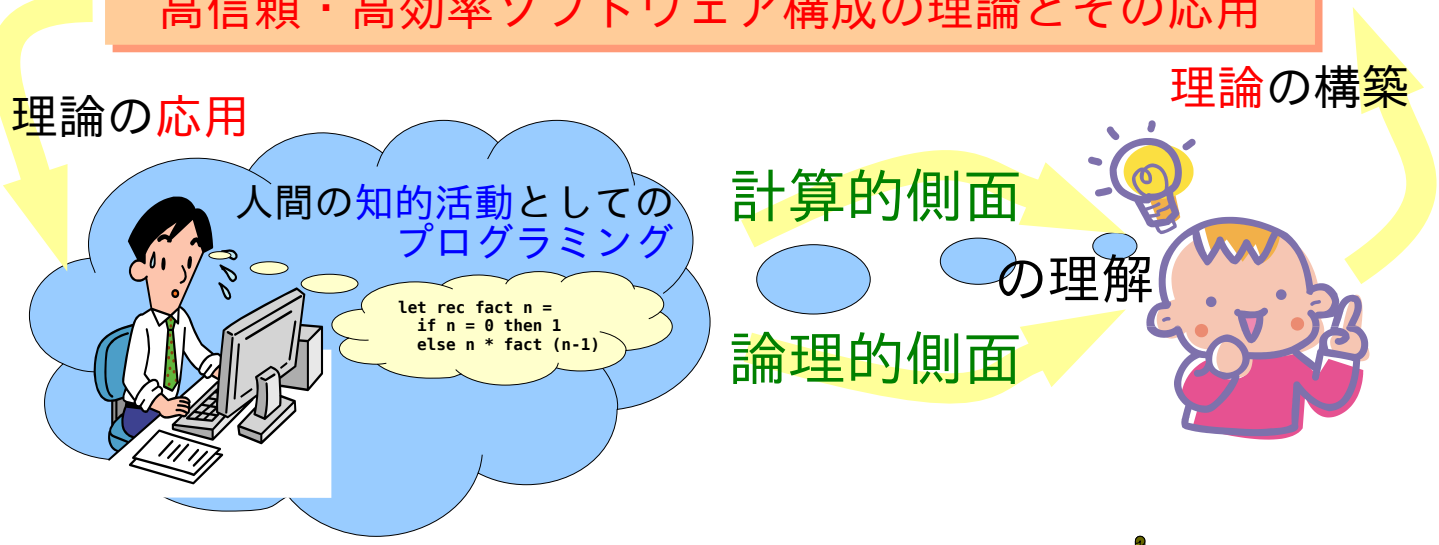
京都大学 大学院情報学研究科 通信情報システム専攻
コンピュータソフトウェア分野
教員：五十嵐淳 教授　末永幸平 准教授　和賀正樹 助教
http://www.fos.kuis.kyoto-u.ac.jp/

研究内容
高信頼・高効率ソフトウェア構成の理論とその応用



現在取り組んでいる主な研究課題

高信頼ソフトウェアのための形式検証

ソフトウェアが社会の様々な場所で使われている現在、それらのソフトウェアが高信頼であることが非常に重要になっている．当分野ではソフトウェアの信頼性を高めるために形式検証の研究に取り組んでいる．形式手法は、数学的に正しさの証明されたアルゴリズムでソフトウェアを解析し、ソフトウェアの正しさを「証明」する手法で、最近になって産業界においても注目を集めている．

具体的には（1）ポインタ操作を伴うプログラムを所有権という考えを利用して検証する手法の研究、(2) ハイブリッドシステム（ソフトウェアと力学・電気系が協働するシステム）の検証、(3) スマートコントラクト(ブロックチェーン上で動作する決済などを行うプログラム)の検証、(4)検証手法が内部で用いているヒューリスティックを強化学習を用いて高速化する研究などを行っている．

物理情報システムのための軽量形式手法

自動車やロボットの様な物理情報システム(cyber-physical system, CPS)と呼ばれるシステムは複雑なソフトウェアによって制御されている一方でシステムの物理的特性も重要となるため、システムの形式的なモデルの難易度が高い等、従来のソフトウェアの検証とは異なる側面がある。またCPSは連続的な物理的特性のためにモデル検査を始めとした従来の形式手法の適用が計算量的に難しいため、より軽量な形式手法が注目されている。
具体的には（1）ブラックボックス検査と呼ばれる、システムの形式的なモデルの学習と形式検証を組み合わせたテスト手法や、(2) オートマトンや時相論理等の形式仕様を用いたシステムモニタリングを、より多彩なデータや仕様に対して扱える様に拡張する研究を行っている。

静的検証・動的検査の融合

形式手法は静的検証と呼ばれる「プログラムを実行せずにバグを発見する」手法だが、大規模なソフトウェアへの適用はまだまだ難しい。一方、テストや実行時監視といった「プログラムを実行してバグを発見する」動的検査手法はバグがないことを保証することは難しいが形式手法に比べて敷居は低い。

当分野では、お互いの長所を生かすために、言語レベルで静的検証と動的検査を融合させるための研究を行っている。特に、(1)漸進的型付けと呼ばれる、ひとつの言語に静的・動的型検査を混在させるための手法や、(2)ソフトウェア契約という、テストプログラムを命題として使いプログラムの仕様記述・静的検証、動的検査に用いるための手法の研究を行っている。

新しいプログラミングのための型理論

型理論は，プログラム中のデータ型の整合性を実行前に検査し誤りを検知するための枠組みである．型理論を新しいプログラミング機構に応用する研究を行っている．具体的には(1)プログラムをデータとして扱うメタプログラミングの安全性を保証するための様相論理を応用した型理論や(2)量子計算のためのプログラミングへの応用などを研究している．

深層学習モデルの解釈可能性向上

深層学習を用いるソフトウェアが多く開発されているが，そのようなソフトウェアのデバッグや検証においては，深層学習モデルの動作が人間に解釈できないことが問題となることが多い．この問題に取り組むために，深層学習の解釈性向上手法の研究に取り組んでいる．特に深層学習モデルをブラックボックスとして外部から観察できる入出力の関係だけから，その出力を説明する手法を研究している．

コンピュータソフトウェア分野

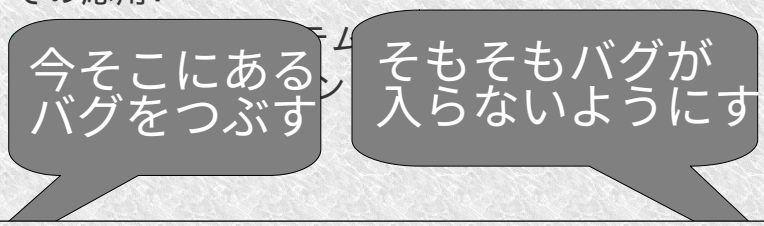
通信情報システム専攻
五十嵐 淳
末永 幸平
和賀 正樹

研究テーマ

プログラム・プログラミング言語の基礎理論

- プログラムの数学・論理学
- 言語処理系の実装技術

その応用:



プログラムの基礎理論

これは何をするプログラムでしょう？

```
while (x != y) {
  if (x > y) { x = x - y; }
  else { y = y - x; }
}
print(x);
```

どうやって確かめる？

- アルゴリズムの教科書を見る
- 入力を何通りか試す（テスト）
- 証明する!
  - 論理の力を使った全数テスト

証明する!

- まず、以下が同値なことに注意
  - 「相異なる自然数 x, y の GCD が g である」
  - 「互いに素な自然数 a, b が存在し x = ag, y = bg」
- while の中を一度実行すると x = (a-b)g, y = bg になる(x > y の場合)
  - a-b と b は互いに素なので、新しい x, y の GCD は 引き続き g
  - x < y の場合も同様
- ループを抜けるのは x = y の時で GCD は x

プログラムの基礎理論とその応用

- プログラムの「意味」とは何か？
- プログラムの挙動に関する論理
- プログラム検証 （応用）
  - (予め決められた)ある種のバグがないこと
  - 実行結果が期待したものであること

  をプログラム実行前に自動で確かめる
- プログラミングのための新しい機能
  - 意味がわかりやすく、挙動が推論しやすく

プログラム検証に関するテーマ

- ポインタ操作を伴うプログラムの検証
  - C, Java, Rust など(大抵のプログラム？)
- プログラムインバリアントの自動発見
- ブロックチェーン実装(in OCaml)の検証
- スマートコントラクトの検証
- 物理情報システム(CPS)の形式検証

物理世界を情報技術で制御するシステム(自動車、ロボット)

ブロックチェーン上の決済を自動化するプログラム

プログラム検証に関するテーマ

- 物理情報システムのための軽量形式手法
  - ブラックボックス検査
  - オートマトン、様相論理を応用したシステムモニタリング技法
- 強化学習を用いた検証ヒューリスティクスの高速化

広い意味でシステム安全性に関わるテーマ

- 深層学習モデルの解釈可能性向上手法

新しい プログラミング機構

今のプログラミング言語で十分？

- Q: なぜ新しい言語が続々と現れるのか？

応用分野に応じた適切な抽象化・機能

め

- 数値
- 記号
- データ

計算機・人間両方にとって見通しのよいプログラム

- システムプログラミングのための C
- ブラウザのための JavaScript
- 統計処理のための R

...

適切な抽象化

- 制御の抽象化
  - goto からサブルーチン、関数へ
- データの抽象化
  - レコードから抽象データ型、オブジェクトへ
- 型による抽象化
  - 多相性、ジェネリクス
- 大規模プログラミングのための抽象化
  - モジュールシステム
- 言語による抽象化
  - Domain-Specific Languages

未来のプログラミング言語のために

主な研究テーマ

- 論理と計算の関係を理解する
  - 「証明=プログラム」の追求
- 静的検証と動的検査の統合
  - 漸進的型付け
  - ソフトウェア契約
- メタプログラミング
- 量子計算

うまくいけばこんなことだって…

Java 言語仕様第三版 Preface と索引より

The theoretical basis for the core of the generic type system owes a great debt to the expertise of Martin Odersky and Phil Wadler. Later, the system was extended with wildcards. These were based on the work of Atsushi Igarashi and Mirko Viroli, which itself built on earlier work by Kresten Thorup and Mads Torgersen. Wildcards were initially designed and implemented as part of a collaboration between Sun and Aarhus University. Neal Gafter and myself participated on Sun's behalf, and Erik Ernst and Mads Torgersen, together with Peter von der Ahé and Christian Plesner-Hansen, represented Aarhus. Thanks to Ole Lehrmann-Madsen for enabling and supporting that work.

研究室の特色

研究室の特色

- 研究テーマの幅広さ
  - 教員から提案はあっても「割り当て」はない
  - 学生からの新規提案も歓迎
- 海外も含め活発な共同研究
  - 東大、筑波大、千葉大、国立情報学研究所
  - 三菱電機、ダイラムダ、センスタイムジャパン、LegalForce、PatentField
  - エジンバラ大(英)、リスボン大(葡)、フライブルク大(独)、CNRS(仏)、(ポツダム大学（独）、ペンシルバニア大学（米）)

楽ではないかもしれないけど
楽しい研究室！

を目指し日夜精進中...

こんな学生を歓迎します

- 論理学に興味がある
  - パズルを解いたり頭を動かすのは好き
  - 物事を厳密に捉えるのが好き
- プログラミング言語に興味がある
  - 今のプログラミング言語に不満がある
  - プログラムがどう動いているか興味がある
  - プログラムを(じっくり)書くのが好き
- やる気はだれにも負けない
- 世界を舞台に活躍したい

構成員一同

みなさんの挑戦をお待ちしております

おまけ

チャレンジ: MU パズル

ホフスタッター(著)
「ゲーデル、エッシャー、バッハ」より

- M,I,U からなる文字列の書き換え
- 許されている操作
  1. Iで終わる文字列の最後へのUの付加
  2. Mで始まる文字列のMより後の部分の複製
  3. (部分文字列として現れる)IIIをUで置換
  4. (部分文字列として現れる)UUの除去
- 操作1～4はどの順番に何度使ってもよい

書き換えの例

MI	(操作2： 複製)
→ MII	(操作2： 複製)
→ MIIII	(操作1： Uの付加)
→ MIIIIU	(操作2: 複製)
→ MIIIIUIIIIU	(操作3： III → U)
→ MIIIIUUIU	(操作4: UUの除去)
→ MIIIIIU	

問題：
MIから始めMUまで書き換えられるか？

- M,I,U からなる文字列の書き換え
- 許されている操作
  1. Iで終わる文字列の最後へのUの付加
  2. Mで始まる文字列のMより後の部分の複製
  3. (部分文字列として現れる)IIIをUで置換
  4. (部分文字列として現れる)UUの除去
- 操作1～4はどの順番に何度使ってもよい