コンピュータサイエンス基礎
ソフトウェアと数学

長谷川 真人

京都大学数理解析研究所

２０１７年７月１３日

ソフトウェアってどんなもの？

うんと簡単な例（アセンブリと関数型プログラム）

```
sum: addi $sp, $sp, -8      L1: addi $a0, $a0, -1
     sw   $ra, 4($sp)           jal  sum
     sw   $a0, 0($sp)           add  $0, $0, $0
     slti $t0, $a0, 1           lw   $a0, 0($sp)
     beq  $t0, $0, L1           lw   $ra, 4($sp)
     add  $0, $0, $0            addi $sp, $sp, 8
     add  $v0, $0, $0           add  $v0, $a0, $v0
     addi $sp, $sp, 8           jr   $ra
     jr   $ra                   add  $0, $0, $0
     add  $0, $0, $0
```

```
let rec sum (x) =
  if x <= 0 then 0 else x + sum (x - 1)
```

どちらも整数 x に対し $1+2+\cdots+(x-1)+x$ を計算

ソフトウェアの不具合はとても厄介

現在、多くの商用・非商用ソフトウェアは、かなりの大きさになっています（数万～数百万行）。

そのようなソフトウェアは、作るのも大変だけれど、不具合（バグ）を直すのはもっと大変です。

けれども、社会や産業の主要部で用いられているソフトウェアの不具合は致命的です。

対策：
- （予防）正しく扱いやすいソフトウェアを作る技術
- （対処）ソフトウェアの不具合を検出し修正する技術

ソフトウェアはとても厄介

ソフトウェアに関する問題を解決するのは、実はとても難しいことです。

第一に、ソフトウェアに関する多くの問題は、機械的に解決する手順が原理的に存在しないことがわかっています。（照井先生の講義を参照）

第二に、理論的には解決する手順がわかっていても、そのために多大な計算資源（時間や記憶容量）を必要とする場合が多々あります。それらは、巨大なソフトウェアについては現実的ではありません。

ソフトウェアの危うい現状

現在使用されている一定規模以上のソフトウェアで、完璧な（一切不具合がない）ものは、ほぼ皆無だと考えられます。

多くのソフトウェアは、徹底的なテストを経て出荷・公開されています。が、テストが完全である保証はありません。

皆さんの携帯電話や自動車が問題なく使えているのは、実は奇跡のようなことです。

ソフトウェアを解きほぐす数学

今日は、数学を武器にしてソフトウェアに立ち向かう研究についてお話しします。特に、数学の、以下のふたつの長所にスポットを当てます。

（１）正確な定式化
数学を用いて、ソフトウェアの複雑な問題を正確に記述し、定式化することができます。

（２）柔軟な抽象化
数学を用いて、ソフトウェアの複雑な構造を適度に抽象化し、問題の本質に肉薄することができます。

（宣伝：数理解析研究所では、古くからソフトウェアの数理科学的な研究が盛んに行われてきました）

お断り

- 今回の講義は、今までの講義内容をある程度踏まえていますが、かなり独立した内容を含んでいます。
- 長谷川の私見を多く含んでいます。照井先生・星野先生の見解と一致しない可能性があります。
- 重要: 今回の講義の内容からはレポート問題には出題しません。（レポート問題は来週配布予定）
- どうか広い心で、気楽に聞いてください。

ソフトウェアについてきちんと語るために

ソフトウェアは、それ自体はただの記号の列

```
sum: addi $sp, $sp, -8   L1: addi $a0, $a0, -1
     sw   $ra, 4($sp)        jal  sum
     sw   $a0, 0($sp)        add  $0, $0, $0
     slti $t0, $a0, 1        lw   $a0, 0($sp)
     beq  $t0, $0, L1        lw   $ra, 4($sp)
     add  $0, $0, $0         addi $sp, $sp, 8
     add  $v0, $0, $0        add  $v0, $a0, $v0
     addi $sp, $sp, 8        jr   $ra
     jr   $ra                add  $0, $0, $0
     add  $0, $0, $0
```

```
let rec sum (x) =
  if x <= 0 then 0 else x + sum (x - 1)
```

これらの記号列に基づいてコンピュータが仕事をするためには、いろいろな約束を決めなくてはなりません。

プログラミング言語（復習？）

ソフトウェアを記述するために、さまざまなプログラミング言語が用いられています。プログラミング言語は、おおむね以下のふたつの要件を備えています。
（照井先生・星野先生の講義を参照）

（１）書き方を定める規則（文法）
どんな「文章」が正しいプログラムかを定めます。

（２）動かし方を定める規則（操作的意味論）
文法に従って書かれたプログラムの動作を定めます。

プログラミング言語では、書き方・動かし方を数学的に正確に定義することが（少なくとも原理的には）可能

プログラミング言語

私がよく使うプログラミング言語の文法（書き方）によれば、

```
let rec sum (x) =
  if x <= 0 then 0 else x + sum (x - 1)
```

は、正しいプログラムであることが証明できます。
そして、その操作的意味論（動かし方）によれば、このプログラムに 0 以上の整数 x を入力すると、

$$1 + 2 + \cdots + (x-1) + x = \frac{x(x+1)}{2}$$

を出力することが証明できます。

定式化≠解決

プログラミング言語やソフトウェアは、数学的に定式化できるってことですよね？
そうです。原理的には、完全に定式化できるはずです。

じゃあ、ソフトウェアの問題は全部数学的に解決可能？
いや、そうとは限らないのです。定式化＝解決、ではないのです。

例えるなら、定式化することは方程式を立てることに相当しますが、その方程式が解けるかどうかは、また別の問題なのです。

解ける問題、解けない問題（復習）

ソフトウェア（あるいは数学全般）に関する問題は、おおまかには
（１）解ける問題
（２）どうがんばっても解けない（決定不可能な）問題
に分類できます。

ここで、「解ける」というのは、「かくかくしかじかの手順でやれば必ず解ける方法がある」（計算可能）ということです。その問題を解くソフトウェアをつくることができる、と言いかえることもできます。
（照井先生の講義を参照）

どうしても解けない問題（復習）

どうがんばっても解けない問題って???
実は、ソフトウェアに関する多くの問題は、どんなにがんばっても解けないことがわかっています。

その最初の例は、１９３０年代に、チューリングが発見しました。簡単に言えば、それは、
「与えられたソフトウェアが（与えられた入力に対し）暴走するか？」
という問題です。
（照井先生の講義を参照）

解ける問題だってあります

じゃあ、数学の力でもソフトウェアの問題は解決できない？

そうとも限りません。解ける問題もたくさんあります。
また、「どうがんばっても解けない問題」であっても、部分的には「解ける」ことが多々あります。
そのような部分的な解決が、実用上有用であることも、しばしばあります。

以下では、そのような事例として、我々が取り組んでいる「ソフトウェアを解きほぐす数学」についてお話しします。

ソフトウェアはもつれやすい

ソフトウェアは、大変もつれやすいものです。
無計画にソフトウェアを手直ししたり、追加の機能を書き足したりしていると、いつの間にか、とんでもなく判読困難な悪文に。

また、たいていのプログラミング言語には、複雑な制御構造（計算の流れを記述する機能）が備わっています。制御構造を濫用すると、あっという間に難解なプログラムが出来上がります。

もつれたプログラムの例（復習）

プログラムが自身を呼び出す制御構造が、再帰です。

マッカーシーの ‘ 91 関数 ’

$$\mathsf{F}(x) \;=\; \text{if } x > 100 \text{ then } x - 10 \text{ else } \mathsf{F}(\mathsf{F}(x+11))$$

竹内郁雄の ‘ たらいまわし関数 ’

$$\begin{aligned}&\mathsf{t}(x,y,z) \;=\\ &\quad \text{if } x \le y \text{ then } y\\ &\qquad \text{else } \mathsf{t}(\mathsf{t}(x-1,y,z),\mathsf{t}(y-1,z,x),\mathsf{t}(z-1,x,y))\end{aligned}$$

（参考： 数理解析研究所の大学院入試問題，2005・2014）

ソフトウェアはもつれやすい

よくある問題：
「△△ 3.0 というソフトを△△ 3.1 という新しいヴァージョンに更新したら、ちゃんと動かなくなりました。」

あるソフトウェアを別のソフトウェアに置き換えて不具合が生じないかどうか、という問題は至るところで頻繁に発生します。

しかし、これは原理的に「どうがんばっても解けない問題」です。ふたつのソフトウェアが同じ働きをするかどうかを判定する手順は存在しません。

プログラミング言語の意味論

プログラムの意味をシステマティックに解読する理論が、プログラミング言語の意味論です。

プログラムを直接読んだり実行したりするかわりに，適切な抽象化によって得られた，より扱いやすい情報を用いて，そのふるまいを理解する理論です。

適切な抽象化は，問題の本質を浮き彫りにします。

プログラミング言語の意味論

私の個人的な「プログラミング言語の意味論」の定義：

> プログラミング言語の意味論とは，計算の実行の過程で変化しないプログラムの性質を研究する学問である

不変量の考え方

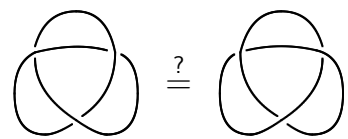
プログラムの意味を抽象化して取り出すことに深く関連するのが、数学の不変量の考え方です。

> 幾何学は与えられた変換群に属する任意の変換で不変な図形の性質を研究する学問である
> （クライン）

まえのスライドでのプログラム意味論の「定義」は、この有名な言葉のパクリです。

トポロジー（やわらかい幾何）との比較

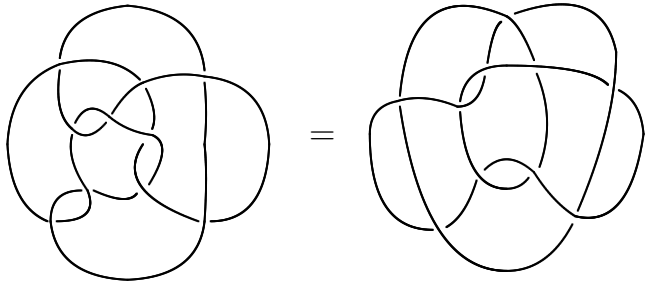
数学の問題：
与えられたふたつの結び目は同じものだろうか？



⇒ 結び目の，連続的な変形に対して不変な数学的な量を考察（結び目の不変量）

ソフトウェアの問題：
与えられたふたつのプログラムは同じ働きをするか？
⇒ プログラムの実行に関して不変な数学的な量を考察（プログラミング言語の表示的意味論）

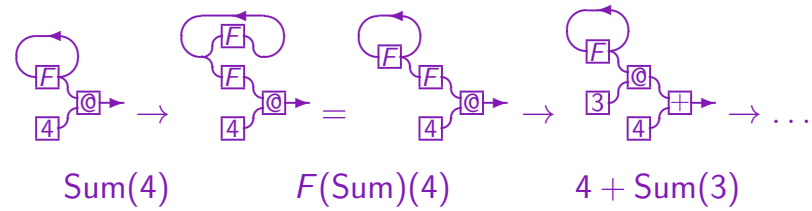
あまり自明でない結び目の例



これらが同一の結び目であることがわかるまで、
およそ８０年かかりました

抽象化（１）プログラムを絵に書いてみる

再帰プログラム
$\mathsf{Sum}(x) \;=\; \text{if } x = 0 \text{ then } 0 \text{ else } x + \mathsf{Sum}(x-1)$
のあらわす計算を図示してみると

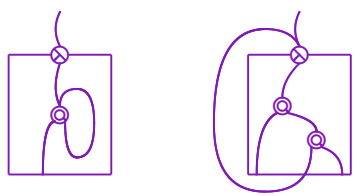


$\mathsf{Sum}(4)$ $F(\mathsf{Sum})(4)$ $4 + \mathsf{Sum}(3)$

（再帰呼び出し = 巡回的に共有された資源のコピー）
なんだか結び目と似ている？

少しこみいった例：巡回的ラムダ計算

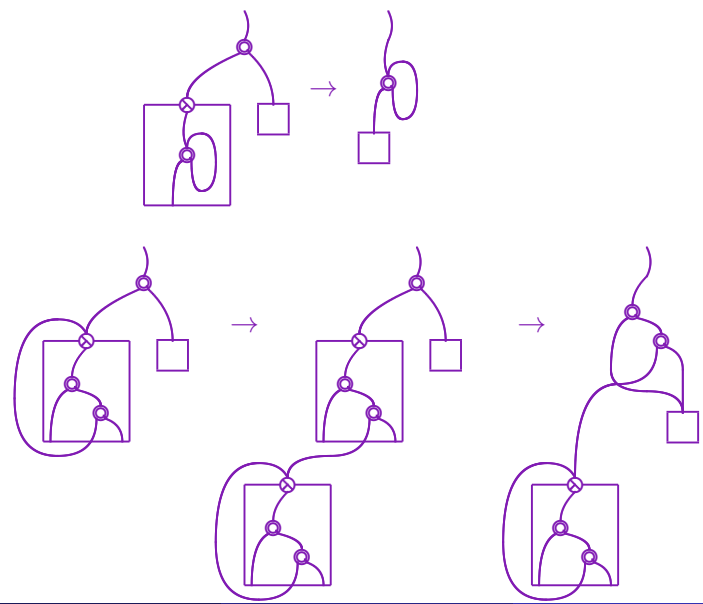
巡回的ラムダ計算における再帰演算子のふたつの実装



$\lambda f.\text{letrec } x \text{ be } fx \text{ in } x \;\stackrel{?}{=}\; \text{letrec } F \text{ be } \lambda f.f(Ff) \text{ in } F$

カリーの再帰演算子 チューリングの再帰演算子

$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ $\Theta = Y(\lambda F.\lambda f.f(Ff))$



抽象化（２）不変量を求める

図示されたプログラムの、計算の実行と連続的な変形に関する不変量には、そのプログラムの本質的な情報が詰まっているはず

でも、どうやって不変量を求めるの？

いろいろなアプローチがあります。
よく用いられる手法では、プログラムの持つ情報の量の比較から定まる順序構造（大小関係）を調べます。
（完備半順序集合など・星野先生と長谷川の講義参照）
しかし、最近になって、それが結び目の不変量と深く関連することがわかってきました。

量子不変量

かつては、結び目の不変量をシステマティックに求める方法はあまりありませんでした。

しかし、１９８０年代以降、 数理物理学的な手法により、様々な結び目の不変量が体系的に与えられるようになりました（量子不変量）。

数理解析研究所でも、量子不変量は盛んに研究されています。また、量子不変量の理論で重要な量子群は、１９８０年代に数理解析研究所（当時）の神保道夫とロシアのドリンフェルドによって独立に発見されました。

うんと簡単な不変量

行列 $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ と $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ の積は、

$$AB = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}, \quad BA = \begin{pmatrix} 23 & 34 \\ 31 & 46 \end{pmatrix}$$

対角成分の和はどちらも 69 になります。実は、どんな行列でも AB と BA の対角成分の和は一致します。

これは、行列の入出力をつなげて輪にしてしまえば、AB から作った輪も、BA から作った輪も、連続的な変形により同じになるということに対応しています。

よく似た図を見たことがあるはず

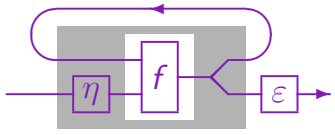
連続関数 $f\colon X \to_c Y$ および $g\colon Y \to_c X$ について、

$$fix_X(g \circ f) = g(fix_Y(f \circ g))$$

が成り立つ

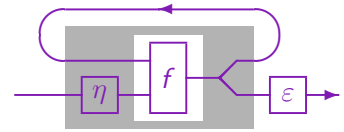
再帰プログラムの不変量

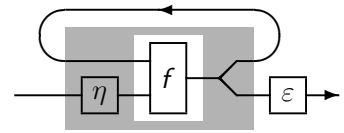
再帰プログラムについても、行列の対角成分の和をとるのと同じような操作を考えることにより、プログラムの意味を抽象化したかたちで抜き出すことができます。（完備半順序集合の場合、連続関数の最小不動点をとることが行列の対角和をとることに相当）



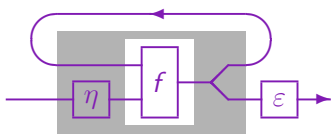
$$f^{\dagger} = \varepsilon \circ Tr^{FUX}(m^{F^{-1}} \circ F(\Delta \circ Uf \circ m^{U} \circ (\eta \times id)) \circ m^{F})$$

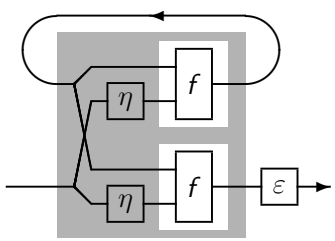
再帰プログラムの不変量 (1/9)



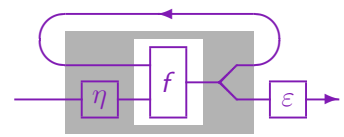


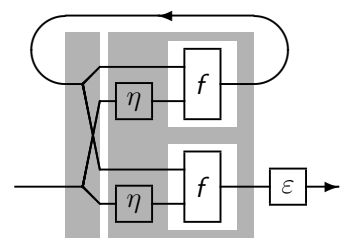
再帰プログラムの不変量 (2/9)



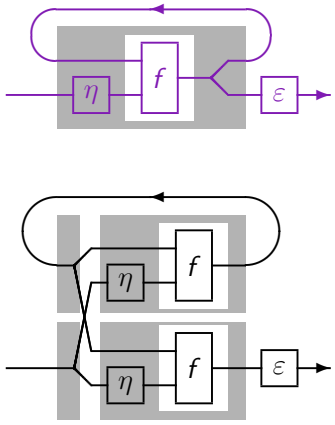


再帰プログラムの不変量 (3/9)

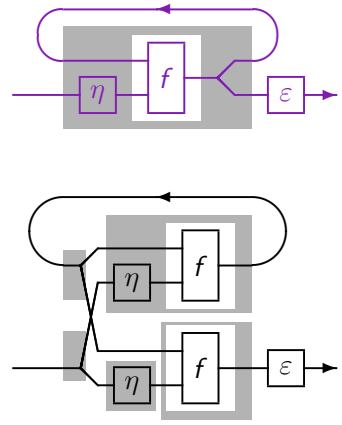




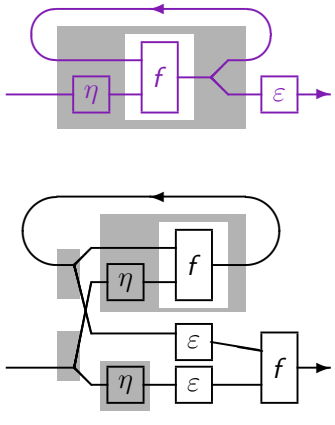
再帰プログラムの不変量 (4/9)



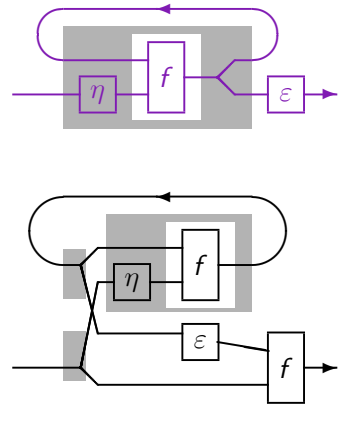
再帰プログラムの不変量 (5/9)



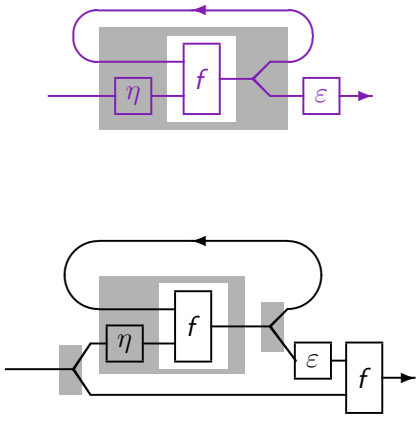
再帰プログラムの不変量 (6/9)



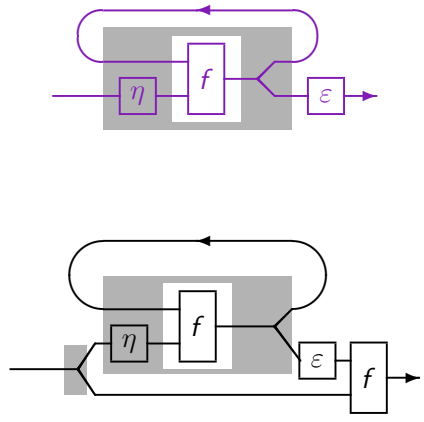
再帰プログラムの不変量 (7/9)



再帰プログラムの不変量 (8/9)



再帰プログラムの不変量 (9/9)



圏論について、少しだけ

圏（category）の定義は述べませんが、たとえば、

- 集合と関数の世界
- 完備半順序集合と連続関数の世界
- 有限次元ベクトル空間と線型写像（行列）の世界
- 量子群の表現と表現の間の適切な射の世界

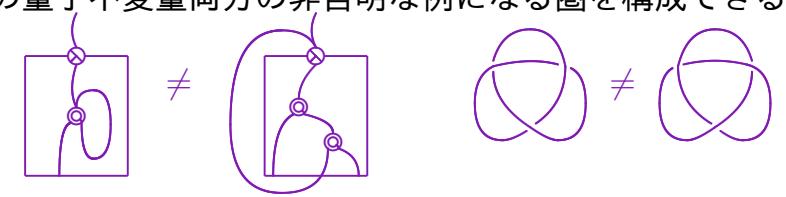
は、いずれも圏です。後の３つは、対角和（トレース）をもつ圏の例になっています。

完備半順序集合と連続関数の世界ではトレースは最小不動点から構成できます。

実は、適当な条件を満たす圏では、トレースを持つことと、不動点演算子を持つことは同値です（長谷川・Hyland）

再帰プログラムの量子不変量？

さらにがんばると、再帰プログラムの意味論と、結び目の量子不変量両方の非自明な例になる圏を構成できる



方針：プログラム意味論で用いられる圏において量子群に相当するもの（リボンホップ代数）を構成し、そのモジュールの圏を考える

７年ほど前からやっていますが、いまだ未完成の話題です
実用的な応用？(位相的量子計算？)
理論的な限界？(うまくいかないことも多い)

とりあえずのまとめ

ソフトウェアと数学の関わりの、ごく一部を紹介しました。

ソフトウェアの問題に立ち向かうために、数学の正確な定式化と柔軟な抽象化が非常に役に立つということを、認識していただければ幸いです。

そして、数学が、ソフトウェアのような社会に密接に関連した分野でも威力を発揮する、私たちの頼もしい味方であることを、大いに強調しておきたいと思います。

これからのプログラム意味論

プログラム意味論は、これまで、数理論理学と代数学を中心的な道具として発展してきた
（cf. 数理モデリングにおける微分方程式等の解析的な道具）
今後は、論理・代数に加えて、幾何学・トポロジーが重要な役割を果たすようになるのではないか。

最近の例：

- 位相的量子計算とその数学的基礎としての位相的量子場の理論（3-TQFT）（今日の話にも関連する）
- ホモトピー型理論（HTT）：データ型を「空間」、データの同値性をデータ間の「経路」の存在とみる

いずれにおいても圏論は不可避

先人の言葉から

> 解析力学の学習はありのままに世界をみない手法を身につけるための修行である
> （佐藤文隆）

近現代の物理学は，目に見える物理現象の背後に隠された本質を捉える努力を積み重ねることにより，飛躍的に発展してきた

とりあえずの結び

ソフトウェアの理論の研究は，ありのままに
ソフトウェアをみない手法を身につけるための
修行である

表層にとらわれないことにより浮き彫りになってくる
本質に目を凝らし，新しい認識への手がかりを求めて
いきたい

次回予告

- 次回は７月２０日（木)。最終回です。
- 最終回にレポート課題を配布します。
  レポートの締め切りは２週間後を予定しています。
- 最終回は教員３名が全員来ます。
  講義やレポートに関する質問を受け付けます。