのではないだろうか。

2. フィールドワークの事例紹介

本節では、AIST/CVS の設立以来、これまでに携わった事例のうちから、典型的なものを三件えらんで紹介する。自動車、精密機器、製造機器などの組込みシステムに関するものばかりである。ソフトウェアの信頼性向上への要求が、組込みシステムの業界において特に強いように感じられるのは興味深い。

(1)では、矢崎総業グループにおけるシステム開発過程の信頼性をさらに向上させるため、モデル検査手法の導入実験をおこなった経過を紹介する。これは六年間にわたる共同研究の中間成果というべきものであり、個別の検証事例がこの研究から多数生まれているが、それらは既にさまざまな機会に発表済である。本稿では、この共同研究でこれまでに得られた成果を概観して考察をくわえる。

(2)は特定の不具合の解析にモデル検査を用いた事例である。ハードウェアおよび機械部分を含めたシステム全体の挙動の中に不具合が観察された、という例に関して、アセンブリ言語で書かれたソースコードと仕様書、ハードウェアの仕様書、不具合の観察記録などの提供を企業から受けて、不具合の挙動の原因をモデル検査によって調べたものである。

(3)は開発が終わってから検証するのではなく、開発過程に検証過程を埋め込み、相互に入り組ませた開発･検証過程の実現を試みた事例である。不具合をいくつか検出したが、検証をおこなう我々は、検証にとどまらず、開発にも参加した。検証と開発は、交互におこなわれながらすすんでいくもので、両者をきれいに切り分けることができない、という我々の直観を裏付ける事例ともいえる。

(1)数理的技法の導入による信頼性向上を目指して

はじめに紹介する事例は、車載組込みソフトウェアの品質および信頼性向上に関する矢崎総業グループ（以下では矢崎総業）との共同研究である。我々にとってのこの共同研究の目標は、モデル検査技術の製品開発の現場におけるフィージビリティを確かめることであ

る。いっぽう、反例の出力によるバグ発見を特徴とし、ツールも整備されつつあったモデル検査は、数理的技法のなかでももっとも実用化に近い技術であるとの予想が、共同研究開始当初から産業側にあった。

共同研究当初は手探りの状態で、文献などの調査からはじめたが、すぐに実地に試してみようということになり、まず過去に開発されたプロトタイプを対象としてモデル検査による検証をおこなってみた。その過程でしだいに産業側からの信頼を得、次は、過去に開発された製品（プロトタイプではない）を対象として同様のことをおこなった。不具合の記録が残っているので、それを我々には隠してもらい、その不具合がモデル検査によって見つかるかどうか、といった一種のブラインドテストを行なったりもした。結果が良好だったので、さらに、新製品の検証を、従来手法による検証と並行させておこなう、といった試みもおこなえるようになった。「縁側」から「奥座敷」にしだいに移っていったのである。

開発現場の様々な開発工程で、モデル検査による検証をおこなう適用実験を繰り返した。共同研究の途中からは、開発過程のいわゆるＶ字モデルに基づいて考えるようになり、要求分析、基本設計、詳細設計、実装の各工程で適用実験を実施した。検査対象も、開発が終了したものから、既知のバグの存在するものや存在しないもの、開発途中のものなど、様々であった。それらの実験の過程から、モデル検査を開発現場に導入するための観察をいくつか得た。その中から三つを紹介する。これらの観察はこの事例に対するものであるから、どこが一般的なもので、どこがこの事例に特別なものであるか、ということの分析は今後をまたねばならない。

観察の一つ目は「モデル検査の効果は、下流工程の実装段階の検証におけるのがもっとも高い」ということである。一般には、モデル検査をはじめとする数理的技法は上流工程に使うのが有効だとされる[13]。我々も、他のプロジェクトにおいてはそのような観察を得ている。仕様の設計などの上流工程における間違いの方が、手戻りが生じた場合のコストが大きいことや、実装などの下流工程ではそのままモデル化すると、状態爆発を引き起こしやすいなどの理由からであろうと思われる。

しかし、この事例において上流工程への適用を試みると、システムの外界の振舞いを忠実に再現し検査をおこなえる程度にモデルを詳細に作成することが非常に困難であることがわかった。検査対象に関する情報を十分にえるためには、開発者への聞き取り調査が必要であるが、研究所にいるモデル検査の担当者と、企業内で開発を担当するエンジニアとが意思疎通をするのは、用語や背景知識や概念の共有が十分でないために、困難であり、かつきわめて時間がかかるのである。このような意思疎通は、一種の異文化交流であるとさえいえる。また、開発担当者は開発作業に多忙であるから、共同研究参加者からのインタビューに長い時間をかけて応じることができるとは限らない。

一方、実装段階では、仕様が既にプログラミング言語などの形で形式化されており、背景知識が不足していても正確な理解が可能である。また、開発現場も、実装段階での品質向上を望んでいた。そこで、ソースコードレベルでの実装段階の検証を試みることにした。ソースコードのモデル検査では、状態数の爆発が課題であることはよく知られているとおりである。そこで、状態爆発問題を解決する方法を模索し、何回かの適用実験を通して、環境ドライバの概念を導入し、これに基づく検証シナリオを開発した[4]。環境ドライバ法では、周期駆動型システムのモデル化を、状態爆発を起こさないようにしながらおこなう手法である。周期駆動型システムは、制御システムにおける代表的なソフトウェアアーキテクチャのひとつであるから、環境ドライバ法は、組込みシステムをはじめとする制御システムに広く応用することができる。なお、環境ドライバの概念は、フィードバック系における「外界」、HILS (hardware in the loop simulation) における「ループ」にほぼ相当するが、尾崎らはそれらとは独立にこの概念を導入した。また、環境ドライバ法における状態数削減の手法は、いわゆる cone of influence reduction と関連すると思われる。

「もっとも時間を要するのは、検査対象の理解である」というのが、この事例から得た二番目の観察である。一般に、モデル検査による検証シナリオは、1)検査対象の理解、2)検査対象のモデル化および検査項目の翻訳、3)モデル検査、4)反例解析の四つの段階から構成される。この中で、2)、3)は、共同研究の進展によるノウハウの蓄積や技術者への教育を通じて、時間を短縮することができた。また、4)は適当なツールの導入で大幅な時間短縮が可能だと思われる[12,13]。しかし、特に、検証技術の研究者など、開発のフィールドの外の者が検証を担当する場合、1)にもっとも時間を要した。開発者と知識の共有が十分でないことが主な理由であろうと考えられる。したがって、第三者検証ではなく、開発過程の一部としてモデル検査による検証をおこなう場合には、検査対象を開発している技術者自身、またはそれに近い者が直接モデル検査をおこなうほうが効果的なのではないかと考えられる。この実現のためには、技術者に対してモデル検査技術の教育を施すことが必要である。本共同研究のなかだけでも数多くのモデル検査の教育プログラムを実施し、また後述するように、AIST/CVS では、一般技術者向けの研修コースも開発している。

三つ目の観察は「多くのソフトウェア技術者にとって難しいのは遷移系の記述ではなく、検査項目の記述である」ということである。遷移系の記述は行数が多いが、必要な概念はプログラミングにあるので、ソフトウェア技術者にとって殆ど困難はない。しかし、検査項目の記述は、時相論理の論理式でなされなければならず、大抵のソフトウェア技術者は習得に苦労する。時相論理の論理式の意味は無限列や無限木の集合で与えられるが、無限の概念に関する数学的リテラシーの不足が、ここでの苦労の原因の本質であると考えられる。このような課題を克服するソフトウェア技術者教育が求められる。

時相論理に対する現場技術者の心理的ハードルだけでも低くしようと、我々は検査項目に

対する直観的な理解を促す図的な表現、すなわち図示記法を考案した[2]。とくに時相論理LTLの部分クラスに対する図的な表記法を定式化した[3]。図示記法を用いることによって、検査式の作成の習得やデバッグの心理的難易度を下げ、検査内容を伝達するためのコミュニケーション手段を得ることができた。真に必要な数学的リテラシーがソフトウェア技術者の間に広がるまでの間の過渡的な知的道具として、あるいは初学者のための簡易な導入法として、図示記法は有効であると考えている。

以上に記したような共同研究を、2002年以来六年間にわたって遂行し、数理的技法を現場導入する際の必要条件である検査工数の削減に成功した。13回にわたる適用実験をおこなった結果、初期の事例では、2ヶ月程度の時間を要した検査実験が、2005年度には、9日間で終了することとなった[4,5]。

(2)不具合解析を目指して

二つ目の事例は、ソフトウェア、ハードウェア、機械部分のすべてを含めた組込みシステムの不具合解析に、数理的技法を適用する試みである。ハードウェアの故障解析手法にはいくつかの定石があり、対象を限定すれば解析のシナリオが確立されている場合も多い。しかし、ソフトウェアの不具合解析のシナリオは、未だ定まっているとは言いがたい。通常のデバッグ技術や、テスト、レビューなど、要素となる手法はあるが、それらをどう組み合わせるのか、また、これらの手法だけで十分なのかどうかなど、未開拓の部分が多い。

ある企業（以下パートナー企業）が、ある組込みシステムの原因不明の不具合の解析を、AIST/CVSに相談したことが共同研究開始のきっかけとなった。不具合の現象は実機テストにより確認できるが、原因がどこにあるのかわからず、機械部分にあるのか、ハードウェアにあるのか、ソフトウェアにあるのかすら不明であった。不具合が観察された組込みシステムは、ハードウェアに関しては徹底的に調査をおこない、ソフトウェアに対しても、テストやレビューにより、不具合解析をおこなっていたが、原因究明にはいたっていなかった。AIST/CVSとしても、テストやレビューではできなかったことが、数理的技法によりはじめて可能になるとすれば、数理的技法の有効性を示すよい材料になると考え、共同研究を開始した。三ヶ月の共同研究遂行の結果、不具合の全容解明にはいたらなかったが、ソースコードにおいて、アドレッシングモードの間違いをモデル化の過程で発見した。この経験から、数理的技法を効果的に用いることにより、ソフトウェアの不具合解析に限らず、安全性をはじめとするいろいろな性質を分析するためのシナリオを開発することができるのではないかと考えている。

この事例で不具合解析の対象としたのは、ソフトウェアである。ソースコードは約一万行のアセンブラで記述されたプログラムであった。また、ソースコードとほぼ対応するよう

な詳細なフローチャートが存在し、解析過程で参照することができた。ほかに、ハードウェアの仕様書、および不具合の観察記録も参照した。ソースコードを、人手によって適宜抽象化しながらモデル検査器 Spin によってモデル検査した。フローチャートではなく、状態数爆発を引き起こしがちなソースコードを、あえてモデル検査の対象としたのは、テストやソースコードや仕様書に対するレビューでは不具合が検出できず、ソースコード上の微妙な間違いが不具合を引き起こしていると予想したためである。

解析はアセンブラプログラムを対象とする段階と、フローチャートを対象とする段階の二段階にわけて実施した。まず、アセンブラのソースコードすべてをモデル化し、モデル検査をするという方針で始めた。アセンブラプログラムを半自動的に Promela に変換することもおこなった。アセンブラのコードを Promela に変換する簡単なプログラムを Perl で実装し、変換されたものを必要に応じて修正する半自動的な変換によりモデルを作成したが、このような素朴な方法で作成したモデルでは簡単に状態数爆発を発生させてしまい、Spin の通常モードではなく、シミュレーションモードでしかモデル検査ができなかった。したがって、ソースコードを忠実にモデル検査したわけではない。

しかし、このモデル化の過程で、値とアドレスを取り違えるアドレッシングモードの間違いを発見した [6]。これは、ソースコードをモデル記述言語で記述しようとしたことから発見されたものだから、モデル検査によって発見したというよりも、形式記述という行為の直接の効果によって発見したといえるだろう。なお、Spin のシミュレーションモードを用いて、上記の間違いのあるコードに実行が到達する可能性があることも、実行パスを提示して証明できた。したがって、このコーディングの間違いはバグと呼ぶに値するものであるといえる。

第二段階では、ソースコードではなく、フローチャートから我々が抽出した抽象的な遷移系を対象に解析をすすめた。発見されたアドレッシングモードの間違いによるバグが、報告された不具合につながるか否かをしらべるのが目的である。バグに関係する部分を、フローチャートから抜き出した抽象的なモデルを対象に、報告された不具合が発生するか否かをモデル検査によってしらべた。反例がえられれば、対応する実機上での実行パスをしらべ、ソースコードの上での不具合解析を続けることができる、というシナリオである。しかし、反例がえられなければ、その後解析のすすめようがなく、アドレッシングモードの間違いが不具合の原因であるとも原因でないともいえない、ということになる。

検証作業の結果、反例はえられなかった。したがって、アドレッシングモードの間違いと不具合との因果関係は不明なままである。アセンブラのソースコードのレベルではこのように結論するしかない。けれども、フローチャートのレベルでは、両者の因果関係がない、ということを以下のようにして考察することができる。実機で稼動しているアセンブラの

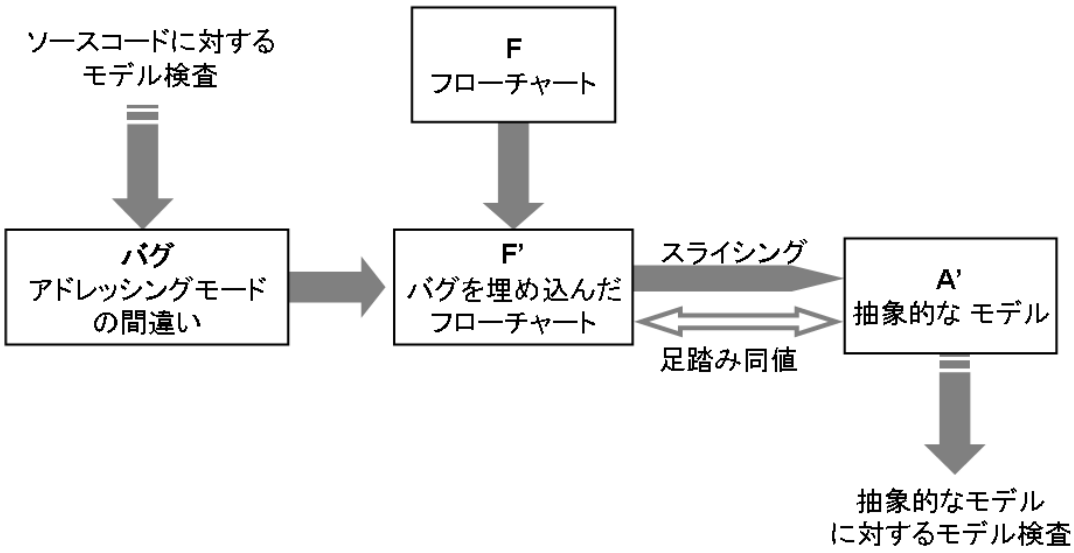

図1: 不具合解析事例におけるフローチャートの解析

ソースコードプログラムとフローチャートプログラムとの間に模倣関係があるかどうかまでを証明することができていないので、この考察からすぐにアドレッシングモードの間違いと報告された不具合との間に因果関係がないと結論することはできないが、因果関係がないことの蓋然性を与える材料にはなる。

さて、フローチャートのレベルでアドレッシングモードの間違いと不具合の因果関係の有無を調べるために、まず、もとのフローチャートFにアドレッシングモードの間違いを埋め込んだ。このフローチャートをF'とする（図1)。いわゆるスライシングのテクニックを用いてF'から抽象モデルA'を抽出し、状態数削減を試みた。この抽象モデルA'が報告された不具合を持たないことをモデル検査によって検査したところ反例はえられず､A'は不具合を引き起こさないと結論した。いっぽう、A'とF'がどのような関係にあるかをしらべ、両者が足踏み同値(stuttering equivalence) [8]の関係（いわゆる弱双模倣 weak bisimulation と同義）にあることを確かめた。足踏み同値なモデル間では、next 演算子を含まない LTL を保存していることが知られている。F'が反例を持たないので、今回発見したバグと報告された不具合の間には因果関係がないと結論した [7]。

さて、さらに、パートナー企業側が挙げていた別のソースコード上の原因候補をモデル化し、それらが報告不具合にどのように影響するのかの解析もモデル検査を用いておこなった。その結果、不具合間の因果関係を明らかにすることができた。

上記の考察は、抽象的なモデルを注意深く作成して検査することにより、抽象的なレベルにおいてソフトウェアモジュールを不具合の原因候補からはずすことに、一定の根拠が得られることを意味する。ここで鍵になるのは、模倣関係の概念である。本研究のシナリオを普遍化して、一般的な不具合解析シナリオを作るのは今後の研究課題である。

本プロジェクトは、企業との共同研究としておこなったが、期間が 3 ヶ月と短く、リソースも十分には得られなかったこともあって、不具合の全体像を明らかにすることはできなかった。遷移系モデルに不具合を埋め込んでモデル検査し、不具合の原因解析を進めるというシナリオには臨床医学とのアナロジーがあり、示唆的である。

(3)開発と検証の融合

最後の事例は株式会社イシダとの共同研究である。産総研からはシステム検証研究センターだけではなく、グリッド研究センターからも参加者を得た。フィールドワークの枠組みで、検証だけでなく、実際のソフトウェアの開発もおこない、その傍らで検証もおこなったというものである。検証に関しても、モデル検査のみならず、インフォーマル定理証明による検証も導入した。工場機器に対するセキュアな自動遠隔ソフトウェア更新システムのプロトタイプの開発過程のなかにさまざまな数理的手法による検証を埋め込み、システムセキュリティの信頼性を総合的に上げようとする野心的なプロジェクトであった。

対象システムは、工場機器の自動遠隔ソフトウェア更新システム、すなわち工場内の組込み機器のプログラムの更新を、インターネットやイントラネットを通じて提供するものである。システムの満たすべき性質の中で、今回検査対象としたものは、更新プログラムが改竄されないこと、更新プログラムが最新であること、などのセキュリティに関する性質である。設計および開発は、暗号系などを提供する既存のコンポーネントを、セキュリティ上の性質を考慮しつつ組み合わせておこなった。これらの既存のコンポーネントの秘匿性や認証性を仮定すればプログラムに関する上記の性質が成り立つことを検証した。

開発はまず、プロトコルの設計から始めた。実際の製品で使用するプロトコルに対して、検証技術によるだけでなく、不具合のない利用実績を考慮した proven by use の観点からもセキュリティに関するアドバイスをしながらの開発となった。数理的技法による検証は、プロトコルに対するモデル検査[9] および定理証明[10]、また、ソースコードに対するモデル検査[11]をおこなった。

次に、モデル検査によってプロトコルの認証性、つまり通信手順のせいで、偽のデータを受け取ってしまうことがないか、相手に成りすました他人と通信していることがないか、などといったことを検証した。モデル作成の段階で、不正な工場機器が存在する場合には、

誤った更新プログラムを受理する可能性が明らかになった[10]。モデル検査によって新たな問題は出現しなかったが、上記問題や、仕様書のレビューの段階で発見した問題を、モデル検査器の上で再現することが、共同研究参加者間で情報を曖昧さなく正しく共有する上で有益であった。これら発見されたバグの情報に基づき、プロトタイプの改良につなげた。

さらに、インフォーマルな（つまり人手による）定理証明にもとづく検証を、BAN Logicを用いておこなった[9]。一定の枠組みのもとで、改竄の検出可能性を証明した。また、更新ソフトウェアが最新でない場合があることを、反例を与えて示すことができた。そして、更新ソフトウェアを常に最新のものに保つための実施可能な改善案をBAN logicにおける有効性の証明とともに示した。

このプロジェクトでは、モデル検査および定理証明など複数の検証手法の適用をこころみ、検証の結果をプロトタイプ開発に反映させることができた。その結果、定理証明による検証で得られた結果をどのように、開発現場で位置づけるかという問題が課題として残された。つまり、モデル検査に比べて、定理証明の開発現場への導入は、より困難であると思われる。モデル検査では、命題が成り立つという情報か反例が得られるが、どちらも論理学になじみのない者にもよくわかるものであり、開発現場の技術者にとって有益な情報である。しかし、定理証明では、命題を証明できた場合はともかく、証明できなかった場合には何も情報が得られない場合がある。とくに構成的な証明法をもちいていない場合には、命題が成り立たなくとも反例が得られるとは限らない。このような道具立てを現状でのソフトウェア開発現場でもちいるには、より一層詳しい説明を技術者に提供する必要がある。

3. まとめ

フィールドワークを通して、先端技術に関する知見と技能を共同研究先に移転することが、フィールドワークに期待される効果の一つであることはいうまでもない。そのためには、研究開始時はともかく、開始後一定の期間が経過した後には共同研究相手の技術者がモデル検査などの先端技術を使って検証作業に携わることが必要である。そのためにはもちろん、技術者に技術を教えなければならない。要するに、フィールドワークは人材養成活動を必然的に生み出すものであり、人材養成と切り離して考えることはできない。AIST/CVSでも、モデル検査の実用研究において、技術者養成の必要が生じたため、それを独立させて、モデル検査に限らず、数理的技法一般における人材養成の事業をはじめている[1]。

フィールドワークに期待される効果のもう一つのものは、この経験から科学研究の新しい方向を生み出すことである。本稿で紹介した例だけでも、図示記法というグラフィックな記述法の研究、環境ドライバの概念の定式化、故障解析の枠組など、今後の科学研究の方

向への示唆がいくつかある。学術研究が象牙の塔にこもって閉塞状況に陥ることを避ける意味からも、フィールドワークを重視していきたいと我々は考えている。

最後に、本稿で紹介したフィールドワークにおいて共同研究の相手を勤めてくださった矢崎総業株式会社および関連会社、株式会社イシダおよび事例(2)における匿名の企業に、共同研究を遂行していただいたこと、および本稿の発表を快諾いただいたことに対して厚くお礼申し上げます。これらの企業の協力がなければ、フィールドワークは成り立ちませんでした。

参考文献

[1] システム検証研究センター：4 日で学ぶモデル検査初級編，エヌ・ティー・エス，2006 年.

[2] 小池憲史，吉田聡，大崎人士: LTL モデル検査の為の図示記法，第 14 回ソフトウェア工学の基礎ワークショップ(FOSE2007)，下関，日本ソフトウェア科学会，2007年11 月.

[3] 吉田聡，竹内 泉，小池 憲史，大崎 人士：図示記法表現と LTL 論理式，PPL，2008 年 3 月.

[4] 高井利憲，古橋隆宏，尾崎弘幸，大崎人士：環境ドライバを用いたモデル検査による検証事例，DSW 名古屋，2007 年 11 月.

[5] 河本孝久，小池憲史，古橋隆宏，鈴木伸一: 周期タスク型モデル検査法と車載組込みシステムへの適用事例,組込みシステムシンポジウム(ESS2007)，東京，情報処理学会，2007 年.

[6] 高井利憲，吉田聡: アセンブラプログラムのモデル検査による検証事例，第 5 回ディペンダブルシステムワークショップ(DSW2007)，函館，日本ソフトウェア科学会，2007 年 7 月.

[7] 高井利憲，吉田聡：アセンブラプログラムのモデル検査によるバグ解析事例，DSW 名古屋，2007 年 11 月.

[8] E. M. Clarke, O. Grumberg, D. Peled, *Model Checking*, MIT Press, 1999.

[9] 吉田聡，山形頼之: ソフトウェア更新システムプロトコルの BAN Logic による安全性検証(Preliminary Version)，算譜意味論研究速報，PS-2007-006 ，2007 年 6 月.

[10] 山形頼之, 斎藤正也: ソフトウェア更新システムのモデル検査によるセキュリティ, 算譜意味論研究速報, PS-2007-008, 2007 年 7 月.

[11] 斎藤正也, 高井利憲, 池上大介: Java の例外処理の SPIN による検証, 第三回システム検証の科学技術シンポジウム, 豊中, 2006 年 10 月.

[12] 篠崎孝一, 太田弘, 早水公二, 星野光勇, 今村哲典, 吉田雅昭:モデル検査の実用化課題と支援ソフトウェアの開発, 第三回システム検証の科学技術シンポジウム, 2006年11月.

[13] 栗田太郎, 太田豊一, 中津川泰正: モバイル FeliCa IC チップ開発における形式仕様記述手法導入の効果と課題, 第 25 回 Software Symposium 2005, 2005 年 6 月

フォーマルメソッドのフィールドワーク
(算譜科学研究速報)
発行日 2008 年 3 月 13 日
編集・発行：独立行政法人産業技術総合研究所システム検証研究センター
同連絡先：〒563-8577 大阪府池田市緑丘 1-8-31
e-mail：informatics-inquiry@m.aist.go.jp

Fieldwork of formal methods (in Japanese)
Mar. 13, 2008
Research Center for Verification and Semantics (CVS)
Ikeda Site
National Institute of Advanced Industrial Science and Technology (AIST)
1-8-31 Midorigaoka, Ikeda, Osaka, 563-8577, Japan
e-mail: informatics-inquiry@m.aist.go.jp

AIST01-J00022-81