

## 成果の概要

# 移動コードを基本とした セキュアなプログラミング言語処理系 ——ソフトウェアの欠陥を防止するための Systematicなソリューション——

研究代表者 米澤 明憲（東京大学）

## 1 背景

一般に現代のコンピュータは、大きく分けてCPU、メモリ、ハードディスク等のハードウェアと、OS、WWWブラウザ、メール、ワープロ等のソフトウェアという二つの要素から構成される。電子機械技術の進歩により、現代のハードウェアは性能のみならず、信頼性・安定性の点においても向上が著しい。ところが、その上で動作すべきソフトウェアに関しては、設計開発技術の進歩が大規模化や多機能化に追従できず、その信頼性・安定性はむしろ低下の傾向が顕著である。たとえば、現在の一般的な家庭用ないしは企業用パソコンにおいて、アプリケーションが突如としてエラーを起こしデータが失われる、あるいは金融機関のシステムが誤動作して、誤った振り込みや引き落としが起こる、といった問題のほとんどは、ハードウェアではなくソフトウェアの欠陥が原因である。

こういったソフトウェアの欠陥は、不特定多数のコンピュータがインターネットのような広域分散ネットワークで結合され、それらを媒体として様々なデータやプログラムが交換・移動される今日の環境では、さらに重大となる。というのは、そのようなソフトウェアの欠陥を悪用することで、企業や個人の秘密情報を盗み出したり、場合によってはさらに他人のコンピュータを完全に乗っ取ってしまうことも可能なためである。実際にそのような被害が多発していることは、各種機関によって報告されている。

以上のような問題にたいして、一般に現在のコンピュータ産業界では、できるだけ欠陥が発生しないように人間が注意確認し、それでも欠陥が発見されたらそのたびに修正するという、あまり根本的な解決策になっていない場当たりのアプローチがとられることが多い。

## 2 研究概要

上述のような状況を改善するために、本研究班では、ソフトウェアの開発においてもっとも重要な要素であるプログラミング言語（コンピュータへの命令を記述するための言語）の様々な問題について、場当たりの対症療法ではなく、堅固な理論にもとづく系統的な解決策を与えることを目指している。以下に、本年度の代表的な個別成果を要約する。

### 2.1 暗号 $\lambda$ 計算

インターネットのように不特定多数が参加するオープンなネットワークにおいて、秘密情報を他者に漏洩しないようにやりとりするためには、通信の内容を暗号化するのが一般的である。そのような暗号系の安全性は、従来から活発に研究されてきた。

ところが、たとえ暗号系自体が数学的に安全であったとしても、それを利用するプログラムに欠陥があれば、やはり秘密情報が漏洩してしまう。実際に報告される漏洩の事例も、暗号系自体の問題ではなくプログラムの欠陥が原因で

ある場合が大半である。

そこで本研究では、暗号を利用するプログラム全体の安全性を数理論理的に議論するために、プログラミング言語の代表的なモデルである  $\lambda$  計算を暗号操作で拡張した暗号  $\lambda$  計算の体系を定義した。さらに、 $\lambda$  計算の一種である多相  $\lambda$  計算におけるパラメタ性の理論を、自明でない方法で暗号  $\lambda$  計算に導入し、暗号を利用したプログラムについて、(暗号系が完全ならば) 実際に秘密情報が漏洩しないことを厳密に証明するための理論を確立した。

本研究は PPL 2001 (日本ソフトウェア科学会) および CSFW'14 (IEEE) にて、それぞれ査読をへて採択・発表され、PPL 2001 では論文賞を受賞した。

## 2.2 Fail-Safe な C 言語処理系

C 言語は、1970 年代に提案された古典的なプログラミング言語の一つで、現在でも広範に使用されている。UNIX という代表的なオペレーティングシステムを実装するために開発されたこともあって、もっとも原始的なプログラミング言語であるアセンブリ言語に近い、低水準の操作を詳細に記述することが可能である。

ところが、このような C 言語の特徴は一方でプログラマのミスを誘発しやすく、プログラムの複雑化ともあいまって、ソフトウェアの欠陥が多発する主要な原因の一つとなっている。アカデミックな分野では ML や Scheme といった、より安全な言語も利用されているが、プログラムを教育するコストなどの社会的要因により、産業界で普及するにはいたっていない。

そこで本研究では、たとえプログラムに欠陥があっても「コンピュータを乗っ取られる」「データが壊される」といった致命的影響がない (= Fail-Safe な) C 言語の実装方式を提案・実験した。具体的な方法は以下のとおりである。C 言語の仕様には不十分な部分があり、多くの誤った操作の結果がエラーではなく「未定義」とされている (バッファ溢れや誤った型変換など)。前述のような致命的影響は、そのような未定義の操作の、実際の結果である場合がほとんどである。そこ

で、結果が未定義とされているような危険な操作を実行時検査によりすべて検出し、致命的影響をおよぼす以前にプログラムを中断する、という方法である。

我々の実験によれば、上述のような実行時検査による速度低下は、単純な実現方式でも約 1 倍～十数倍以下であった。また、我々とは独立な研究者らの実験によれば、事前にプログラムを解析して無駄な実行時検査を削減することにより、平均で数十%以下、最悪の場合でも 2~3 倍程度に速度低下がおさえられたと報告されている。

## 2.3 Java におけるオブジェクト使用解析

オブジェクト指向は、プログラムの内部に存在するデータ構造や、プログラムの外部に存在する計算資源 (ファイルやネットワーク通信手段など) を、オブジェクトと呼ばれる単位で扱う、ソフトウェアの設計開発において現在では主流のパラダイムである。このパラダイムにもとづくプログラミングでは、各々のオブジェクトに対して不可能な操作を行わない (たとえば、開いていないファイルに対して読み取りや書き込みを行わない) という性質が必要とされる。

しかし、通常のオブジェクト指向プログラミング言語では、オブジェクトに対して可能な操作の集合を定義することはできても、それらの操作の順序は保証できなかった。そのために「ファイルを読み書きするには、まず開かなければならない」といった性質を十分に検査することができず、ソフトウェアの欠陥につながっていた。

このような問題を解消するために、本研究では代表的なオブジェクト指向プログラミング言語である Java を対象に、オブジェクトに対して可能な操作の集合のみならず順序をも解析・検査する体系を定義した。Java のような命令型言語における重要な問題 (エイリアシング、破壊的代入など) に対処しており、類似の研究と比較しても技術的に新規性がある。

本研究は A01-03 班 小林直樹 東京工業大学 助教授との共同研究である。