# 21世紀の数式処理に対する三つの新機軸: ディペンダビリティ・ユーザビリティ・分散処理

# Towards a reliable computer algebraic system: dependability, usability and distributed processing

池上大介
Daisuke Ikegami *
産業技術総合研究所 情報技術研究部門
Information Technology Research Institute

Abstract

Computer algebraic systems (CASs) have a long history of being used for computation. One aspect of the systems which makes them powerful as a mathematical language is that it mixes efficient algorithms with computation. Other is graphical user interfaces, literally. This talk is concerned related topics; dependability, usability, and distributed computations for CASs. To describe these features, I will introduce a modern programming language, called Haskell, and show what the language enables to work with CASs briefly.

## 発表概要

この発表では、新しい数式処理を構築するという目的で、純粋関数型プログラミング言語 Haskell [4] を紹介する。Haskell は、その特徴の一つとして、 再帰的な定義を記述すると、それがそのままプログラムとなって実行することができる点にある。 これを、宣言的プログラミング(declarative programming)と呼ぶ。 従来の手続き型言語との違いとして、宣言的プログラミングが数学的記述により近いことがあげられる。

たとえば、手続き型プログラミングでは制御のためにループを用いる。しかし、 数学の証明においてループのような制御構造は通常現れない。たとえば、「x を 0 とする」と証明に書いたとき、 そのあと x が 1 から 5 までを動くとして、「つぎの命令を実行する」、 といった言明は証明の中にはあらわれない。かわりに、「x が 1 のとき、2 のとき」といった場合分けを行う。また、 任意の自然数の場合には、数学的帰納法という言明を用いて証明を行うことができる。

*ikegami.da@gmail.com

Haskell に限らず純粋関数型言語では、プログラミングにおいてループを用いない。 その代わりに、「〜のとき」というパターンマッチや、 帰納法に相当する再帰を用いてプログラムを書く。 この結果、プログラムと証明はきわめて似た形となる。

この特徴から、数式処理を作成するうえで二つの利点がうまれる。ひとつは、 証明をそのままアルゴリズムにすることができる(ときがある) ので、 手続的なアルゴリズム、特にループ、 について頭を悩ませなくてもよいという点である。 もうひとつは、逆に、作成したプログラムが数式の求める答を導出するか? について、 数学的な証明(reasoning of programs) [1]を与えることができる点にある。

このことから、数式処理を作成するうえで Haskell を選択する理由があることを発表のなかで説明する。 また、いままでの数式処理システムを抽象化したうえで、 理想的な数式処理がどのような形か、発表者の意見を述べる。

最後に、ソフトウエアに求められる基軸として、発表題名にあげた三つのことがら、 ディペンダビリティ・ユーザビリティ・分散処理について述べる。

ディペンダビリティ [3]とは、システムに対して「依存できる (dependable)」性質を分類した、ソフトウエア工学における学術用語である。 現在、数式処理ソフトウエアは本来の数式を処理するという目的だけでなく、 機械の設計や情報処理などにまで用いられるようになった。 したがって、数式処理にバグがあった場合の影響範囲はひろがったと考えられる。 ディペンダビリティの考え方は、 ソフトウエアに対し何が求められているのかを、 以下のように分類する (すべてを列挙せずに重要な五つを挙げる:[2] を参考にした):

- 可用性 (availability) 常に利用できる、利用を阻害する条件とは何か
- 信頼性 (reliability) バグがないことをどのように表明するか
- 安全性 (safety) 人間にたいする損害が発生しないことを保証できるか
- 完全性 (integrity) 不正に悪意のある変更が行われないか
- 保守性 (maintainability) 運用にあたって問題となることはなにか

このような、異なる視点からの直交した要求を満たすかどうかを調べることによって、 ソフトウエアの信頼性を高めようというのがディペンダビリティの考え方である。 この発表では、数式処理もまたソフトウエアであり、 信頼される数式処理として、ディペンダビリティの考え方も要求されることを紹介する。

ユーザビリティは、広い意味でディペンダビリティに含まれるが、 ここでは特に昨今の情報端末のユーザインターフェースが、 多様に変化していることに着目する。 キーボードとマウスと画面といった、 以前のパーソナルコンピュータのインターフェースは、 小型化されたタッチパネル式の携帯端末の存在によっておおきく様変わりした。 数式処理も、コアな部分はそのままでも、 ユーザに触れる部分の設計は考え方を 180 度転回しなければならない局面にたたされている。 もうひとつ、数式処理と数式処理同士の結合とその仕組みについても触れる。

さいごに分散処理について述べる。計算機のメニコア化、 グラフィック処理ユニットによる浮動小数点演算の活用、 ネットワークの高速化により、 従来の計算の概念はひとつのユニットが担当するアルゴリズムから、 複数のユニットがそれぞれ計算を分割して実行し、 それらを束ねるユニットが統治するという並列・並行計算へと移行している。

本発表で紹介する、プログラミング言語 Haskell はアルゴリズムの意味の数学的証明から導出される信頼性、 各ユーザインターフェースのプログラムを書くための FFI (Foreign Function Interface)、データ並列、並行 Haskell (Concurrent Haskell) という機能を持つ。このことから、近代の数式処理を作成するための道具の一つとして Haskell がふさわしいという意見を述べる。

# References


[1] Richard Bird and Oege de Moor. Algebra of Programming, Prentice-Hall International Series in Computer Science, 1996.

[2] 倉光 君郎. “Konoha 物語 episode 9 ディペンダビリティの追求”, Software Design 1 月号 2011, 技術評論社 pp.148–152.

[3] Jean-Claude Laprie. Dependability: Basic Concepts and Terminology, Springer, 1991.

[4] Haskell `http://www.haskell.org/`