研究室紹介 青戸研究室

ソフトウェアの世界

- イノベーションの加速要因の1つは，ソフトウェア
  - 情報ネットワークや物流やインフラが，神経や血流や体とすると，ソフトウェアは頭脳
- アイデア(本質)とその実現の間の大きなギャップの縮小
  - 物理的な構築プロセスではなく，単なる記述(プログラム)で実現可能
  - 複雑な処理が高速に実現し，修正や変更も容易
- より本質的なところに，より多くの脳力と時間を割ける

  $\Rightarrow$ 生産性の向上

ソフトウェアの信頼性


すべて　ニュース　レビュー

Google、Android向けパッチを開発、Linuxの未解決の脆弱性に対応
Googleの担当者は「影響を受けるAndroidデバイスは当初の報告よりもずっと少ないはず」との見方を示す。(2016/1/22)

Intelのドライバ更新ツールに脆弱性、SSL接続使用せず
Intel Driver Update Utilityでは更新ダウンロード用のURLにSSLが使われていなかった。(2016/1/21)

Apple、iOSとOS X、Safariの更新版を公開
「OS X El Capitan 10.11.3」と「セキュリティアップデート2016-001」、「iOS 9.2.1」、「Safari 9.0.3」が公開され、多数の深刻な脆弱性が修正された。(2016/1/21)

「BIND 9」の修正版公開、広範囲に影響する脆弱性も
DNSサーバソフト「BIND 9」で2件の脆弱性報告があり、1件は広範囲に影響する深刻なものという。(2016/1/20)

Oracleが定例セキュリティパッチを公開、深刻な脆弱性を多数修正
DatabaseやFusion Middleware、Javaなどの脆弱性に対処した。(2016/1/20)

Apple、iOSとOS Xのセキュリティアップデートを公開
iOS 9.2.1では13件、OS Xでは合計9件の脆弱性を解決。悪用された場合、任意のコード実行や権限昇格などの被害が予想される。(2016/1/20)

Linuxカーネルに脆弱性　PCやサーバ、Androidの大多数に影響
Linuxカーネルに2012年から存在していた脆弱性をセキュリティ企業が発見。Linux搭載のPCやサーバ数千万台と、Androidデバイスの66%が影響を受けるという。(2016/1/20)

バッテリー残量表示問題修正はまだ:
「iOS 9.2.1」配信開始――13件の脆弱性対処やバグ修正など
今年初のiOSアップデートはバグ修正とセキュリティアップデートが中心。このアップデートでは、iPhone 6s／6s Plusのバッテリー残量表示問題の修正はないが、修正された脆弱性は13件に上る。(2016/1/20)

Ciscoのアクセスポイントに固定パスワードの脆弱性
インストールの過程で作成されるデフォルトのユーザーアカウントに固定パスワードが使用されていた問題に対処。他にも多数の製品に見つかった深刻な脆弱性が修正された。(2016/1/15)

OpenSSHに脆弱性、秘密鍵流出の恐れ
悪質なサーバを使って悪用された場合、クライアントユーザーの秘密鍵を含むクライアントメモリが流出する恐れがある。(2016/1/15)

Adobe、AcrobatとReaderの脆弱性を修正
AcrobatとReaderの更新版では17件の脆弱性に対処した。悪用された場合、任意のコードを実行され、システムを制御される恐れがある。(2016/1/13)

Microsoft、9件の月例セキュリティ情報を公開　古いIEは最後の

- 多くのソフトウェアに，人為的なミス(バグ)が含まれる.
- 日々，ソフトウェアの不具合の報告がある.
- セキュリティやサービス停止など，甚大な被害を含むものも多数.

どうやったらソフトウェアの信頼性を高めることが出来るか?

なぜ，正しさと真面目に取り組む必要があるのか

- プログラマは，なぜ自分のプログラムが正しく動くと考えるか?
  - アルゴリズムが正しい.
  - 私はプログラムの正確な動きを知っている.
  - 実際，いくつかテストして上手く動いている.
  - だから，プログラムが正しくないわけがない.

なぜ，正しさと真面目に取り組む必要があるのか

- プログラマは，なぜ自分のプログラムが正しく動くと考えるか?
  - アルゴリズムが正しい.
  - 私はプログラムの正確な動きを知っている.
  - 実際，いくつかテストして上手く動いている.
  - だから，プログラムが正しくないわけがない.

実際は，プログラマの幻想．間違いである場合も多い.

なぜ，正しさと真面目に取り組む必要があるのか

- プログラマは，なぜ自分のプログラムが正しく動くと考えるか?
  - アルゴリズムが正しい.
  - 私はプログラムの正確な動きを知っている.
  - 実際，いくつかテストして上手く動いている.
  - だから，プログラムが正しくないわけがない.

  実際は，プログラマの幻想．間違いである場合も多い.
- なぜ間違えるのか?
  実は，人の直観に頼った推論は非常に誤りやすい.

なぜ，正しさと真面目に取り組む必要があるのか

- プログラマは，なぜ自分のプログラムが正しく動くと考えるか?
  - アルゴリズムが正しい.
  - 私はプログラムの正確な動きを知っている.
  - 実際，いくつかテストして上手く動いている.
  - だから，プログラムが正しくないわけがない.

  実際は，プログラマの幻想．間違いである場合も多い.
- なぜ間違えるのか?
  実は，人の直観に頼った推論は非常に誤りやすい.
- 本当?
  数学の歴史に学ぶと，その通りとしか考えられない.
  2千年以上の数学の歴史において，その不確かさが問題になったのは，ようやく19世紀後半になってから.
  ～19世紀後半の数学の世界の大きな変革～
  直観に頼った数学 $\Longrightarrow$ 数理論理学に基礎付けられた数学

Logic in Computer Science 計算機科学における柱の*1*つ

計算機科学(コンピュータサイエンス)は,

数理論理学をおおいに取り込むべきである

Logic in Computer Science 計算機科学における柱の1つ

計算機科学(コンピュータサイエンス)は，
数理論理学をおおいに取り込むべきである

計算機科学における，数理論理学の手法に基く理論の数々

- 計算をどのように捉えるか?
- プログラムの意味はどう捉えられるのか?
- プログラムの性質をどう示すのか?
- 計算における，非決定性および並列性.
- 計算対象の型付けをどのように利用するか.

Logic in Computer Science 計算機科学における柱の1つ

計算機科学(コンピュータサイエンス)は，

数理論理学をおおいに取り込むべきである

計算機科学における，数理論理学の手法に基く理論の数々

- 計算をどのように捉えるか?
- プログラムの意味はどう捉えられるのか?
- プログラムの性質をどう示すのか?
- 計算における，非決定性および並列性.
- 計算対象の型付けをどのように利用するか.

$\Rightarrow$ ソフトウェア基礎理論 とよばれている

ソフトウェアの信頼性に大きく貢献

本研究室の研究内容

計算モデル(項書き換えシステム)にもとづく
ソフトウェア基礎理論と定理自動証明

本研究室の研究内容

計算モデル(項書き換えシステム)にもとづく ソフトウェア基礎理論と定理自動証明

ソフトウェアにおける，証明と計算は融合されるべし

- 個々のプログラミング言語ではなく，計算モデルを用いて，一般的な理論と技術の構築を目指す.
- コンピュータで自動化された，ソフトウェアや仕様の検証法や証明法を構築する.

具体的には?

以下のようなトピックスについて研究を行っている：

- 計算・証明融合システムの基礎理論
  - 第一階項書き換えシステム(TRS), ラムダ計算，条件付き項書き換えシステム，名目書き換えシステム
  - これらの計算モデルのさまざまな性質の検証技術
- 書き換え理論に基づく定理自動証明
  - 等式定理の自動証明，帰納法による定理の自動証明，補題発見，検証ツールや定理自動証明システムの開発
- プログラムの自動生成・変換・検証
  - 計算モデルにもとづくプログラム変換法の形式化，プログラムと計算モデル間の変換法，プログラム意味論に基づく正しさ

項書き換えシステムによる証明と計算

- 項書き換えは計算モデルの1つ
- 項書き換えシステム (TRS)は方向付けされた等式の集合
- TRSは2つの側面をもつ：
  - 等式として見ると，論理(等式推論)
  - 書き換え規則として見ると， 計算手順(プログラム)
- 推論と計算を組み合わせた柔軟な操作
  $\Rightarrow$ ソフトウェア検証法原理や定理自動証明法の構築

自然数：$0, 1, 2, \ldots = 0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$

$$\mathcal{R}\left\{\begin{array}{lcl} \mathsf{sum}([\,]) & \to & 0 \\ \mathsf{sum}(x : y) & \to & +(x, \mathsf{sum}(y)) \\ +(0, y) & \to & y \\ +(\mathsf{s}(x), y) & \to & \mathsf{s}(+(x, y)) \end{array}\right.$$

$$\begin{array}{ll} & \underline{\mathsf{sum}(\mathsf{s}(0) : \mathsf{s}(0) : [\,])} \\ \to_{\mathcal{R}} & +(\mathsf{s}(0), \underline{\mathsf{sum}(\mathsf{s}(0) : [\,])}) \\ \to_{\mathcal{R}} & +(\mathsf{s}(0), +(\mathsf{s}(0), \underline{\mathsf{sum}([\,])})) \\ \to_{\mathcal{R}} & \underline{+(\mathsf{s}(0), +(\mathsf{s}(0), 0))} \\ \overset{*}{\to}_{\mathcal{R}} & \mathsf{s}(\mathsf{s}(0)) \end{array}$$

等式推論と書き換えによる証明(1)

$$\mathcal{E}\left\{\begin{array}{lcll} +(0,x) & \approx & x & (1)\\ +(-(x),x) & \approx & 0 & (2)\\ +(+(x,y),z) & \approx & +(x,+(y,z)) & (3)\end{array}\right.$$

$-(0) \approx_{\mathcal{E}} 0$の等式推論による証明(発見的，自動化困難)

$$\begin{aligned} -(0) &\overset{(1)}{\approx}_{\mathcal{E}} +(0,-(0)) \overset{(2)}{\approx}_{\mathcal{E}} +(+(-(-(-(0))),-(-(0))),-(0))\\ &\overset{(3)}{\approx}_{\mathcal{E}} +(-(-(-(0))),+(-(-(0)),-(0)))\\ &\overset{(2)}{\approx}_{\mathcal{E}} +(-(-(-(0))),0) \overset{(1)}{\approx}_{\mathcal{E}} +(-(-(-(0))),+(0,0))\\ &\overset{(2)}{\approx}_{\mathcal{E}} +(-(-(-(0))),+(+(-(-(0)),-(0)),0))\\ &\overset{(3)}{\approx}_{\mathcal{E}} +(-(-(-(0))),+(-(-(0)),+(-(0),0)))\\ &\overset{(2)}{\approx}_{\mathcal{E}} +(-(-(-(0))),+(-(-(0)),0))\\ &\overset{(3)}{\approx}_{\mathcal{E}} +(+(-(-(-(0))),-(-(0))),0) \overset{(2)}{\approx}_{\mathcal{E}} +(0,0) \overset{(1)}{\approx}_{\mathcal{E}} 0 \end{aligned}$$

等式推論と書き換えによる証明(2)

$$
\mathcal{R}\left\{\begin{array}{lcl}
-(0) & \to & 0 \\
+(0,x) & \to & x \\
+(x,0) & \to & x \\
-(-(x)) & \to & x \\
+(-(x),x) & \to & 0 \\
+(x,-(x)) & \to & 0 \\
-(+(x,y)) & \to & +(-(y),-(x)) \\
+(+(x,y),z) & \to & +(x,+(y,z)) \\
+(-(x),+(x,y)) & \to & y \\
+(x,+(-(x),y)) & \to & y
\end{array}\right.
$$

等式推論と書き換えによる証明(2)

$$\mathcal{R}\left\{\begin{array}{lcl} -(0) & \to & 0 \\ +(0,x) & \to & x \\ +(x,0) & \to & x \\ -(-(x)) & \to & x \\ +(-(x),x) & \to & 0 \\ +(x,-(x)) & \to & 0 \\ -(+(x,y)) & \to & +(-(y),-(x)) \\ +(+(x,y),z) & \to & +(x,+(y,z)) \\ +(-(x),+(x,y)) & \to & y \\ +(x,+(-(x),y)) & \to & y \end{array}\right.$$

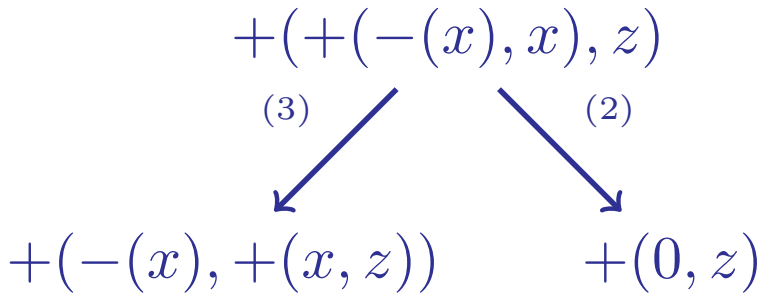
$-(0) \approx_{\mathcal{E}} 0$の書き換えによる証明(自動化が容易)

$$-(0) \to_{\mathcal{R}} 0$$

完備化のアイデア：等式推論から書き換えへ

$$\mathcal{E}\left\{\begin{array}{lcll} +(0,x) & \approx & x & (1)\\ +(-(x),x) & \approx & 0 & (2)\\ +(+(x,y),z) & \approx & +(x,+(y,z)) & (3)\end{array}\right.$$

$$\Rightarrow \quad \mathcal{R}_0\left\{\begin{array}{lcll} +(0,x) & \to & x & (1)\\ +(-(x),x) & \to & 0 & (2)\\ +(+(x,y),z) & \to & +(x,+(y,z)) & (3)\end{array}\right.$$

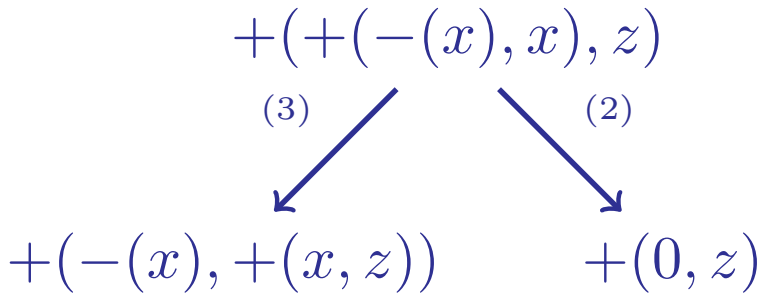


等式の向きを限定すると
証明能力は弱くなってしまう

完備化のアイデア：等式推論から書き換えへ

$$\mathcal{E}\left\{\begin{array}{lcll} +(0,x) & \approx & x & (1)\\ +(-(x),x) & \approx & 0 & (2)\\ +(+(x,y),z) & \approx & +(x,+(y,z)) & (3)\end{array}\right.$$

$$\Rightarrow \quad \mathcal{R}_0\left\{\begin{array}{lcll} +(0,x) & \to & x & (1)\\ +(-(x),x) & \to & 0 & (2)\\ +(+(x,y),z) & \to & +(x,+(y,z)) & (3)\end{array}\right.$$

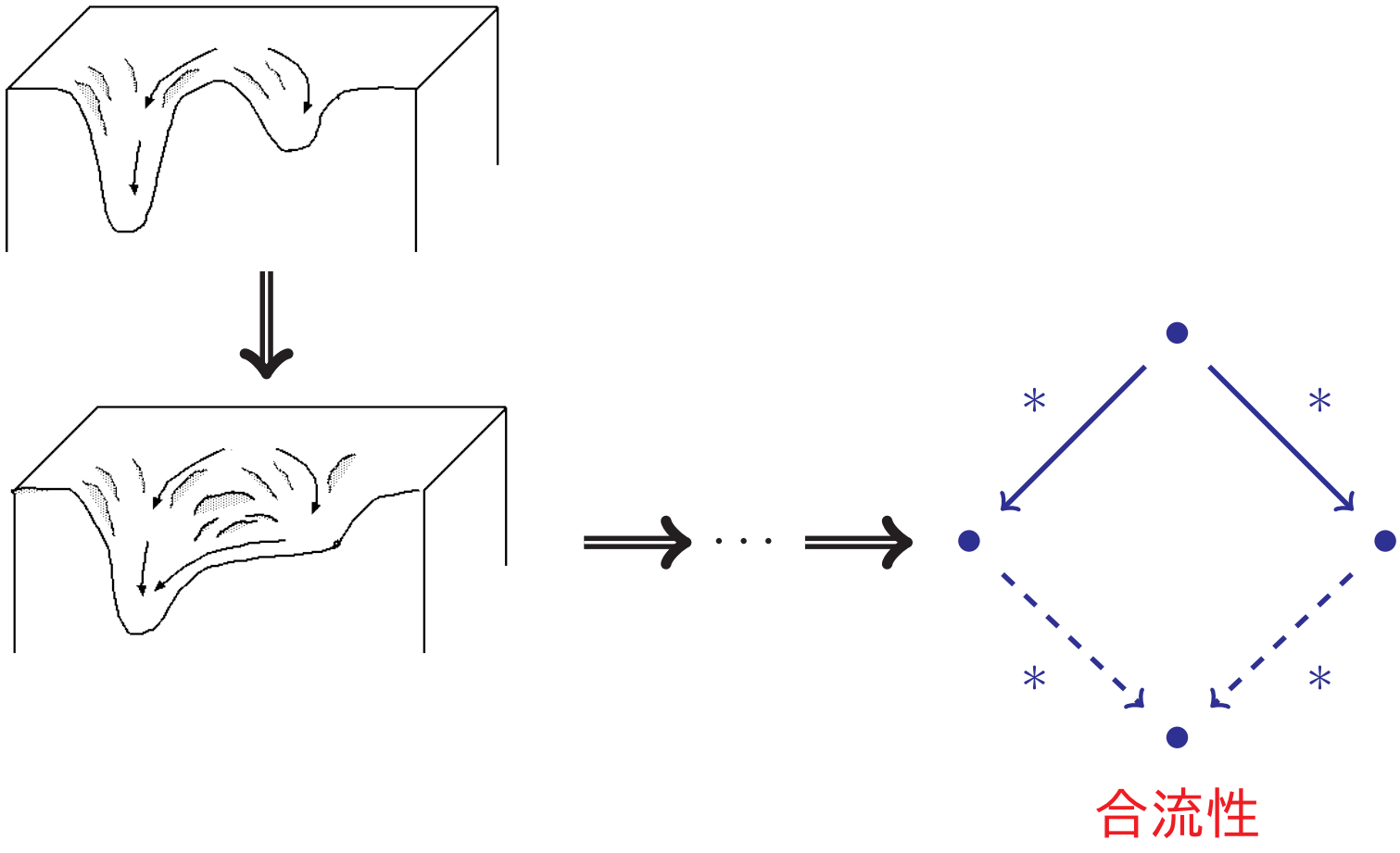


等式の向きを限定すると
証明能力は弱くなってしまう

$$\Rightarrow \quad \mathcal{R}_1\left\{\begin{array}{lcll} +(0,x) & \to & x & (1)\\ +(-(x),x) & \to & 0 & (2)\\ +(+(x,y),z) & \to & +(x,+(y,z)) & (3)\\ +(-(x),+(x,z)) & \to & +(0,z) & (4)\end{array}\right.$$

書き換え規則を追加することで証明能力を回復

完備化から合流性へ

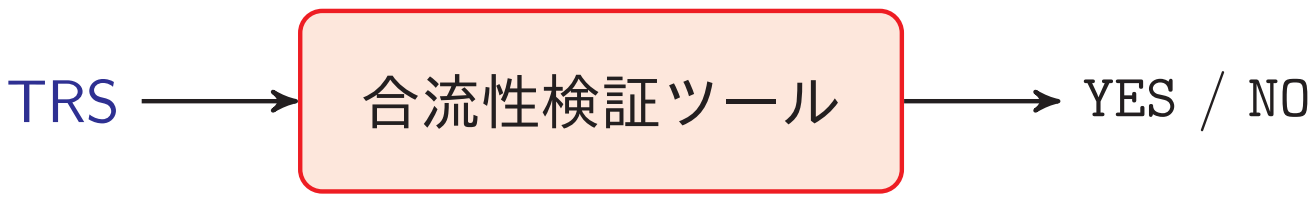


項書き換えシステムの理論や検証技術で，合流性の有無は本質的な違いを生む要因 $\Longrightarrow$ さまざまな合流性検証法が研究

合流性検証ツール

109.trs

```
1  (VAR x y z)
2  (RULES
3     join(x,meet(x,y)) -> x
4     meet(x,join(y,z)) -> join(meet(x,y),meet(x,z))
5     meet(x,x) -> x
6     join(x,x) -> x
7     meet(meet(x,y),z) -> meet(x,meet(y,z))
8     meet(x,y) -> meet(y,x)
9     join(join(x,y),z) -> join(x,join(y,z))
10    join(x,y) -> join(y,x)
11 )
12 (COMMENT Theory of Distributive Lattice from [PS81])
13
```



- 合流性検証システムACPの開発 (2007年〜)

合流性検証ツール競技会への参加

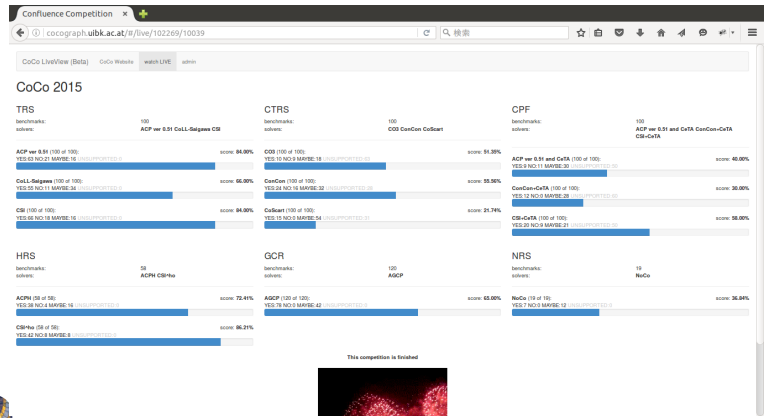
FLoC Olympic Games 2014

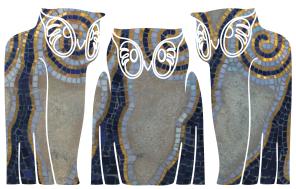
Citius, Maius, Potentius — Faster, Bigger, More Powerful

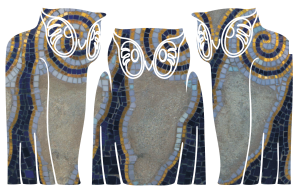
Takahito Aoto Nao Hirokawa Harald Zankl

Tohoku University, Japan JAIST, Japan University of Innsbruck, Austria

Confluence Competition 2014







- CoCo 2014は14競技会の1つとして，FLoCオリンピックゲームへ参加. (FLoCは *Logic in Computer Science* 分野の主だった国際会議の共同開催会議.)
- 近年の成績は CoCo Webpage で：

`http://project-coco.uibk.ac.at/`

項書き換えシステムの仕様検証

例：加算の規則

$$\mathcal{E} = \left\{ \begin{array}{lcl} \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array} \right\}$$

このとき，$\mathcal{E} \models \mathsf{plus}(\mathsf{plus}(x, y), z) \doteq \mathsf{plus}(x, \mathsf{plus}(y, z))$?

項書き換えシステムの仕様検証

例：加算の規則

$$\mathcal{E} = \left\{ \begin{array}{lcl} \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array} \right\}$$

このとき，$\mathcal{E} \models \mathsf{plus}(\mathsf{plus}(x, y), z) \doteq \mathsf{plus}(x, \mathsf{plus}(y, z))$ ？

反例：$\mathcal{A} = \langle \mathbb{Z}, \mathsf{plus}^{\mathcal{A}}, \mathsf{s}^{\mathcal{A}}, 0^{\mathcal{A}} \rangle$を，　$0^{\mathcal{A}} = 0$ $\mathsf{s}^{\mathcal{A}}(x) = x$ $\mathsf{plus}^{\mathcal{A}}(x, y) = y - x$ より定める．このとき，

$$\begin{array}{l} [\![\mathsf{plus}(0, y)]\!]_v = v(y) = [\![y]\!]_v \\ [\![\mathsf{plus}(\mathsf{s}(x), y)]\!]_v = v(y) - v(x) = [\![\mathsf{s}(\mathsf{plus}(x, y))]\!]_v \end{array}$$

なので公理は満たすが，

$$\begin{array}{rl} & [\![\mathsf{plus}(\mathsf{plus}(x, y), z)]\!]_{\{x \mapsto 1, y \mapsto 1, z \mapsto 1\}} \\ = & 1 - (1 - 1) \\ \neq & (1 - 1) - 1 \\ = & [\![\mathsf{plus}(x, \mathsf{plus}(y, z))]\!]_{\{x \mapsto 1, y \mapsto 1, z \mapsto 1\}} \end{array}$$

帰納的定理 帰納法によって証明される性質

例：加算の規則

$$\mathcal{E} = \left\{ \begin{array}{lcl} \mathsf{plus}(0, y) & \to & y \\ \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \end{array} \right\}$$

$\mathsf{plus}(\mathsf{plus}(x, y), z) \doteq \mathsf{plus}(x, \mathsf{plus}(y, z))$ は帰納的定理.

$\Rightarrow$ 等式論理の定理証明では証明不可

(人の手による)帰納法の証明

(証明) 自然数xに関する帰納法.

- $x = 0$のとき.

$$\begin{array}{lll} \mathsf{plus}(\mathsf{plus}(0, y), z) & \leftrightarrow_{\mathcal{E}} & \mathsf{plus}(y, z) \\ & \leftrightarrow_{\mathcal{E}} & \mathsf{plus}(0, \mathsf{plus}(y, z)) \end{array}$$

- $x = \mathsf{s}(x')$のとき.

$$\begin{array}{lll} \mathsf{plus}(\mathsf{plus}(\mathsf{s}(x'), y), z) & \leftrightarrow_{\mathcal{E}} & \mathsf{plus}(\mathsf{s}(\mathsf{plus}(x', y)), z) \\ & \leftrightarrow_{\mathcal{E}} & \mathsf{s}(\mathsf{plus}(\mathsf{plus}(x', y), z)) \\ & \leftrightarrow_{I.H.} & \mathsf{s}(\mathsf{plus}(x', \mathsf{plus}(y, z))) \\ & \leftrightarrow_{\mathcal{E}} & \mathsf{plus}(\mathsf{s}(x'), \mathsf{plus}(y, z)) \end{array}$$

□

$\Rightarrow$ コンピュータに適した帰納法の証明は?

$\Rightarrow$ (等式論理の定理証明 + 帰納法)の自動証明法の開発

「帰納的定理の自動証明」

その他の主要な研究内容

高階項書き換えシステム
: 変数束縛を含む記述力の高い計算体系の検証法

条件付き項書き換えシステム
: 条件再帰を含む記述力の高い計算体系の検証法

項書き換えシステムの性質の検証
: 一意正規形性，可換性，階層合流性，etc.

プログラム変換 プログラム変換法とその正しさ

到達可能性判定 ある計算戦略が解を得るために十分かの判定

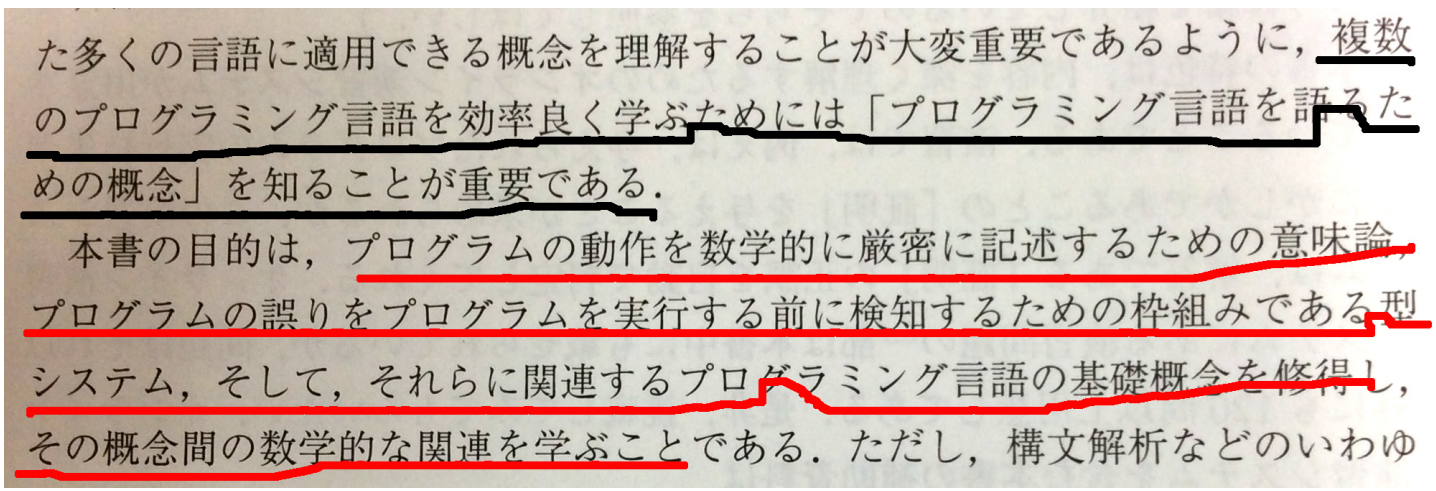
計算量検証 プログラムの効率性の自動検証

最近の論文や研究に関する情報は研究室のホームページで：

http://www.nue.ie.niigata-u.ac.jp/

研究室体験(3年生後期)

- (専門度がある程度広い)日本語の専門書を輪講します.
- これまで輪講した本(の例)：
  - 五十嵐淳，プログラミング言語の基礎概念，サイエンス社，2011年.
  - 大堀淳，プログラミング言語の基礎理論，共立出版，1997年.
  - 小林直樹，住井英二郎，プログラム意味論の基礎，サイエンス社，2020年.



た多くの言語に適用できる概念を理解することが大変重要であるように，複数のプログラミング言語を効率良く学ぶためには「プログラミング言語を語るための概念」を知ることが重要である.

本書の目的は，プログラムの動作を数学的に厳密に記述するための意味論，プログラムの誤りをプログラムを実行する前に検知するための枠組みである型システム，そして，それらに関連するプログラミング言語の基礎概念を修得し，その概念間の数学的な関連を学ぶことである．ただし，構文解析などのいわゆ

研究室のゼミ

- 週3回程度のゼミ，コアタイムはありません
- 1年間かけて専門書を1冊輪講(全体ゼミ)

どんな分野を勉強するにしても，その分野の基本的な専門書をじっくり勉強することが必要です．そのための力を養うゼミです．専門書は，理論計算機科学やソフトウェア基礎理論分野から幅広く吟味して，力の付きそうな本を選びます．

これまで輪講した本の内容
計算可能性の理論，数理論理学，オートマトン，関数型プログラム，型理論，プログラム意味論，均し計算量，第一階理論の決定手続き，XML，自動定理証明，並列計算

毎年，異なる本を輪講します．

$\Longrightarrow$ 研究室の輪講の状況はホームページで．

4年生のスケジュール

- 専門書の輪講(全体)
- 基礎ゼミ(前期)
  - 計算機環境の構築(Linuxのインストールや設定など)
  - 関数型プログラミング言語SMLの学習
  - 定理自動証明システムの実装
- 論文ゼミ(前期)
  - 項書き換えシステムの基本的な論文の勉強
  - 論文の読み方を身につけます

  理論能力・プログラミング能力を鍛錬

- 卒業研究(後期)

  プログラムなどが早くに完成して前期から卒業研究を始める人もいますが，多くの場合は後期から卒業研究に取り組みます.

卒業研究(後期)

- ～10月: 研究テーマの相談，研究プロポーザル作成
  指導教員の方からも，いろいろテーマ案を出しますが，学生それぞれの希望を聞きつつ，テーマを相談します.
- ～12月: 論文紹介
  各自の卒業研究に必要な論文を勉強して，研究室のゼミで論文紹介します.
- ～2月: 卒論執筆/卒研発表

過去の卒業論文タイトルは研究室ホームページで

卒業研究は，項書き換えや近い分野のトピックでしたら教員からいろいろ提案やアドバイスすることが出来ると思います．それ以外でも，学科内でこの研究室の守備範囲に入るトピックでしたら，できる範囲で協力・支援していきたいと思います.

研究分野のバックグラウンド

研究分野のバックグラウンドとなっている専門分野：

- 計算モデル
- プログラム意味論
- 数理論理学
- オートマトンと形式言語，計算論
- プログラミング言語とその基礎理論
- 離散数学，代数

- 知能情報システムプログラムに進むと，いくつかの分野は講義で学びます.
- 本研究室のゼミでは，これらの分野の素養や知識を深めます.

プログラムやプログラミングを，より深く，広い視野で俯瞰することが出来るようになると思います.

計算モデル

プログラムを一般的に扱うためのさまざまな抽象的な枠組み

項書き換えシステム，
ラムダ計算，
π計算，
プロセス計算，
組み合せ子論理，
停止性，合流性，
カリー・ハワードの対応，
型理論，型推論，
双模倣性，余帰納法

プログラム意味論

プログラムを意味をどう捉えるのか． どう解析するのか.

操作的意味論,
公理的意味論,
表示的意味論,
帰納法,
ホーア論理， 分離論理,
不動点,
抽象解釈

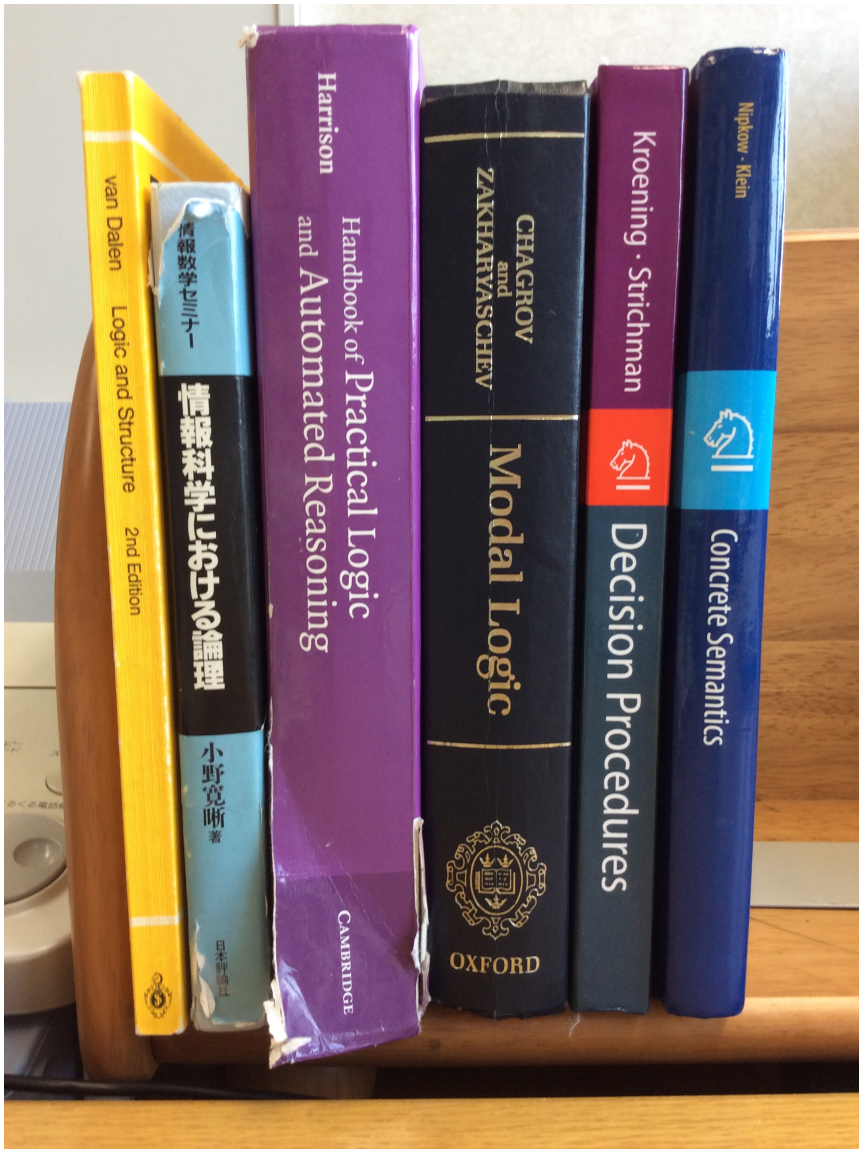
計算論

コンピュータで可能な計算とは? コンピュータでも計算できないこととは?

チューリングマシン,
計算可能性,
再帰関数,
決定可能問題,
万能チューリングマシン,
決定不能問題,
帰納的集合,
枚挙可能集合,
計算量, NP完全問題,
オートマトン,
木オートマトン

数理論理学

推論の方法とその正しさとは．推論の計算可能性と，定理および推論の検証．



様相論理，
直観主義論理，
線形論理，
シーケント計算，
自然演繹法，
完全性定理，
クリプキ意味論，
第一階述語論理，
エルブランの定理，
プレスブルガー算術，
SATソルバ，
モデル検査，
対話的定理証明器

プログラミング言語とその基礎理論

プログラミング言語に現われる概念，その計算機構をどう捉えるか.

関数型プログラミング言語，
論理型プログラミング言語，
高階関数，多相型，依存型，
先行評価，遅延評価，
名前よび，値よび，
単一化，リゾルーション，
継続，CPS変換，
代数仕様，
プログラム変換，
プログラム導出，
SML, OCaml, Haskell,
LISP, Scheme, Prolog

離散数学，代数

関連分野のさまざまな場面で用いられ，応用される数学ツール

順序，関係，整礎，
モノイド，束，
集合論，
数学的帰納法，構造帰納法，
最小不動点，最大不動点，
余帰納法，
普遍代数，
圏論

研究室で重視する能力

- 論理的な思考能力
  - 本に書いてあるから/先生が言うからではなく，自分の頭で考えることが出来るか
- 数学的な基礎能力
  - 理解するとはどういうことか
  - 証明が追えるか
  - 証明が書けるか
- プログラミングの基礎能力
  - アルゴリズムが考えられるか
  - アルゴリズムを見通せるプログラムが書けるか
- その他
  - 基本的な英語能力
  - ソフトウェアリテラシー (シェル,Linux,コンパイラ,計算機管理,...)
  - 科学的な文章の記述力
  - 研究発表の能力

大学院への進学について

卒業研究と修士研究の違い

- 求められるレベル.
- 卒業研究で求めるのは研究への取り組み．修士研究では，研究への取り組んだ結果としての成果が求められる.

主に，以下のような場で研究発表をしてもらいます．つまり，修士修了には，これらの場での研究発表に足る成果が必要になります.

- ソフトウェア科学会大会(JSSST大会)
- プログラミングおよびプログラミング言語に関するワークショップ(PPL)
- TRSミーティング
- 情報処理学会プログラミング研究会(PRO)
- 専門分野の国際会議 (FSCD,IJCAR,PPDPなど)