数学的プログラミング環境 *

1Q−6

古田秀和　高橋正司　今村二康 †
中国日本電気ソフトウェア ‡

1 はじめに

コンピュータを使って、数学の問題などを考えていくとき、ユーザ・インタフェースと実際の問題との関連性が希薄になってしまうという問題点がある。また、プログラミングを行う際にも、同様の理由から、操作の系統的な理解が難しいという問題がある。このような点を、数学的な考えを取り入れたグラフィカル・ユーザ・インタフェースによって解決することがこの研究の目的である。

この問題を考察するために、本稿では証明支援システムを考えてみた。このシステムでは、証明の中にあらわれるような推論に直接対応する操作体系を持つことを目指している。このような操作をユーザが定義することによって、ユーザの考えた新しい概念を容易に理解できるようになるということを示すことが、このシステムの目標である。

2 証明支援システムの基本的な構想

数学の問題をコンピュータを使って解決しようとするとき、その問題について考えているという感じがしないということが非常に多い。この、「コンピュータを使って問題を考えることができない」ということの主な原因は、コンピュータの操作体系にあると考えられる。

本稿で述べる証明支援システムは、考えることができるシステムを目標としている。そのための操作体系として、証明の中にあらわれるような操作を使えばよいのではないか、というのが基本的な構想である。

数学の問題をコンピュータを使って解決するときの操作の方法としては、エディタを用いて数式や証明を入力したり、コマンドを入力することによって式を変形したり、図を表示したりすることが考えられる。このような操作の方法では、問題との対応がはっきりしていないために、操作を通じて数学的背景を理解することができないという点が問題となる。すなわち、エディタを使って数式を書いたり、読んだりすることや、メニューなどのシステムを操作することや、また、紙に数式や図を書いたりそれを読んだりすることは、それ自体は数学と直接結びつかないのではないか、と思われる。したがって、数学の問題をやっているにもかかわらず、数学をやっているような印象を受けないということにもなる。数学での定義を理解するためには、実験的な操作が必要だが、その操作があまりに実際の問題とかけ離れていると、それによって定義を理解することは困難となる。

この問題を解決するために、操作の体系自体を、たとえば図形の変形のようにある種の数学の理論と対応づけたものにするということを考えた。このような方法の長所は、操作と実際にやろうとしている問題との対応が理解しやすく、実際に問題を考えているという感じがするということである。また、この操作をユーザが定義することによって、ユーザの考えた新しい概念を、ユーザが理解することが容易に行えるようになると思われる。このようにユーザの理解を支援することが証明の支援であると考えられる。ユーザには証明はある操作に対応するものとして理解されるはずである。それは、自分の考える能力を支援してくれるものと言うことができる。

3 証明支援システムの構成

数学の問題を考えるためのシステムに必要な機能は、次のようなものとなる。

(1) ユーザが定義を入力することができる。

(2) 定義などを理解するために、ユーザが式の変形や図の変形などの操作をすることができる。

(3) ユーザが証明を入力することができる。また、証明が正しいかどうかを自動的に判定する。

(4) 証明を自動的に合成する。

自動証明システムは、(1)、(3)、(4)の機能を持つものと考えられ、証明チェッカと呼ばれるものは、(1)、(3)の機能を持つものと考えられる。また、(2)の機能を持つものには、数式処理システムなどがある。新世代コンピュータ技術開発機構(ICOT)で開発された証明支援システムCAP([1])は、証明記述言語、証明チェッカ、証明エディタ、式変形機能、等式エディタ、清書機能などからなり、(1)、(2)、(3)などの機能を持つものと考えられる。

本稿で述べる証明支援システムは、(1)、(2)、(3)の機能を持つものである。

---

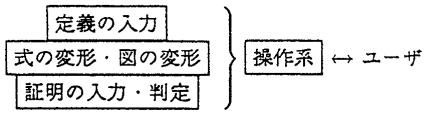
*A Mathematical Programming Environment

†Hidekazu FURUTA, Masashi TAKAHASHI, Tsuguyasu IMAMURA
‡NEC Software Chugoku

(1) の機能を実現するためには、あるプログラミング言語が必要であると考えられる。その言語では、数学の定義が容易に記述できなければならない。定義は複雑であることが多いので、この言語では、GUI を用いて入力をする必要がある。

(2) の機能は、式や図を表示し、ユーザの入力によってそれを変形していく機能である。ここでも GUI を使う。

(3) の機能を実現するためには、証明を記述するための言語が必要となる。その言語でも GUI を用い、その操作には証明の一部分が対応している。ユーザはそれを組み合わせることにより、証明を構成することができる。



4 証明支援システムの操作系

まず、次のような例を考えてみる。Δ を

$$\exists x \in X(T(x) \wedge V(x)) \tag{1}$$

$$\forall x \in X(T(x) \wedge Q(x) \supset S) \tag{2}$$

$$\forall x \in X(V(x) \wedge Q(x) \supset U) \tag{3}$$

という条件とする。この条件 Δ のもとで

$$\exists x \in X(Q(x) \supset R) \tag{4}$$

を証明する問題を考えてみる。証明図を LK のように書くとすると、

$$\Delta \longrightarrow \exists x \in X(Q(x) \supset R) \tag{5}$$

となる。まず、(5) を証明するために、局所的な変数 x を導入し、

$$\Delta \longrightarrow Q(x) \supset R \tag{6}$$

を証明することを考える (この x は (1) から導かれるものである)。(6) を証明するには

$$\Delta, Q(x) \longrightarrow R \tag{7}$$

を証明しなければならない (LK では

$$\frac{\Delta, Q(x) \longrightarrow R}{\Delta \longrightarrow Q(x) \supset R} \tag{8}$$

となる)。さらに証明を続けていくと、

$$\frac{\Delta \longrightarrow T(x) \qquad \Delta, T(x), Q(x) \longrightarrow S}{\Delta, Q(x) \longrightarrow S} \tag{9}$$

という推論規則から、

$$\Delta \longrightarrow T(x) \tag{10}$$

$$\Delta, T(x), Q(x) \longrightarrow S \tag{11}$$

を証明する必要があるいうことになる。この推論は (下式から上式を作りだしたと考えると)、論理式 $T(x)$ を作り出すものと考えることができる。ここで、(10) は条件 (1) から、(11) は条件 (2) から証明できる (これは (2) という性質を使ったと考えられる) ことがわかる。さらに証明を続けると、

$$\Delta \longrightarrow V(x) \tag{12}$$

$$\Delta, V(x), Q(x) \longrightarrow U \tag{13}$$

を証明することができれば、x を含まない式

$$S, U \longrightarrow R \tag{14}$$

を証明することに帰着できることがわかる (この時点で変数 x が消去されたと考えることができる)。

さらに X を実数全体の集合とし、(9) が次のようなものであるとする。

$$\frac{\Delta \longrightarrow x \leq a \qquad \Delta, x \leq a, y < x \longrightarrow y < a}{\Delta, y < x \longrightarrow y < a} \tag{15}$$

(x、y、a、b は実数とする)。このとき、この推論は、不等式の性質 (2) を使って y を消去するような不等式の変形と考えることができる。また、(1) は、実数の性質から成り立つことがいえる。

以上のような考察から、証明支援システムの操作は、次のようなものから構成されると考えることができる。

(1) 論理式を変形して、論理記号を取り去ったりする

(2) 局所変数を導入・消去する

(3) 新しい論理式を作る

(4) 与えられた性質を使う

(5) 等式・不等式の変形によって式の消去をする

(6) 実数の性質などを使う

5 まとめ

証明支援システムの操作として、どのようなものが必要であるかを分析した。今後これらの操作を、プログラミングに応用する方法を考えていく必要がある。

参考文献

[1] 坂井公: 証明支援システム CAP, 人工知能学会誌, Vol.5, No.1 (1990), pp.33-40.