数理解析研究所講究録 1395 巻 2004 年 259-266

自動証明を効果的に行うための手法

大嶋 真治
SHINJI OHSHIMA
電気通信大学情報工学科

村尾 裕一
HIROKAZU MURAO
電気通信大学情報工学科

1 はじめに

計算機による証明は数学的知識の証明のみならず，セキュリティプロトコルやプログラムの仕様の検証 [3] といった方面での利用も注目されており，今後より一般に使用される機会が増えるものと考えられる．その反面，計算機による証明は曖昧さを排除した厳密なものであるという性質上，そのための記述も相応のものが要求されるため，利用者にとっては手軽に扱えるものでないことも事実である．即ち，証明記述のためにはシステムに関して熟知しているだけでなく，経験的な要素も重要となってくる．また，先に述べたとおり既に多くの内容の証明がなされており，それを整理し今後利用可能な形にすることも必要である．

本研究では，証明支援システム Isabelle/HOL[1] とインターフェースである Proof General[2] を利用した計算機による証明を主たる対象として，証明をより効果的・効率的に行うことができるような手法について検討し，それに沿った実装を行った．具体的な研究内容は，これまでに記述されてきた証明内容の整理から着手しそれを利用・参照することで新たな証明の記述を行うときの負担を軽減できるようにすることである．また，そのためのユーザーインターフェースの充実なども不可欠であろう．

本論文では，まず第 2 節で計算機による証明の概要として，本研究で主に対象とした証明支援システム Isabelle/HOL および Proof General について説明する．次に第 3 節では，それらを利用した計算機による証明で現在発生している問題点を挙げ，それらを改善・解消していくために必要な事項および実装すべき内容・方針を検討していく．続いて第 4 節では，第 3 節で挙げたものに従って実装した内容について説明する．最後に，それらに関して評価すると共に，現在不足している部分や今後検討していく必要があると考えられる内容について考察する．

2 証明支援システム

【Isabelle/HOL による証明】

Isabelle/HOL で定理または補題を証明するには，推論規則に基いて subgoal(前提) と呼ばれる証明に至る途中の式に分解することから始める．これを Simplification (簡単化，簡易化) といい，又このような手法を tactic と呼んでいる．実際には，この tactic を何度も繰り返し (tactics という) 最終的に前提となる subgoal が仮定や定理 (goal 又は結論) に到達すればもはや subgoal はなくなる．この subgoal がなくなった状態を No subgoals と呼んで証明されたことを示す．

実際に証明を行うための記述例を図 1 に示す．この例では，最初の"…"によって括られた文字列が分配則 (lemma nsDistr) を表す証明すべき式であり，それに続く apply の列が証明を記述する推論規則の適用である．全体の流れとしてはまず初めに帰納法を適用し，続いて幾つかの定理・補題 (詳細は略) や仮定に含まれる式を用いて徐々に簡単化していき，最後の行の done でこの定理の証明が終了する．

```
1    lemma nsDistr:
2      "\<lbrakk> Ring R; x \<in> carrier R \<rbrakk>
3     \<Longrightarrow> add R (nscale R x n)
4      (nscale R x m) = nscale R x (n + m)";
5      apply (induct_tac n)
6      apply simp
7      apply (subst gZeroL)
8        apply (rule RingIsAGroup, assumption+)
9        apply (rule nsClose, assumption+)
10      apply simp
11     (* induct: prep=case of n ==> case of n+1 *)
12     apply simp  (* apply (subst nscale_suc) *)
13     apply (subst gAddAssoc)
14       apply (rule RingIsAGroup, assumption+)
15       apply (rule nsClose, assumption+)
16       apply (rule nsClose, assumption+)
17     apply simp
18     done
```

図 1: "$(x \times m) + (x \times n) = x \times (n + m)$" を証明する記述

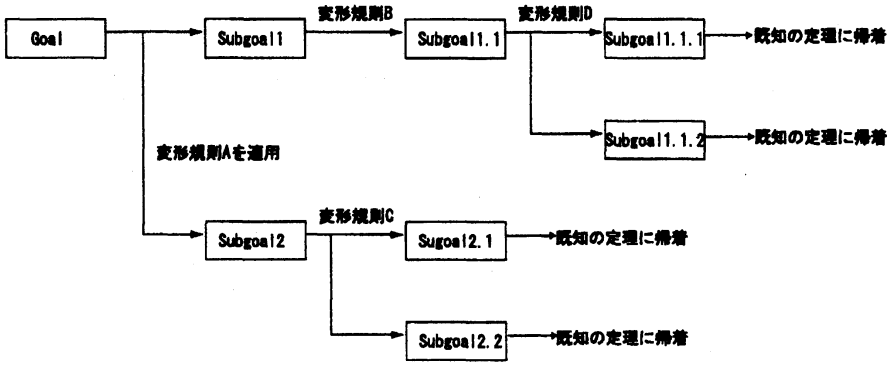

図 2: 証明の流れの概念図

【Proof General 概要】

Proof General(以下、PG と略すこともある) は，各種証明支援システムを利用しやすくするための，GNU Emacs 上で動作する汎用インターフェースである．対象となる証明システムとして代表的なものに Coq や LEGO などがあるが，本研究ではその中で特に Isabelle 向けに用意されている部分を利用する．

Proof General は，Emacs 上でのコマンドラインインタープリタとして動作し，証明記述を一行ずつ対話的に解釈・処理する．このとき，証明の進行状況を色別に表示するといった処理がなされ，また証明の任意の場所まで進んだり，逆に戻ったりするという操作が容易に行えるようになっている他，証明内容でエラーが起こった場合などもわかりやすくなっている．更に，Proof General を XEmacs 上で動作させる場合は，X-Symbol(http://x-symbol.sourceforge.net/) と呼ばれるパッケージが利用可能であり，これによって特殊な数式記号などもグラフィカルに表示可能となるなど有用なインターフェースとなっている．

【Isabelle と Proof General の連係】

Proof General は，インタプリタとして対話的な動作をするために，Isabelle へのコマンドとして受け取った命令に独自な特殊なプロトコルに従った情報を付加し，それを処理した Isabelle からの返答を解釈することで，処理の同期を取るほか証明進行のグラフィカルな出力等を行っている．

このために，Isabelle では Proof General との対話用のモードが用意されており，データを受け取って処

理を行うと，同様の情報を付加した結果データを Proof General への返答とするようになっている．このことは，4.1 節の内容と関連しており，証明過程の有用な情報を多く含んでいる．また，このようなプロトコルに従ったデータの交換を今後 PGML により符号化することで行うといった方法も考えられている[1]．

Proof General は Emacs 上で，3 つの主要なバッファを使用して動作する．1 つ目が，使用者が実際に証明のためのコマンドの記述などを行う script buffer であり，編集作業ををここで行う．コマンドが正常に実行された部分は色分けして表示される．ここで記述されたコマンドを実行して，証明の過程である Goal や Subgoal の情報を保持するのが goals buffer である．この部分は，証明のステップが進むごとに随時更新されていく．3 つ目の response buffer では，Isabelle へのコマンドの送信と対応する返答がそのままの形で残されている．他に自動で簡単化を行った処理の内容 (trace) を保持するバッファなども利用可能である．

3 証明を行う上での問題点と改善案

3.1 利用者に要求されるもの

証明支援システム Isabelle を利用した計算機による証明は，前述したように曖昧さを排除した厳格なものとして行える反面，利用者は単に証明したい命題の分野に関連する知識だけでなく，証明システムを使いこなすために様々なことを意識しなくてはならないのが現状である．また，ある命題を証明する場合に熟練者であれば経験的に解法のパターンを見出してそれに従って記述をするというケースもありうるが，そうでない利用者にとっては試行錯誤を繰り返すことになる．更には，証明が複雑になり関連する定理・補題が多くなると熟練者であってもそれを把握するのが困難になるのは事実である．以下に，利用者が Isabelle を利用して証明を記述する際に意識しなくてはならないと考えられる事項を列挙する．

1. Isabelle 上での定義 (概念の形式化)：証明を可能とするために最も重要な事項であると言える．即ち，ある命題を計算機上で証明するためにいかにその概念を証明しやすい形で形式化できるかというであり，この定義次第で証明の難易度は大きく変化する．

2. 繁雑さの除去：ある証明を完結させるためのアイデアは必ずしも一通りではなく，結論が導かれればそれは論理的に正しいとは言える．しかし，可能な限り解り易い記述となるよう心掛けるべきである．具体的には，証明過程の各ステップでどのような推論規則を適用するのか，それにどのような意味があるのかなどを明確にすることである．また，既に証明した補題を把握することで無駄な補題の証明を減らすことも必要である．

3. システムの動作への理解：Isabelle は Goal や Subgoal に推論規則を適用する操作などを自動で検索して適合するものに従って変形・簡単化を行うことができる (simp, auto など) が，このようなコマンドを多用することは証明を解りにくくするだけでなく，意図しない変形を行うといった場合もある．このような動作を防ぐためにも，Isabelle が保持する親 theory を証明に必要な必要最低限のものにしたり，無駄な補題の証明を省き，重要な部分での証明には適用する推論規則を明確にするなどの努力をするべきである．また逆に，自明な推論規則が自動的に適用されるよう，適用可能な形への部分的な変形，あるいは補題の作成も必要である．

ここに挙げた事項は，補題の証明をサブルーチンの作成とみなして，一般的な計算機上でのプログラムの記述の場合の一例として考えることができる．プログラミングで同様におこる問題を参考にすることで，これらを改善する別のアプローチも存在すると考えらえる．

[1] 文献 [2] に関連するウェブサイト http://zermelo.dcs.ed.ac.uk/kit
内容は http://homepages.inf.ed.ac.uk/da/papers/drafts/#white に詳しい

3.2 証明支援システムの性質上の問題点

1. 証明内容の類似・重複性
   Isabelle で変形規則として補題を用いるには厳密に式の両辺のパターンを一致させるほか変数の型などを考慮しなくてはならないため，机上の証明のほど柔軟な形で用いるのが難しい．そのため，補題に厳密に適合する形への式の変形などがその都度必要となり，非常に類似した内容のものでも手間が掛かることがある．類似した内容，例えば変数が一部異なるといったようなものへの柔軟な対応を考えることが必要である．

2. 証明の規模とそれに伴う処理時間増加
   上で述べた通り，ある命題を証明するときは既知の定理から補題等を導きそれらを利用するというパターンになるため，複雑な内容になるほど記述するべきものも多くなる．Proof General は Isabelle と利用者とのインターフェースとして動作するとき，インタプリタとしてコマンドを読み込むとそれに独自のプロトコルに従ったデータを付加し，それに対応する返答を待つという同期を取って処理を行う．既知の定理として必要な theorem や最終的な命題に至るまでの補題などが多くなるとその全てにそのような処理を行うため，かなり処理時間に影響してくる．従って，可能な限り無駄な記述や引用を省くほか，類似した補題等の証明を効率的に行う必要がある．

3.3 改善案と設計方針

【証明の解析例】

これまで説明してきたように，Isabelle を利用した計算機による証明には単純に解決できない問題が数多くある．ここではそれらにいかに対処するかという案を実際に証明を行った際の例をもとにして提示するとともに，それを実装するために必要なことを検討する．

まず，証明が行われるとき Goal を導くまでにどのような過程を通って分解・変形されるかについて説明する．**図 3** は，2 節の Isabelle の説明の中で証明記述例として挙げた，環上の分配則の証明 (**図 1**) を Isabelle に実行させたときの推論規則を適用させる各コマンドと返答を対応付け，全体の流れを解析したものである．証明は，結論である Goal に対して推論規則を適用させて分解・変形をし，そこで得られた Subgoal に対して同様に繰り返すため図のような木構造として考えることができる．

【証明過程の表示法】

現状の Emacs 上での Proof General を介した Isabelle の利用では，インタプリタとして一行ずつコマンドを読み込み，それに対する返答である Subgoal が表示されるという形になっている．この場合，証明の各段階での状況は常時確認可能であるが，全体の流れを掴むためには証明を進めたり戻したりという作業が要求されてしまう．また，結果的に証明が完了して Goal まで導くことができたとしても辿った経路などの情報は不足し，前節で述べたような適切な記述となっているか判断しがたい．

そこで，証明の流れを把握しやすくするために**図 3** で示したようなグラフィカルな形で表示できるような実装を考える．即ち，図に示したような木構造を用いたチャートを GUI で実装し，そこに適用する各推論規則と Subgoal の情報を対応させて証明の過程を表現する．これにより，証明全体の見通しがよくなり推論規則の適用とそれに対する変形がどのように行われたかが解りやすくなる上，後述するような証明が重複する部分などを発見することで無駄な記述を省くのに役立つと考えられる．

【類似検索とデータベース】

また別に実装すべき機能として，証明の類似性などを検出することで重複部分の排除の他，証明済みの補題を検索してそれを引用した記述を行うことが考えられる．具体的には，証明の過程に出現する Subgoal の式などを比較の対象として類似性の検出を行ったり，そのような式をデータベースに保存することでその時

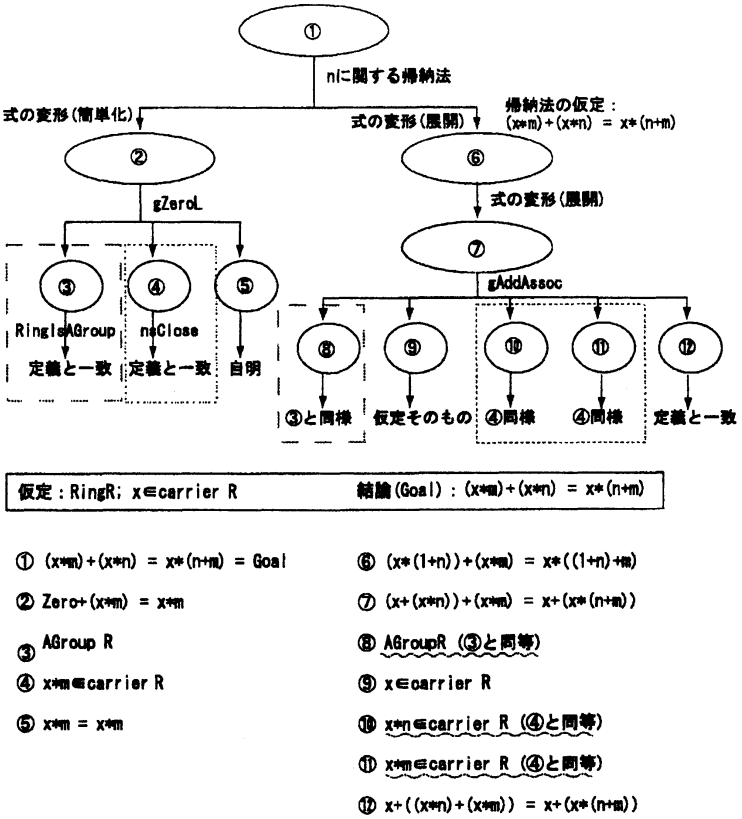

図 3: 証明過程の解析例

点で行っている証明だけでなく別の証明を行う際に参照可能となる.

ここで, Isabelle で使用する場合の数式の記法は独自のものである上, Proof General で X-Symbol を使用するための数学的記号などが混在するため, 単純に検索等の処理を行うのが困難となる. また, 数学オブジェクトの表現方法を標準化した OpenMath[5] に従って数式を XML 形式で符号化して格納しているデータベース [7] との連係等の面から, 同形式を含めた別の形式へと一度変換するのがよいと考えられる. OpenMath に限定しないのは, これらの式を一般的な数式としてグラフィカルに表示する場合, OpenMath よりも MathML[8] などがより適していると思われるためであり, 検討する必要がある.

類似性の検出などの操作は, 証明過程において無駄なで排除可能な部分を特定することが目的である. 例えば, **図 3** では最初の推論規則として帰納法を適用させ大きく 2 つの部分木に分けているが, この結果出現する Subgoal を順番に証明していくと同等, あるいは類似と言える箇所が出現する. このような場合, 予め仮定として必要となる定理・補題などを含める形にするといった記述をすることが 1 つの解決策となる. これらの作業を自動化し, 検出した上で記述の変更を行うということが理想的であるが, 現時点では利用

者が証明の全体の構造を把握した上で，式を指定することで検索が行えるような実装を考える．

4 証明解析モジュールの実装

4.1 証明ログの抽出とその解析

【Isabelle の返答】

3.3節で説明したような機能を実装するために証明過程の解析作業を行うが，そのためにはまずIsabelleに与えた命令であるコマンド(推論規則)とその返答であるSubgoalの内容とを対応づける必要がある．Isabelleの出力からは証明開始から終了までのSubgoalの増減の情報を読み取ることは可能であるが，具体的な適用した推論規則や定理・補題等の情報がないため，入力との対応づけを別作業として解析することになる．そのような解析法で実装する場合，Isabelleの実行とともに常時動的な解析を行える可能性があるという利点はあるが，そのためにはProof Generalを介した処理なども考慮する必要があり非常に手間が掛かる上，Isabelleのログに現れない自動で推論規則の適用を行った箇所(apply simpなど)などの対応付けが困難である．従って，まずは出来る限りシステムが出力するログのみをもとにした静的な解析を行い問題解決をしていく方法を考える．

【Proof General を介したログの利用】

前節で説明したIsabelleの出力の内容は，Isabelleを単独で動作させたときのものであるが，ここでは同時にProof Generalを利用した場合の出力内容について考える．Proof Generalは利用者とIsabelleとを仲介するインターフェースであり，独自の仕様に従った情報付加等を行いインタプリタとして同期を取っている．

通常，Proof Generalではそのような情報を利用者からは遮蔽し，Subgoalなどの証明状況の情報はIsabelle単独で動作させたものと同様に過不足ない形で見えるようになっている．Proof GeneralはEmacs上で複数のバッファを同時に使用することでそれを実現している．即ち，Proof Generalは利用者とIsabelleとのインターフェースを担っているということから，入出力の双方を管理し必要な情報だけを表示している．

具体的には，Proof Generalは起動されるとIsabelleのtheory file(.thy)として利用者が実際にエディットを行うファイルをscript bufferと見なすほか，(1) isabelle/isar, (2) isabelle/isar-goals, (3) isabelle/isar-response, (4) isabelle/isar-traceの4つのバッファを用意する．これらのうち，(2)と(3)は(1)のバッファから抜き出した情報となっており，証明が前後するたびに更新され，証明を記述する際には専らこれらのバッファをIsabelleからの返答として参照することになる．しかし，解析する場合にはIsabelleへの入出力が全て記録される(1)と，Isabelleが自動で推論規則のマッチを行った記録のための(4)のバッファが有用である．全体的な証明の流れを把握するには，(1)のisabelle/isarバッファをログとして参照し，推論規則の適用と応答の部分を切り分ければよい．従って，実際に証明を行った際にこのバッファの内容を保存しておくことで任意のタイミングで静的な解析を行うこととした．

【ログの解析】

上のようにして得られた証明のログを，推論規則の適用とその結果のSubgoalの個数・内容がわかるように対応付けて切り分け，後述するGUI部分に対応させるための木構造のデータを構成するモジュールを作成した．ここで注意すべきこととして，Proof GeneralとIsabelleでデータ交換を行うときには独自のプロトコルに従ったコードの付加がされているため，その部分を考慮し切り分けのための情報とするほか，データとしては除外する必要があった．Proof Generalではそのようなコードと返答の内容を切り分けるために，証明用のキーワード(コマンド)やコードを正規表現として扱っている．

実装した解析用のモジュールでは，これらの情報をProof Generalと同様な形で最大限に活用し，証明のログからSubgoalの分岐と適用する推論規則を対応付けた内部的な木構造データを持つように切り分けることを可能とした．実装は，後述のGUI部や既存のデータベース等との接続を考慮しJava言語で記述した．

4.2 GUIによる表現–証明構造のグラフィカルな出力

3.3節で説明した内容に従って得られた構造を持ったデータを，利用者にとって見やすいグラフィカルな表現で出力するために実装したモジュールについて説明する．図4は，ここで実装した証明の全体の構造を木をモデルとしたGUIモジュールのスナップショットである．

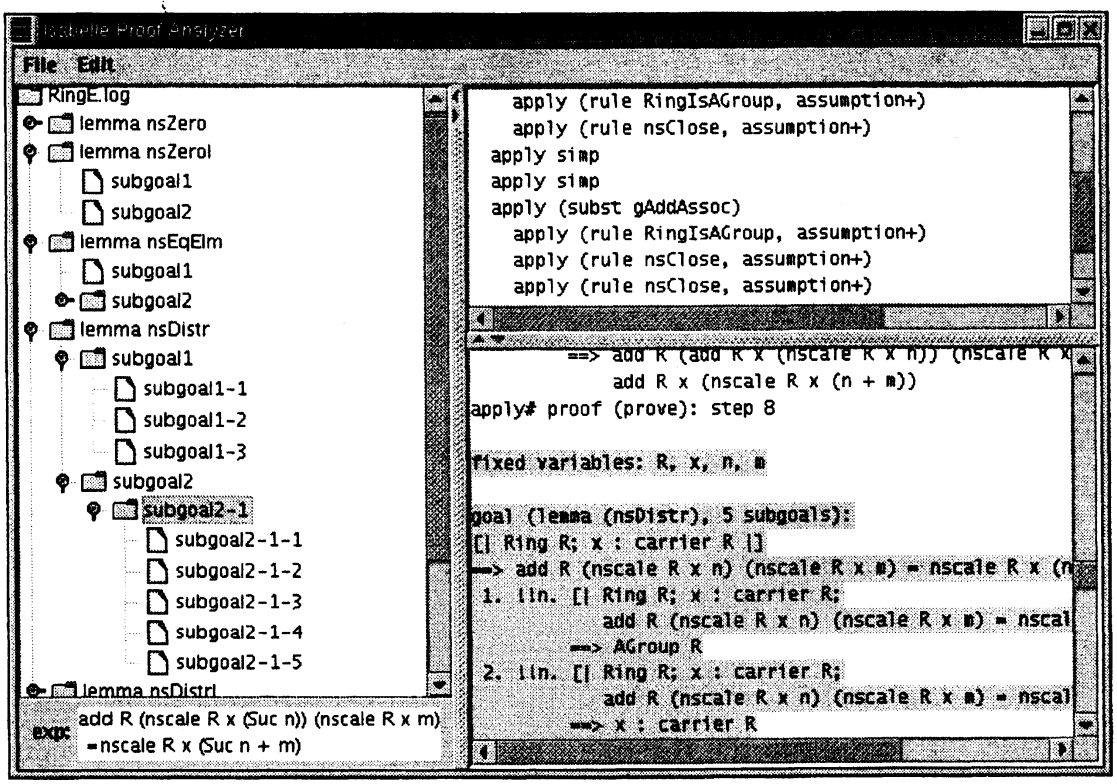

図 4: 証明構造を表現するGUIモジュール

GUIモジュールの左半分には，証明の流れとしてGoalやSubgoalに対して推論規則を適用し分解・変形していく様子を木構造で表示している．木の各ノードにはGoalあるいはSubgoalが対応しており，対象の式を確認することができる(左下に表示されている)．同時にそのノードに対して適用した推論規則の情報が格納されており，子ノードを持つ場合は更にSubgoalが導かれた場合であり，持たない場合はそこで証明されたことがわかる．右側のテキストエリアには，ログを解析することにより得られたIsabelleへの入力と出力について，各補題単位の対応する箇所を表示している．

Isabelleで利用可能な推論規則は，非常に基本的なものから複雑なものまであり，現時点では全てに対応するのは困難である．実装した解析モジュールでは基本的な，即ち証明を記述するときにテクニカルな手段(Subgoalの入替え，仮定の挿入等)を用いない記述によって証明を進めた場合を中心として対応している．

表示を行うまでの段階では，データとしての式は全てIsabelleに対して記述する際の形式のまま扱っている．即ち，Subgoalが複数ある場合に推論規則がどのSubgoalに対して適用されたかといった解析には，前後のSubgoalを比較し一致確認の操作を行っているが，この処理においてもテキストとしての式のマッチング処理となる．このような，同じ式が繰り返し出現するような性質を持ったログを対象として，式が一致することを前提としたマッチングでは単純な方法でも対応可能であるが，式の類似性などの検出を目的とする場合には効率性の面からデータ形式を考慮する必要がある．

5 おわりに

今回本研究では，計算機による証明を利用する上で現状の困難な状況を改善し，より効率的に利用・記述することを目標とした．証明を支援するシステムとしてIsabelle/HOLとインターフェースのProof Generalを利用し，証明を試みるとき発生している問題点，あるいは改善可能であると考えられる内容について，利用者側そしてシステム側に起因して起るものを提起し，それぞれに対する改善策を検討するとともにモジュールとして実装を行った．実装した内容としては，証明のログを抽出しその内容である各定理・補題における推論規則の適用とSubgoalの分解・変形の構造を解析するとともに，それをGUIを利用することでグラフィカルな表現で出力可能とするものである．

これらの解析結果を今回実装したモジュールを使用して参照することにより，これまでのXEmacs上でProof Generalをインターフェースとして用いる場合のテキスト主体な構成から比較すると，利用者にとってはGUIによる証明内容の木構造をモデルとした表現を参照することで証明の構造の把握がしやすくなり，それをもとにして証明記述の見直しをする上では有用なものになったと言える．しかし，現時点ではその解析結果を利用した計算機によるアプローチが困難であり，証明内容の見直しなど人為的に行われている部分での利用者による負担を軽減する意味でも更なる拡張・発展が必要不可欠である．

そこで，今後の機能拡張によってより有用なものとするための候補としては，機械的な手段によって証明内容の類似性・重複性の検出を可能とすることで，現在利用者が行っている部分を計算機に処理させ，既に記述された証明内容のうち関連するものを数式データを指定して検索するといった機能が課題して挙げられる．このような機能の実装により，目的とする補題を式を元にして捜したり，或いは類似した一連の定理全体を記述のための参考にすることが考えられる．このような用途のためには，証明された定理・補題に関する数式をデータとして保持する外部データベースとの連係や，単に数式データにとどまらず証明過程を一つのデータとするようなシステム[6]との関連性も無視できないものである．それ以外にも，数式を一般的なグラフィカルな形で表示するため外部ブラウザなどとのデータ交換なども拡張案としては考えられ，それらの実装のために重要となるデータ形式の検討も急務である．

参考文献

[1] T.Nipkow, L.C.Paulson, M.Wenzel: *Isabelle/HOL A Proof Assistant for Higher-Order Logic*, LNCS2283, Springer (2002).

[2] Proof General. `http://zermelo.dcs.ed.ac.uk/`

[3] L.C.Paulson: The inductive approach to verifying cryptographic protocols, *J. Computer Security* 6, pp.85-128(1998).

[4] The MBase System. `http://www.mathweb.org/mbase/`

[5] The OpenMath Society. `http://www.openmath.org/`

[6] 舩戸正和 他：“代数の自動証明を目的とした分散システム”，第2回情報科学技術フォーラム (2003).

[7] H.Murao, S.Ohshima, H.Kobayashi: Experimental Theorem Database System for Mechanizing Ring Theory, International Congress of Mathematical Software,(2002).

[8] Mathematical Markup Language (MathML) Version 2.0 (Second Edition). `http://www.w3.org/TR/MathML2/`