数理解析研究所講究録
906巻 1995年 138-145

自然推論による微積の初歩の完全な形式化と そのプルーフ・チェッカー

Complete Formalization of Some Elementary Calculus Theorems by Natural Deduction and its Proof-Checker

日大理工数　高橋英之（Hideyuki Takahashi）
日大理工数　山下正人（Masato Yamashita）
（現・サンビックシステム（株））

要約　Genzenの自然推論（を用いた証明）に対する原初的な証明チェッカ PrfChecker を、Prologで自作した—これはよくある話で、特に難しい箇所はない。ついで、微積の初めによく出てくる極限に関する二、三のごく簡単な定理を例として、形式的な証明を考えた。それらに特殊な問題を扱えるように証明チェッカの拡張を行ないながら、それらの完全に形式的な証明を作成し、その証明を証明チェッカにかけて、通るのを確認した。この形式化の方は文献がないわけではないのだが、多様な定式化を許すであろうこの研究分野にしては研究が少なすぎる。また普通の数学科の教員・学生にとってfamiliarなものになっていない。そこには、数列をどうコンピュータに教えるか（「無限に対する有限的取り扱い」問題の一部）だとか、“関数の性質”についての「日常言語による自然な理解」をどう形式化するか、といった困難で興味深い問題がある。それについてささやかな経験を述べる。

これに関連して次の提言をしたい。いわゆる「研究」とは別に、学生の論理的な数学学習を助けることを目的とした「教育用研究」があるべきではないか。そこでは一方では論理処理ソフト（Ｌｏｇｉｃａとでも呼ぶべき、Mathematica の論理処理版ソフト）その他のソフトウェア的環境の整備は勿論であるが、他方ではそれ以上に、微積分ほかの数学の、より一層の形式化、記号論理化が課題となる。ここには為すべき多くの仕事がある。

§Ｉ　はじめに

孔子研究との関係　本研究のモティベーションを述べる。筆者の一人（高橋）の研究テーマは「『論語』の公理化から孔子のＡＩ化へ」というものである[1]。その方針は次のように二つの領域を結合することである：①論語を、価値論の論理体系として公理化する。孔子は価値論の論理学者（一種の数学者）であると見なす。②それをArtificial Mathema

tician あるいは Automated Reasoning（自動証明）の研究、という流れにのせる。するとそこにソフトウェアとしての孔子ができるだろう。／本講演は②に関係している。

「ソフトウェアによる孔子（あるいは数学者）をつくる」というテーマをトーンダウンすると、「コンピュータの中にモラルを学ぶ子ども（あるいは数学を学ぶ学生）をつくる」というテーマ、さらに「コンピュータにモラル（あるいは数学）を教える」というテーマになる。そのためにはコンピュータは如何なる装備を有するべきか。

コンピュータに数学を教える？　①モラルでも数学でも“意味のある”新しい命題を発見する「定理発見のメカニズム」は難しいようだ。②命題の“意味を理解”する「理解のメカニズム」は、まだ未知の要素ばかりである。③外からの命題の、証明を自ら見出だす「証明発見のメカニズム」は、低レベルのものなら既に作られている。④最後に、外から命題とその証明を与えられて、その証明の正しさを調べる「証明チェックのメカニズム」は、最も単純なものである。論理なるものの性質上からいっても、チェックは万人にとって、特に計算機にもできるものであるはずだ。／コンピュータが論理を扱うには、最低限、論理のチェック機構（証明チェッカ）を持つべきだ。

「チャート式」はエキスパート・システム？　こうしたテーマで我々がまず思ったことは、受験数学の「チャート式」なるものは、ソフト化すれば受験数学のエキスパート・システムになるのではないかということであった。エキスパート・システムとは現在成功している人工知能システムで、こういう状況ではこうしたらよい、という規則の集積として表される。だが、大学の数学については「チャート式」の参考書はないようである。

微積形式化の文献は少ない　われわれは当初、微積分学には完全に記号論理化した理論があるはずだと考えていたが、実際には数少ない。文献[2] は、自然数から始めて自然数の分数として有理数を定義しデデキントの切断により実数を導入する、という通例の各段階を形式的論理体系で追って行く。だが残念なことにすべての証明は概略のみで、加えて、実数を扱う論理体系が有理数の集合をベースにしており、不便さ・見にくさがある。

最近では自動証明や証明チェッカの研究のなかで、数学の形式的証明の試みもいろいろなされている[3] 。アメリカには、全数学の形式的証明をめざそうという呼びかけ（QED Project)もある。微積分学の形式化という研究分野はその重要性からして、たとえ多少の研究が既にあるとしても、もっと多くの試みがなされるべき分野であると考える。

§Ⅱ　数列の極限についての簡単な定理

（1）証明すべき命題の定式化　まず例題として考えたのは数列の極限に関するごく簡単な定理である。なぜ数列か？　有限（の数学の問題）を有限（であるコンピュータ）で扱うのは興味が少ない。無限（に関する数学の問題）を有限（であるコンピュータ）で扱うことが面白い。数列はその点、無限性が生まに出てくる問題である。／例題として次の命題を証明したい。証明すべき命題：

$\lim_{n\to\infty} a_n = p$　且つ　$\lim_{n\to\infty} b_n = q$　ならば

$\lim_{n\to\infty} (a_n + b_n) = p + q$　である。

この命題を、われわれが定めた簡単な証明記述言語（PrfDL）で書き表す。この言語では　$\lim_{n\to\infty} a_n = p$　という関係ないし述語を、lim(n tends to inf, [a,n], p)というProlog-like な述語で表す。すると上の命題は、次のように表される。

証明すべき命題

```
lim(n tends to inf, [a,n], p) and lim(n tends to inf, [b,n], q)
      implies lim(n tends to inf, [a+b,n], p+q).
```

数列の極限　ところで、このlim(N tends to inf, [A,N], P) とは、実質的に何を意味するのかを、定義してやらなければならない。

定義　lim(N tends to inf, [A,N], P) equiv

```
(sequence( {[A,N1]:N1=1 to inf} )and real number(P)) and
 any(Eps, Eps > 0 implies
  exists(N2, natural number(N2) and
   any(Ns, natural number(Ns) and Ns > N2 implies abs([A,Ns] - P) < Eps))).
```

つまり {[A,N1]:N1=1 to inf} は数列(sequence)であるということ、そして後は、極限値についての通常の$\varepsilon - N$論法である。

実数をベースにする　この論理式では限量子が動く集合は「実数」である。つまり実数の集合を初めから基礎にしている。「自然数→有理数→デデキントの切断→実数」という階段を登っていたのでは日が暮れるからである。これは一つの選択事項である。

（2）数列とは何かをコンピュータに教える　ここで「$\{a_n : n=1 \text{ to inf}\}$ が数列である」という述語は何を意味するのだろうか。数列とは何であるかをコンピュータに教えなければならない。数列に対する普通のイメージは $\{a_1, a_2, a_3, \ldots\}$ というものであろう。人間は柔軟にこの点々「．．．」を理解するが、この想像力をコンピュー

タに持たせることは困難に思える。我々の当座の（上の公式を証明するという）目的のためには、definition of sequence を次のように与えておけば十分であることが分かった。これはほぼ、数列とは自然数から実数への関数である、という言い方にひとしい。なお、sequence(XX) とは、「XXは数列である」という述語を表す。

定義 sequence({[A,N1]:N1=1 to inf}) equiv

name of sequence(A) and

any(N2, natural number(N2) implies real number([A,N2])).

ここで当然、自然数とは何であるか、を必要な範囲で規定しておかねばならない（略）。また、2つの数列の和の数列とは如何なるものか、を教えなければならない。 composite sequence A+B を次のように定義する。

定義 sequence({[A,N1]:N1=1 to inf}) and sequence({[B,N2]:N2=1 to inf})

implies any(N3, natural number(N3) implies ([A+B,N3] = [A,N3]+[B,N3])).

さらに、A+B の形も数列の名前 name of sequence であることを規定しておく。

定義 name of sequence(A+B), if name of sequence(A) and name of sequence(B).

しかし一般に数列の和差積商は，再帰的に行なわれることを思えば、この規則はPrologでプログラム化しておいた方がよい。

name of sequence(A+B):- name of sequence(A), name of sequence(B).

このとき次の補題が形式的に証明される（証明略）。

命題 sequence({[A,N1]:N1=1 to inf}) and sequence({[B,N2]:N2=1 to inf})

implies sequence({[A+B,N3]:N3=1 to inf}).

（3）関数とその性質についての主張 今問題にしている公式では、証明の途中に次のような箇所が出てくる。「自然数Ｎ１，Ｎ２のうち大きな方をＮとするなら、Ｎ≧Ｎ１ 且つ Ｎ≧Ｎ２である」。日常言語によるこの表現を、純形式的な枠組でとらえようとすると、事はそれほど簡単ではない。／ここにはＮ＝ｍａｘ（Ｎ１，Ｎ２）という“関数”が現れている。そして上の文は、そのＮに対して「Ｎ≧Ｎ１ 且つ Ｎ≧Ｎ２である」という性質のあることを主張している。つまり関数の導入と、その関数の性質の主張が、日常言語によって行なわれており、われわれはそれを直観的に了解している。そういった直観的了解をちゃんと形式化してコンピュータに教えてやらなければならない。

関数をどう形式的に定義するかが問題となる。関数はPrologプログラムで与えればよい

のではなかろうか—それもなるべくPrologの論理プログラムとしての部分を使う。

N=max(N1,N2)という関係は、Prologではmaxp(N1,N2,N) という述語で表される（Prologという言語の特徴による）。ここでこれら両者を結ぶ規則

maxp(N1,N2,N) equiv N=max(N1,N2).

が必要である。これに関連して、論理はいわゆる「等号をもつ理論」であるから、

定理 (X = Y) implies (Term1 = Term2), if Term2 は Term1 のX,Y の現れのいくつかをY,X に置き換えたもの.

などの、等号の公理や定理を設定しておかなければならない。

述語maxp(N1,N2,N) に対するプログラムは次のようになる。これが関数の定義である。

```
maxp(N1,N2,N) :- N1 >= N2, N = N1.
maxp(N1,N2,N) :- N1 < N2, N = N2.
```

関数をPrologプログラムで与えたとして、それを論理で扱うには、Prologと論理とのインタフェイス（接点）が必要になる。これは簡単で、PrologのA,B は論理の A and B であること、A:-B. は B imples A であること、Prologでは不成功によるｎｏｔが採用されていること等を言っておけばよい。それを通して関数の性質を論理的に証明する。例えば、

命題 maxp(N1,N2,N) implies (N >= N1) and (N >= N2).

命題 maxp(N1,N2,N) implies (N = N1) or (N = N2).

などの補題である。これらは人間には自明だが、機械には自明ではない。

（４）実数の諸性質 実数は、順序と代数と位相（あるいは距離）の性質をもち、それらは、等号や不等号に関する公式として表される。証明チェッカとしては、等号と不等号を含む式を導出する数式処理の能力をもっていることが望ましい。

デデキントの切断によって実数を定義したのならば、実数の性質は本来、自然数（や有理数）の性質から証明されるべき事柄である。しかし我々はいきなり実数から始めた。その場合でも、実数の性質を公理として立てて、他の性質はそれから導出するのが適切なやり方である。だが我々は時間の都合でそれもせずに、実数に関して必要な公式は、th of reals としてあらかじめ与えた。例えば次のような公式である。

```
th of reals([ X>Y and Y>=Z implies X>Z ]).
th of reals([ X1>X2 and Y1>Y2 implies X1+Y1>X2+Y2 ]).
th of reals([ abs(X) + abs(Y) >= abs(X+Y) ]).
```

基本公式を与えてそのいくらかの変形はプログラムで与える，ということも少しはした。X>Y とY<X の同一視、X and Y とY and X の同一視、などである。我々としては微積の数式的な面よりも、論理的な面により多くの興味をもった。

命題論理の諸公式　自然推論の立場からいえば、命題論理の公式はすべて自然推論によって導出されるものである。証明チェッカは命題論理の公式、特に同値公式は、その場で直ちに証明できるのが望ましい。これについても、必要な公式をあらかじめ与えた。

証明のプロセス　証明したい命題は、条件部および結論部に、述語 lim をもっている。証明のプロセスは、条件部の lim を、definition of limit の equiv の式の右辺で置き換え、それの anyや exists を推論規則によって除去し、途中によく知られたε/2の不等式についての和をとり、次いで anyや exists を推論規則によって導入し、definition of limit の equiv の式の右辺の形を作って、それを左辺の式に置き換えて、結論部の lim を作りだすことで終わる。証明は補題を含め行数にして約３５０行かかった。

関数の極限値とε－δ論法　以上は数列の極限値に関する命題のε－N論法による証明を形式的化するものであったが、他にわれわれは、関数の極限値に関する同様な命題のε－δ論法による形式的証明を実際に与えることができた（これは山下による）。

命題　lim(x tends to a, [f,x], p) and lim(x tends to a, [g,x], q)

　　implies lim(x tends to a, [f+g,x], p+q).

つまり、　$\lim_{x \to a} f(x) = p$　且つ　$\lim_{x \to a} g(x) = q$ ならば

　　$\lim_{x \to a} (f(x) + g(x)) = p + q$　である、

という命題である。／／今、「中間値の定理」の形式的証明を作ろうとしている。まだ最後までできていないが、大体うまくゆくようである。

§Ⅲ　証明記述言語PrfDL と証明チェッカPrfCheckerの実現

論理系としてはGenzenの提案した自然推論が人間の推論に近いものとして適当と思われる。われわれの証明記述言語PrfDL は、ほとんど文献[4] におけるＮＱによる証明の記法そのものである。始めに命題を書く。証明の各行は次のような形をしている。

[Label,　Logical Formula,　[Inference, UsedLabel1, UsedLabel2]].

どの論理式とどの論理式から、推論規則Inference を用いて、本論理式Logical Formula を導く、というふうに、推論規則 Inference を明示するもので、いわばアセンブリ言語

のように原初的である。具体例を書くと、

```
[22, exists(n, natural number(n) and
      any(ns, natural number(ns) and
       (ns > n) implies (abs([a,ns] - p) < eps/2))),
        [implies elimi,19,21]].
```

改良版PrfDL では推論規則を明示しながらではあるが、何ステップかの暗算ができる：

```
[3, continuous(f at X on closed interval(a,b)),[[j1,definition of continuous],
                                                 [j2,equiv elimi,2,j1],
                                                 [j3,any elimi using term X,j2]].
```

証明は、始めから終わりまで書き下して、ファイルに入れておく。証明チェックは一応バッチ式に行なわれるが、証明ミスに対して対話式に正しい式を入力することも可能だ。

次に、我々の証明チェッカPrfCheckerについて述べる。実現言語はProlog（具体的にはK-Prolog）、使ったマシンは SUNのSPARK IPX である。チェッカは証明の各行に対して、正しい推論がなされているか、特に限量記号any やexistsの導入や除去に際しての付帯条件が満たされているか否か、を調べ、その行の論理式が従属する仮定をメモしてゆく。ミスがあればメッセージを出して止まるか、または正しい論理式の入力を求める。よくあるミスは、付帯条件に言うところの「自由」という条件である。／最後にend of file にくれば、始めの命題と最後の行の一致を見、すべての仮定が除去されているかを調べ、正しければ簡単な証明木を出力しCongratulation!!と書いて終わる。

われわれのプログラミングでは、チェッカの作成と（数列の命題の）証明の作成とは互いのエラーを指摘しあうという形で平行して行なわれ、それにより両者のデバッグがうまくできた。要した時間は見当で延べにして、チェッカの作成に約70時間、（数列の命題の）証明の作成に約50時間くらいである。チェッカの作成に難しいことは特に何もない。形式的証明の作成の方が、よりクリエイティブな工夫を要するものでやや難しい所があった。

§Ⅳ　終わりに－評価と提言

数列の命題に関して、形式的証明は確かに存在した。微積のテキストに書かれている証明から、その形式的証明を作るのは、けっして自明ではなく、ある程度の工夫を要するいくらか難しい問題であった。しかし有限的な形式的証明の根本的な功績はε－N論法やε－δ論法にある。この論法の偉大さに比べれば、形式的証明に要した工夫は、価値が小さ

い。／それでもなお、微積のテキストを完全に形式化するという課題は“教育的な価値”がある。大学の教養課程の学生が数学を学び理解する際に、数式処理的な側面以上に大事なのは論理的な理解である。数式処理に対しては数式処理ソフトとしてMathematica やMaple 等があるけれど、それらに対応して、それらと同程度に扱いやすい論理処理ソフト—例えばＬｏｇｉｃａとでもいうような名前の—があることが望まれる。そんなソフトを用いて「コンピュータに数学を教える」という数学学習法がありうる—“教えることが最上の学習法である”という言葉が正しいならば。／次のようにまとめられる。

（１）論理の取扱いという点で、現在、人間とコンピュータとの間にはギャップがある。このギャップは、人間とコンピュータの双方からの歩み寄りで解消されるべきである。

（２）「コンピュータの側から人間への歩み寄り」—これは証明記述言語の高級化、というテーマとなる。他方、「人間の側からコンピュータへの歩み寄り」—これは数学の形式化、という研究テーマとなる。現在は、これら両面からのアプローチが必要であろう。

（３）それでもなおかつ数学に関して、人間とコンピュータとの間のギャップは、本質的で越えがたいものがあるように見える。このギャップを埋めようという試みは、究極的には、人間の数学的能力そのもののＡＩ的研究、というテーマにまでつながる。

いま数学にとって大事なのは、数学それ自体の研究と同時に、数学者が何故数学ができるのかの研究、数学的あるいは論理的思考能力のメカニズムは何かの研究である。そのテーマに、ここでいう「教育用研究」は関連をもつ。

§参考文献

[1] 高橋英之、「『論語』の公理化＝＞孔子のＡＩ化へ」、人工知能学会全国大会論文集、1993,7。

[2] G.Takeuchi, "Two Applications of Logic to Mathematics", Iwanami, 1978.

[3] 萩谷昌己、「証明チェッカとそのユーザインタフェース」、人工知能学会誌、Vol.10, NO.1, 1995年 1月。これにかなり詳しい文献がついている。なお、次の文献も参照。

南　俊朗、沢村　一、「対話型論証支援システム EUODHILOS」、人工知能学会誌、Vol.5,NO.1, 1990年 1月。

A.Quaife, "Automated Development of Fundamental Mathematical Theories", KLUWER ACADEMIC PUBLISHERS, 1992.

[4] 安井邦夫、『現代論理学』、世界思想社、1991。

[5]L.Sterling & E.Shapiro,『Prologの技芸』（松田訳）、共立出版、1986。