ICOT Technical Memorandum: TM-0175

TM-0175

CAPプロジェクト (1)
—ねらいと構想—

廣瀬　健(早稲田大学)，玉井哲雄(三菱総研)
横井俊夫，横田一正，坂井　公

June, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

7L-1

CAP プロジェクト(1)
——ねらいと構想——

廣瀬健，　横井俊夫，横田一正，坂井公，　玉井哲雄
（早稲田大学）（ICOT）　（ICOT）　（ICOT）（三菱総合研究所）

1. はじめに

CAP (Computer Aided Proof)は，数学者の日々の活動を補佐し，協調して問題を解決してゆくためのシステムの実現をめざしている．そこには多くの活動——定理の証明，数式の操作，論文の執筆，など——が考えられる．もちろんこれらの活動は数学者だけではなく，物理学者やコンピュータ科学者や経済学者など，数学を利用するすべての分野の人々のおこなう活動である．CAP はこれらを補佐するツールを研究，開発するために，ICOTおよび同ワーキング・グループが進めているプロジェクトである．

2. 知的プログラム・システムとCAP

このCAP プロジェクトは，ICOTの知的プログラミング・システムの研究，開発の一環としておこなわれており，その全体はデータを中心に考えると図1 のようになっている．ここでJSL,PTS,TRS,FMS とは

- JSL(Japanese Specification Language)
  日本語仕様記述から形式的仕様記述への変換システム
- PTS(Program Transformation System)
  形式的仕様記述からプログラムへの変換システム
- TRS(Term Rewriting System)
  項書換えシステム
- FMS(Formula Manipulation System)
  数式処理システム

のためのプロジェクトを意味している．CAP はこれらと有機的に関連したプロジェクトである．

もちろんそれらの全体を「有機的」に結合するのは容易ではない．そのために仕様，プログラム，証明，計算，アルゴリズム，書換え規則，などを問い直すことから始めている．その中で現在もっとも先行した活動をおこなっているのがこのCAP プロジェクトである．

3. CAP プロジェクトのねらい

このプロジェクトの最終目標の１つは，ほとんどの数学者がもっている膨大な知識の他に，さまざまなユーティリティ——証明エディタ，２次元入出力，数式処理など——を備えた一般的な定理証明のための協調問題解決システムである．しかしこれはICOTプロジェクトの短い期間と限られた労力では実現はなかなか容易でない．そこでICOTでの研究，開発はそのためのプロトタイプの研究，開発であり，現在ワーキング・グループを中心に多くの数学者の協力を得ている．

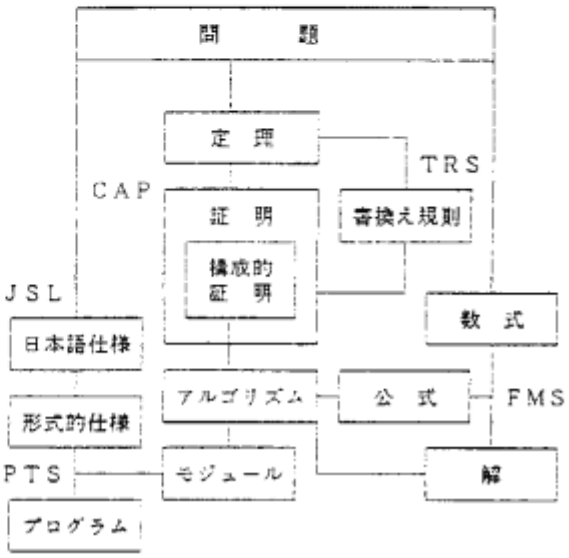


図1 知的プログラミング・システムの枠組

もう１つの目標は，プログラムの変換，合成，検証への定理証明技術の適用である．ICOTで研究している論理プログラミングがこのような数学的な扱いに適しているのはいうまでもないが，変換，検証などの技術が充分に成熟しているとはいいがたい．そこで数学とプログラムとの類似性に着目して，知的プログラミングの基礎技術としてCAP を考えている．すでにプログラムの導出過程を証明過程と対比させて記述するという実験もおこなっており，今後のより密接な関係が予想されている．

4. 証明チェッカの構想

しかし現在のわれわれのこの分野における経験はあまり多くない．そこで方針として，まず数学者がノートがわりに使用できる，数学の特定分野の証明チェッカのシステムを開発し，その経験に基づきシステムを徐々に機能アップしてゆくことにした．

証明チェッカは，与えられた証明が妥当かどうかをチェックするシステムである．そのためには，定理や証明を記述するための形式的な証明記述言語がまず必要になる．そしてその入力や修正のための証明エディタも重要である．これを軽視すると「ノートがわり」に利用することが困難になってくる．次に証明の正しさを検証するためのチェック機能が必要である．ふつうの数学の教科書では，証明の行間にかなり隙間があるが，それをどの程度認めるかがこ

のチェック機能の大きな問題である．そして公理や定理を管理するための知識ベース機能も考えなければならない．

この証明チェッカから得られるさまざまな経験，とくにそのような活動に適した協調問題解決システムのための人と機械とのインタフェースの経験，TRS を含んだ強力な推論メカニズムの経験，論理式や数式を扱うのにふさわしい知識ベースの研究は，次のステップのシステムに大きく役立つだろう．

われわれは，実際に開発する証明チェッカのために次の3つの分野を選んだ：

(1) 線形代数(LA) [1]

(2) QJ [2],[3]

(3) 総合微分幾何(SDG) [4],[5]

LAは言うまでもなくもっともよく知られた数学の分野である．この証明チェッカの目標として，大学初年度の線形代数の教科書を選び，そこに掲載されているすべての証明のチェックをめざしている．行列の扱いはまた2次元入出力についての経験にもなるだろう．

QJは東京大学の佐藤によって設計された論理体系であり，またプログラミング体系でもある．QJに基づいたプログラミング言語Quteは，QJ 証明チェッカの実装言語ともなる．QJでは超数学の扱いも可能であり，Godel の不完全性定理の証明チェックを最初の目標にしている．

3番目のSDG はKockによって創始されたひじょうに新しい分野であり，微分幾何学をカテゴリの枠で研究することをめざしている．この証明チェッカは，項書換えを中心にした証明の作成ができることをめざしている．

この3つはそれぞれ独立したサブプロジェクトとして構想されているが，いずれも数学の特定分野に限定した証明チェッカというわけではなく，汎用的な証明チェッカをめざしており，将来はすべてが統一された定理証明のための協調問題解決システムを考えている．

5. CAP プロジェクトの進行状況

現在これらのサブプロジェクトの中では，LAがもっとも先行している．これまでのCAP およびLAの歴史を簡単に述べよう：

(1) 1982年9 月にICOTの基礎理論ワーキング・グループ(WG5) が第5世代コンピュータの理論的土台作りのために設立された．

(2) 1983年11月より証明チェッカの検討が始まり 1984年4 月WG5 のプロジェクトの1つとなった．

(3) 1984年に数学の知識はもっていない証明チェッカの開発が始まり，1985年3 月に完成した．

(4) 1985年4 月より，具体的な教科書に沿ってLA証明チェッカの研究，開発が始まり，1986年3 月に1.0 版が完成予定である．

(5) 1985年9 月にCAP ワーキング・グループが設立され，LAの2.0 版の検討，3 つのシステムの統一化構想など，将来のCAP システムについて議論をおこなっている．

6. 他システムとの比較

CAP に類似したシステムはすでにいくつか作られている：

- LCF(Logic for Computable Function) [6]
- EKL [7]
- Mizar [8]

LCF は当初スタンフォード大学で，そして現在はエディンバラ大学とINRIA で研究，開発されているシステムであり，LAシステムと比較的類似している．EKL はスタンフォード大学で作られたTRS を基にしたシステムであるが，SDG システムとよく類似している．Mizar はワルシャワ大学で開発されたシステムであるが，現在ではアメリカの大学でも使用されており，自然言語に近い証明記述と長い使用経験が特徴である．CAP はこれらの経験を参考にしながら，今後の研究，開発に生かしてゆきたいと考えている．

謝 辞

CAP プロジェクトのためにいつも多大な時間を費やしていただいているCAP ワーキング・グループの委員ならびにオブザーバの方々に感謝いたします．

参考文献

[1] K.Sakai and K.Yokota, "CAP Project—An Approach to the Mechanization of Mathematics", Proceedings of the Third Japanese-Swedish Workshop, pp10-12, Nov.13-14, Tokyo, 1985.

[2] M.Sato and T.Sakurai, "Qute: A Functional Language based on Unification", Proceedings of the International Conference on Fifth Generation Computer Systems, pp157-165, 1984.

[3] M.Sato, "Typed Logical Calculus", ICOT CAP-WG資料, 1985.

[4] M.Hagiya and S.Hayashi, "Some Experiments on EKL", ICOT TM-101, 1985.

[5] S.Hayashi, "Towards Automated Synthetic Differential Geometry 1——basic categorical construction", ICOT TR-104, 1985.

[6] M.J.Gorden, A.J.Milner and C.P.Wadsworth, "Edinburgh LCF", Lecture Notes in Computer Science, 78, Springer, 1979.

[7] J.Ketonen and J.S.Weening, "EKL—An Interactive Proof Checker, User's Reference Manual", Department of Computer Science, Stanford Univ., 1983.

[8] A.Tryburec and H.Blair, "Computer Assisted Reasoning with Mizar", IJCAI'85, pp26-28, 1985.

7L-2

CAPプロジェクト（2）
— 証明記述支援環境 —

横田一正(ICOT)，藤田正幸（三菱総合研究所）

1．はじめに

本稿では，ICOTにおけるCAPプロジェクトで開発中のCAP－LAシステムの，全体構成と証明記述支援環境の概要を述べる．証明記述支援環境は，計算機になじみのない利用者が，線形代数理論の証明の展開を行うときに，本証明チェッカに抵抗を感じないで利用できることを目的として設計された．このような機能を十分に引き出すために，本システムのユーザ・インタフェースはICOTの逐次推論マシンPSIの提供しているビットマップディスプレイ，マウス等のインタフェース機能を利用している．

2．CAP－LAシステムの全体構成

CAP－LAシステムは，大きく分けて次の3つの部分により構成されている．（図1）

①エディタ（証明エディタ・ブラウザ）

②証明チェッカ

③理論知識ベース

また，これらのサブシステム全体を制御し，利用者に各機能を提供するのがCAP－LAシステム・コントローラである．このCAP－LAシステム・コントローラを中心として，証明記述支援環境は構成されている．理論知識ベースを除く各サブシステムの概要については，CAPプロジェクトの他の発表を参考にされたい．

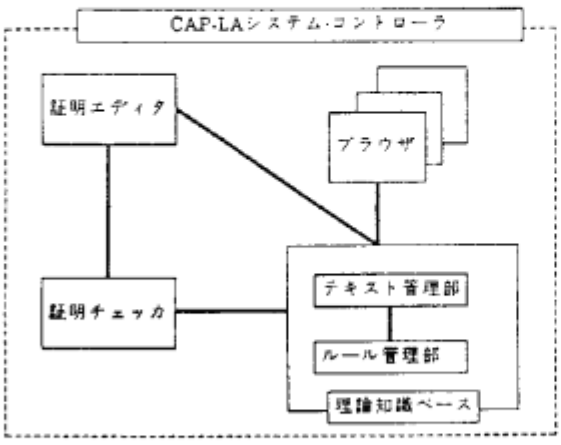


図1　CAP-LAシステムの構成

理論知識ベースは，証明記述を保存し，エディタ，証明チェッカに供給する機能を持っている．このとき，エディタが扱うテキストとしての証明記述と証明チェッカが扱うルールとの一貫性を，理論知識ベースの中で自動的に保っている．

3．証明記述支援環境の機能

CAP－LAシステムの，各サブシステムの機能を利用しやすくするために，以下のような工夫を行った．以下では，定理証明記述の一連の操作を通して説明する．

①証明記述の構築

CAP・LAシステムは，理論知識ベースのテキスト管理部を通して，証明記述を保存，利用することが可能である．本システムで取り扱う証明記述の構成は図2のようになっている．

図中における"理論"が構築及び証明チェックを行う基本単位となる．構築を行う場合も，証明チェックを行う場合も利用者は理論名で指定する．"理論群"は，互いに参照できる理論の集まりである．利用者は，本システムを立ち上げる際に，取り扱うべき理論群を指定する．この指定は，あらかじめ作成したコマンド・ファイルに書かれている理論群名を通して行われる．証明チェックの対象となる理論は，証明エディタ上に呼びだして，構築する．また，ブラウザとよばれるエディタを開くことによって，他の理論を参照したり，構築したりすることができる．

②証明チェックとデバッグ

証明エディタ上に呼び出された理論は，証明チェッカにかけることができる．チェッカでエラーが発見された場合，システムはデバッグ・モードになる．デバッグ・モードで

図2　証明記述の構成

は，エラー・メッセージがエラー・ウィンドウに表示されると同時に，発見された箇所がエディタの画面中央に来る．エラー箇所が複数の場合は，順番にエラー番号が付けられるので，利用者はそれを指示することによって任意の箇所のエラーを選択する．表示された箇所のテキストは，そのまま編集して修正することができる．（図3）

修正した証明は，再度チェッカにかけて，エラーが無くなるまで修正を加える．

③理論の保存と参照

チェックの済んだ理論は，ルール形式の内部表現をセーブすることによって，保存することができる．この理論で証明されている定理は，理論名を指定することによって他の理論から参照することができる．その際は，証明チェックの前に，証明記述に添えられている理論名を持つ理論すべてが，あらかじめ自動的にロードされる．

セーブされた理論は，理論の展開の無矛盾性を保証するために，保護がかけられる．この保護がかかると，テキストを編集してセーブすることと，証明を再度チェックし直すことはできない．ただし，利用者の要請により，この保護は解除することができる．ある理論の保護が解除されると，ルール形式の内部表現が削除され，この理論に含まれる定理が前提になっているすべての理論の保護が，解除される．（図4）

④その他の機能

以上に加えて，次のような機能も提供されている．

・理論群に含まれるすべての理論の名前の一覧を表示する機能

・ルールとして登録され、保護のかけられているすべての理論の名前の一覧を表示する機能

（　この2つの一覧はメニューとなっており，直接指定するために利用できる．　）

・一つの理論の上位，下位の理論の参照関係を表示し，関係をたどる機能

また，参照関係がループしている理論群は，構成することができない．

4．まとめ

ＣＡＰ－ＬＡシステムの証明記述支援環境の機能は，次のようにまとめられる．

①理論展開の無矛盾性の保証を自動的におこなう機能

②テキストとルールの一貫性を保つ機能

③利用者が，以前に証明した定理を利用するとき，証明記述の上だけで指定できる機能

④チェッカの返したエラー情報を，直接エディタ上で追いながら訂正することができる機能

⑤利用者が，自分の展開した理論の構造の情報を得ることができる機能

今回の試作では，デバッカの機能を除いて，以上のような機能を満たす環境を開発することができた．

5．おわりに

今後の課題として，定理レベルでの証明チェックと参照関係の表示機能があげられる．より小さな単位である定理の証明のつみかさねによって，理論を構築してゆくことができれば，システムはより"ノート"のイメージに近づいてくるからである．その際には、エディタとシステム・コントローラが，一層高度なやりとりを行うことが必要になってくる．

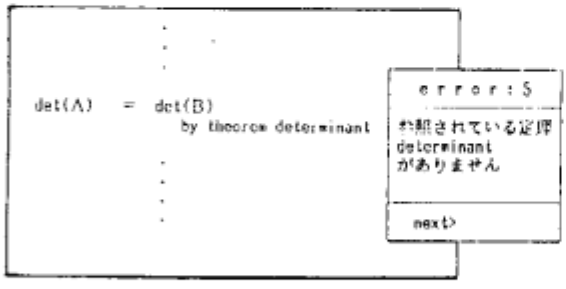
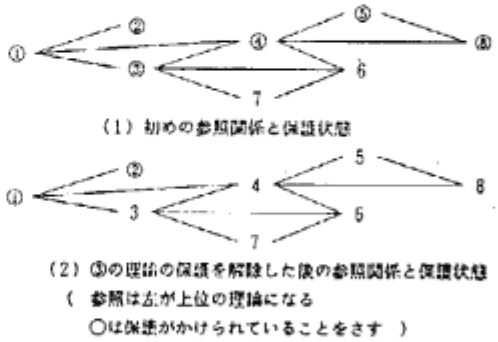


図3　エラーの出力例



（1）初めの参照関係と保護状態

（2）③の理論の保護を解除した後の参照関係と保護状態

（　参照は左が上位の理論になる

○は保護がかけられていることをさす　）

図4　保護の解除の伝播

CAP プロジェクト(3)
7L-3 ―証明記述言語―

堀 雄彦	坂井 公	西山 聡
(立教大学)	(ICOT)	(三菱総合研究所)

1. はじめに

本稿では，CAP (Computer Aided Proof)プロジェクトで開発中のCAP-LAシステムのうち，証明記述言語(PDL) の概要について述べる．

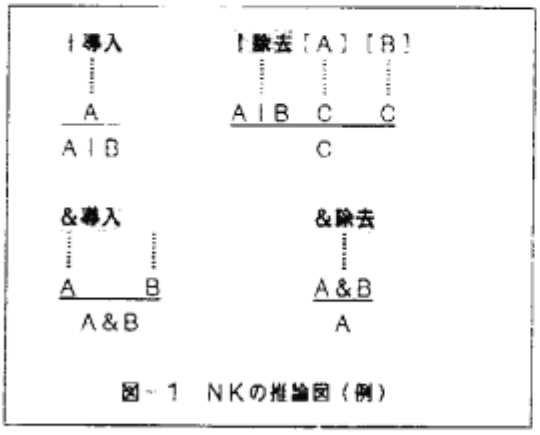
PDL は，自然演繹システム[1, 2]の証明図を1次元的に表現したものに線形代数[3] に特有な諸概念をつけくわえたものになっている．

2. PDL の設計方針

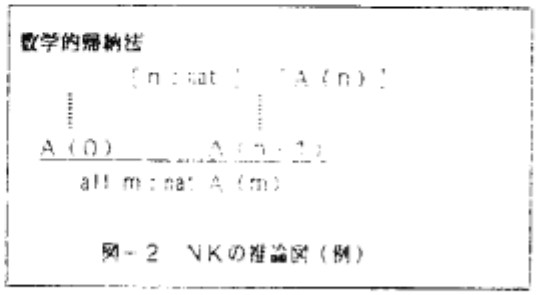
PDL は，計算機と人間を結ぶインタフェースの核であり，計算機に理解できるように形式化されていなければならない一方，人間にとっても読みやすく書きやすい言語でなければならない．さらには，数学の証明を十分表現するだけの記述力を持たねばならない．

このため我々は，形式的体系としては人間の行う推論過程を最も忠実に反映していると思われる自然演繹システム，とくにその代表といえるゲンツェン(Gentzen) のNKを基にして言語を設計した．NKは，論理的に完全であることがわかっているという点でも我々の目的に適している．

NKは図－1の例のように各論理記号に関する導入と除去という2種の推論図で構成されている．&（連言），∨（選言）以外にも ￢（否定），→（含意），＝（等号）や all（全称），some（存在）などに関する推論図がある．一般の数学とくに線形代数を展開するためには，これでは不足なので次のものに関する推論図を追加した．



図－1 NKの推論図（例）



図－2 NKの推論図（例）

```
∨導入
:
A
hence A∨B
∨除去
:
A∨B
hence C
since divide and
conquer A∨B
   case A
      :
      C
   case B
      :
      C
end-since
&導入
:
A
:
B
hence A&B
&除去
:
A&B
hence A
```

図－3 PDL （例）

- 数学的概念…自然数，有限列，有限和（Σ），有限積（Π）など
- 線形代数特有の概念…スカラ，ベクタ，行列，置換など

例えば図－2は数学的帰納法を表現する推論図である．

また，数学的帰論の展開のためにはしばしば2階以上の推論が必要なので，限られた範囲でこれらをも扱えるように拡張した．しかし，2階の述語を一般的に扱うのは，かなりの技術的困難が予想され，当面は考えていない．

PDL は，この拡張されたNKの証明図を1次元的に表現したものである．

3. NKとPDL の対応

PDL には，原則としてNKの各推論図に対応する証明記述のための構文が用意されている．図－1，図－2の例に対応するPDL の構文を図－3，図－4に示す．

図3の構文のうち，∨除去が他の構文よりきわだって複雑である．これは，図－1の∨除去の推論図で仮定の除去(discharge) が起っていることが起因している．つまり[A]，[B]のように［ ］

```
数学的帰納法
all m : nat  A (m)
since induction on  m
 base
       :
    A (0)
 step
  let n : nat be
  such that A (n)
       :
    A (n + 1)
end-since
```

図　4　PDL （例）

でくくられた式は，以下のNKでの証明で用いることができなくなる．この制限をPDLで自然に表現するために，我々はsince とend-since にまるブロック構造を導入し，さらにそこで使われている推論法則を明示するためのキーワードを採用した．例えば！除去の場合のキーワードはdivide and conquerであり，case Aからcまでと case Bからcまでの証明は，それぞれブロックを構成する．あるブロックの中で設定されたり，証明されたりした命題は，そのブロックの外から参照することは許されないので，NKでの仮定の除去に相当することがPDLでも実現される．

また！除去の構文は，次のように読めるので，人間の推論過程をかなりよく反映しており，人間が読み書きする場合，自然とはいえないまでも，あまり違和感を感じずにすむと思われる．

「…によりAまたはBである．従ってCである，というのは，Aの場合…によりCであり，Bの場合も…によりCであるから，」

次に数学的帰納法の推論図であるがこれに対してはinduction onというキーワードを採用しbaseからA（0）までとstepからA（n＋1）までをそれぞれブロックにしている．推論図の中に現れる自由変数nは，n：nat とA（n）以外の仮定の中や結論のall m：nat A（m）のmには現れてはならないという制限があるが，これもブロック構造の採用により簡単に実現できる．つまりlet n：nat be … で導入される変数nは，常に新しい変数でなければならず，その有効範囲はブロックの中にとどまるからである．

一般にsince end-since ブロックによる証明は結論を先に述べ証明が続く後向き推論（トップダウン）になり，それ以外は結論が最後に来る前向き推論（ボトムアップ）になるが，いくつかの実例にあたってみた限りでは，この推論方式は人間にとって自然であるようである．

4．PDL の特徴と将来の課題

さらに PDL1版には長短含めて次のような特徴がある．

- 型付き(strongly typed)な言語であり，型はパラメータを持てるが，型変数はない．
- ベクタの型はない．ベクタは，1行または1列の行列としてユーザが定義する．しかしスカラは1行1列の行列とは考えず，固有の型をもつ．
- 型の論理的な扱いは他の論理式と同じとする．
- 変数，定数は記述の上では区別されない（英字で始まる英数字列）．
- 束縛記号（all，some，Σ，Πなど）で束縛されている変数だけが束縛変数であり，他は自由変数や定数である．自由変数と定数の論理的扱いは同じである．
- ユーザは，必要に応じて定数や関数を定義できる．定数はパラメータの無い関数とみなし，関数定義と同じ構文を用いて定義する．
- 記述記述を自然にするため，ある種の接続詞，形容詞の使用を可能にするが，検証には利用しない．
- 証明済の定理は，名前で参照することができる．また，証明が完全でない場合にも，部分的な検証を可能にするため，完成していない部分は検証なしで済ますような記法が書ける．

我々は，現在 PDL2版の設計を開始したところであるが，新版の設計における中心課題は次の諸点となるだろうと考えている．

- 2階以上の概念のより統一的な扱い．
- 型変数又はgeneric typeの導入，型情報の検証への利用，型の推定
- 証明戦略のユーザによる導入，又は自動抽出
- 形容詞，接続詞等，ユーザによる付加情報の検証への利用
- if …then …elseなどの論理記述の導入とその検証への利用

参考文献

[1] D. Prawitz: Natural Deduction, Almquist & Wiksell, 1965

[2] 鳥内　剛一：数学の基礎，日本評論社，1971

[3] 古屋　茂：行列と行列式，培風館，1957

7L-4

CAPプロジェクト(4)
－証明の検証－

永田守男　　高山幸秀　　西山聡
(慶応義塾大学)　　(ICOT)　　(三菱総合研究所)

1. はじめに

現在，CAPプロジェクトでは，数学的な素養のある人であれば容易に利用することが可能な証明検証システムであるCAP-LAシステムの開発を，逐次推論マシンPSI上で行っている．

本稿では，CAP-LAシステムの中で与えられた証明の論理の展開の正しさの検証を実際に行う証明チェッカの構成とその処理内容について述べる．

2. 証明チェッカの構成

CAP-LAシステムの証明チェッカは，図1に示すように変換部，推論部，ルール生成部，数式処理部，および項書換え部の5種類のモジュールから構成される．

変換部は証明エディタとのインタフェースとなっており，証明エディタが扱うデータ構造である構文木から推論部が扱う証明木への変換を行うモジュールである．

推論部は証明チェッカの本体で，証明木を走査することにより与えられた証明の論理展開が，自然演えき論理(NK)に従ったものであるかどうかの検証を行う．

ルール生成部は検証の終了した定理から推論部や項書換え部が利用することのできるルールの形式への変換を行って理論知識ベースへ引き渡す．

また，数式処理部は論理式中に現れる数式を等価変換するときに利用されるもので，線形代数に特有な演算機能や，有限和(Σ)・有限積(Π)に関する記号処理機能を持っている．

なお，項書換え部では，式の等価性に関する検証を行う．

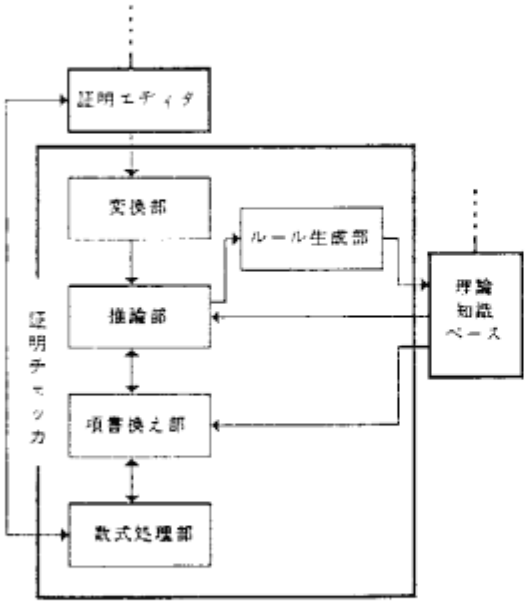


図1.証明チェッカの構成図

3. 証明検証アルゴリズム

本推論部での証明検証は，記述された証明のパタンによって定まる戦略(証明の図式)に従って推論規則を適用することにより行われる．この戦略というのは，連言(&)，選言(∥)，含意(→)，限定子(all，some)などの論理記号を含む論理式を対象とした検証についての手順をアルゴリズムとして，システム内に組み込んだもので，推論規則の適用方法を記述したメタな規則ということができる．したがって，この戦略については変更することはできない．

いっぽう推論規則は，検証の終了した定理や証明中で仮定された論理式，および論理展開の中で成り立つことが明らかになった論理式などをルール化したものである．検証済みの理論に含まれるルールは理論知識ベースに蓄えられており，仮定などのように検証時に明らかになる情報については，チェッカによって随時，一時的な推論規則への変換が行われる(推論規則の生成については4で述べる)．

こうした戦略や推論規則を使ってどのような形で検証が進むかを，例を用いて説明する．例として，図2(a)に示すPDLの記述を検証することを考える(ここで，A，Bは論理式をあらわしている)．証明の検証を行うさいには，図2(a)のPDLによる証明の記述は図2(b)で与えた証明図の形に変形されている．通常の場合，証明図は構築の上から下へと読むが，本推論部では逆に下から上へとたどることによって証明の検証を行っている．

図の例のA＆Bという論理式の場合，「＆でつながれている2つの論理式の検証は，それぞれがその環境で成り立っていることを検証すればよい」という戦略を推論部が持っており，これを利用することにより，AとBのそれぞれをゴールとする2つの証明の検証に分割することができる．分割された部分の検証も同様にして行われ，今度はA(およびB)を証明図の下の部分に持つ戦略，あるいは推論規則の探索を行い，見つかった場合にはその証明図の上の部分にある論理式をゴールとする証明の検証に分割する．見つからなければ，その論理の展開に誤りがあることになる．この操作を繰り返して，証明図の上の部分を持たない規則に達すれば，その証明の正しさの検証が行われたことになる．

したがって本推論部では，推論規則による後向き推論で，証明の検証を行っているということができる．

```
A & B
since
    A
    B
    hence A & B
end_since
```

(a) PDLで記述した証明例

$$\frac{A \qquad B}{A \,\&\, B}$$

(b) (a)を証明図に変換したもの

図2 PDLと証明図の対応

証明の検証メカニズムをもう少し一般的に説明すると，以下のようになる．たとえば，

$$\frac{\begin{array}{ccc}\cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot\\ D & E & F\end{array}}{C}$$

という証明図が与えられた場合には，まず論理式D, E, Fが成立するときに論理式Cを導く戦略が存在するかどうかを調べる．戦略が存在する場合は，論理式D, E, Fを含む証明図がそれぞれ成立するかどうかの検証を行う．

戦略が存在しない場合は，論理式D, E, Fを含む証明図の検証が終了した環境の下で，推論規則を複数回適用することにより，論理式D, E, Fから論理式Cが導けるかどうかの検証を行う．

検証が終了した証明図の中の下端にある論理式は推論規則に変換され，以後の証明を検証するための環境を構成することとなる．

4. 推論規則の生成

これまでの説明でわかるように，本推論部で推論規則がはたす役割は非常に大きなものとなっている．しかもPDLで書かれた証明中に現れる仮定や，検証の終了した個々の論理式は推論規則に変換し，以後の検証の過程で利用可能なものにする必要がある．以下では，この推論規則の生成アルゴリズムについて説明する．

推論規則の生成は，対象としている論理式の種類により以下の手続きに従って行われる．

(1) A→B の場合

$$\frac{}{A \to B} \quad \text{および} \quad \frac{A}{B}$$

(2) A&B の場合

$$\frac{}{A \,\&\, B} \quad \text{および} \quad \frac{}{A} \quad \text{および} \quad \frac{}{B}$$

(3) A|B の場合

$$\frac{}{A \mid B}$$

(4) \A の場合

$$\frac{}{\backslash A} \quad \text{および} \quad \frac{A}{\bot}$$

このとき

$$\frac{\backslash A}{\bot}$$

という推論規則が生成された場合は，

$$\frac{}{A}$$

という推論規則も追加する．

(5) allV.Aの場合

$$\frac{}{\text{all}\,V.A} \quad \text{および} \quad \frac{}{A'}$$

(ただし，A'はAの束縛変数Vを自由変数で置換えたもの)

(6) someV.Aの場合

$$\frac{}{\text{some}\,V.A}$$

さらに，生成された各推論規則の前提部(推論図の線より上の部分)が次の論理式の場合は，以下の手続きで得られる推論規則も生成する(いずれの例の場合も，帰結部にはBが書かれているものとする)

(1) someV.A の場合

$$\frac{A'}{B}$$

(ただし，A'はAの束縛変数Vを自由変数で置換えたもの)

(2) $A_1 | A_2$ の場合

$$\frac{A_1}{B} \quad \text{および} \quad \frac{A_2}{B}$$

(3) $A_1 \to A_2$ の場合

$$\frac{A_2}{B} \quad \text{および} \quad \frac{\backslash A_1}{B}$$

5. おわりに

現在CAP-PDLシステムは開発中であり，本証明チェッカについても実行効率等に関する評価は行われていない．また，本チェッカの推論部で採用した後向き推論による検証アルゴリズムについても，その妥当性・完全性に関する考察が十分に行われているとは言えない．

これらの点について今後検討していく必要がある．

7L-5

CAPプロジェクト（5）
―― 証明支援ユーティリティ ――

坂井 公　　小久保 岩生
（ICOT）　　（三菱総合研究所）

1．はじめに

本稿では，CAPプロジェクトにおいて開発中のCAP－LAシステムのうち証明支援ユーティリティについて述べる．証明支援ユーティリティは，ユーザが証明記述言語（PDL）により証明の記述・編集を行うエディタを提供する．エディタは，汎用的な編集機能に加え，線形代数理論の編集に適した機能を持つよう設計されている．

2．エディタの構成

CAP－LAシステム全体の構成のうち，証明エディタ，ブラウザの2つがここで述べる証明支援ユーティリティにあたる．

証明エディタは，システム起動時から存在し証明チェッカに接続している唯一のエディタである．これに対してブラウザは，ユーザがCAP－LAシステム・コントローラを通していくつでも生成することができ，証明エディタと同様の編集を行えるが，証明チェッカには接続していない．

ユーザは，証明チェッカにかけたい理論を証明エディタを用いて編集し，必要に応じてブラウザを用いて他の理論を参照したり，またその一部を証明エディタで編集中の理論上へ複写したりすることなどにより，証明の記述を進めていくことになる．

証明エディタ，ブラウザのエディタとしての機能は全く同じで図1のような構成となっている．ユーザは，テキスト・エディタと構造エディタをモードを変えることにより利用できる．以下で両者の目的と機能を述べる．

3．テキスト・エディタ

テキスト・エディタは，ICOTのSIMPOSのエディタを継承して開発した〔1〕．テキスト・エディタは文字列を対象とした編集を行うため，文法的な制約を受けずに編集を行いたい場合に使用する．

4．構造エディタ

構造エディタは，テキスト・エディタにより生成されたテキストを構造エディタ用の構造（構文木）に変換し，その構文木に対して構文規則に沿った形で編集を行うエディタである〔2〕．構造エディタ開発の目的は次の2点である．

- 編集する単位と文法的な構造の単位の関係を明確にし，線形代数の定理証明中でよく現れる等式の書換えの記述などを正確かつ速やかに行えるようにする．
- 構文木の持つ構文情報を用いることにより，文法的なガイダンス機能などのユーティリティを提供する．

(1) 構造エディタの構成

図1で示した構造エディタの各部分について説明する．

字句解析部

文字列を，構文要素（トークン）に分割する（SIMPOSのトークンリーダ〔1〕を使用）．

構文解析部

字句解析部が生成した構文要素の列に対して，構文解析を行い構文木を生成する．構文解析にはBUPジェネレータにより生成されたパーサ〔3〕を使用している．構文木

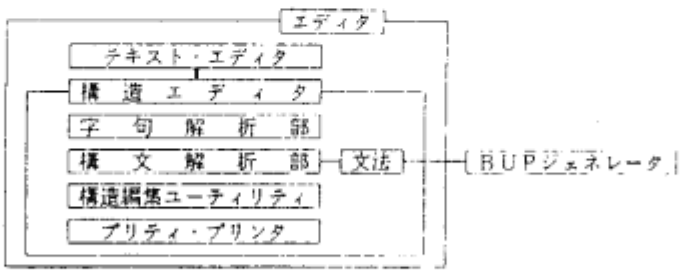


図1　エディタの構成

の構造はＢＵＰジェネレータに与える文法により決定されるが，その文法中にはＰＤＬ言語の構文情報の他，構造編集・清書（プリティプリント）の際に使用する情報も組み込んである．また証明チェッカに渡す証明記述のデータはこの構文木であり，証明チェッカに入る前に必ず１度は通るモジュールである．

構造編集ユーティリティ

構造エディタの核にあたる部分で，ユーザのコマンドに従って構文木の編集を行う（編集方法は後述する）．

プリティプリンタ

構文木からテキストを復元しディスプレイ上に清書する．構文木を深さ優先順に探索し，その末端の節（葉）の部分に書かれた終端記号をプリントしていくことを基本としている．その際の改行・字下げなどの制御も，構文木中の情報を参照することにより行われる．

(2)　構文木の構造

構文木は，＂名前＂，＂型＂，＂部分木リスト＂の３つ組により表される．この３つ組１つが構文木の１つの節とその節以下の部分構文木に対応している．

＂部分木リスト＂は，その節以下の部分構文木を要素とするリストであり，これにより構文木の親子関係が表現される．

＂名前＂は，構文木中のすべての節につけられることになるが，ある部分木だけを編集したときに，その部分木の構文木を再び構成する際の始点の情報として必要になる．

＂型＂には，その節以下の部分木の削除が許されるか否かなど文法的な情報の他，清書の際の改行・字下げなどに関する情報も含まれている（図２）．

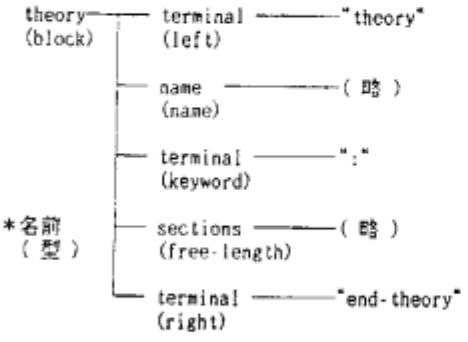


図２　構文木の例

(3)　構造編集の方法

構造エディタは構文木の編集を行う．テキスト・エディタのカーソルに対応するものとして構造エディタでは編集の単位としてターゲットというものを考える．ターゲットは，構文木中の１つの部分木に常に対応し，画面上ではその部分木に含まれる終端記号が長方形の枠で囲んで示される（それが可能となるようプリティプリントされている）．ユーザはそのターゲットに対して，削除，挿入，移動，複写などの命令を送る．これを受けて構造エディタは，ターゲットの対応する節の＂型＂を参照し，その命令の実行による構文木の編集が文法的に正当であると認めた場合にのみ，その命令を実行する．

以上のような構造エディタとしての一般的な機能の他，次のような線形代数の定理証明の記述・編集に適した機能も検討中である。

- ビットマップディスプレイを利用した行列などの表示（２次元プリント機能）
- 項書換え部，数式処理部を利用した，証明中の等式書換え作業の支援
- 証明チェッカと連動したデバック作業の支援
- ＰＤＬ言語に不慣れなユーザに対する文法的なガイダンスを主とするヘルプ機能

5．おわりに

最後に述べた検討中の事項の実現が，今後の課題である．一般に構造エディタは，テキスト・エディタほど操作が手軽に行えないという難点もある．構造エディタ自身のユーザインタフェースの充実，そのための他モジュール（数式処理部，証明チェッカなど）の利用などを中心に，使いやすいシステムとなるよう今後も研究・開発を進めていく予定である．

参考文献

〔1〕　ＳＩＭＰＯＳ使用説明書（第一版），ＩＣＯＴ，６０年４月．

〔2〕　佐藤・坂井：　演算子文法用汎用構造エディタ，第２７回プログラミング・シンポジウム報告集，ｐｐ．63-73．

〔3〕　奥西・平川：　SIMPOSのプログラミング・システム――ＢＵＰのＰＳＩへの移植実験――，情報処理第３０回全国大会論文集，ｐｐ．309-310．

7L-6

CAPプロジェクト（6）
—等式の検証—

大須賀 昭彦　　藤瀬 哲朗
（ICOT）　（三菱総合研究所）

1．はじめに

項書換えシステム(TRS)とは，与えられた等式を方向付された書換え規則とみなして項の等価な変換（書換え）を行うシステムで，等式を含んだ論理に対し簡明で比較的効率の良い計算を与える手段として，近年定理証明やプログラム変換・合成等への応用がさかんに研究されている．CAPプロジェクトに於いても，TRSの数式処理や証明記述支援等への応用可能性に着目し研究を続けているが，CAP-1Aシステムに，TRSに基き等式のチェックを行う項書換え部（TRS1.0版）を実装したので，その機能概要を報告する．

2．TRSの定義

以下に一般的なTRSの定義を与える．等号論理Eが与えられたとき，{l1=R1,...,ln=Rn}と十分等価となる様な書換え規則の（有限）集合 R={l1→R1,...,ln→Rn} をEおよびTRSという．

項tが部分項sを持つことをt[s]と表現すると，部分項sと書換え規則 l→R との間にs=θlなる代入(substitution)θが存在しt[s]からt[θR]が得られるとき，t[s]はt[θR]に簡約(reduction)されたといいt[s]→t[θR]と書く．また部分項sの様に書換え規則が適用可能な部分項を，可約部(redex)と呼ぶ．項tがそれ以上簡約できないとき，tは既約(irreducible)であるといいt↓と書く．

Rにおいてt0→t1→...なる無限の簡約が存在しないとき，Rは停止性(termination)を持つという．またO回以上の簡約で項tがuに到達することをt→*uと書くと，任意の項tに対しt→*u，t→*vならばu→*w，v→*w なる項wが必ず存在するとき，Rは合流性(confluency)を持つという．停止性と合流性を同時に持つTRSを完備な(complete)TRSと呼ぶ．

完備なTRSに於いては，項tから求まる既約項は唯一であり，これをtの正規型(normal form)という．このとき，tから任意の簡約によって正規型に到達することができる[3],[5]．

3．CAPシステム構成

図-1に項書換え部(TRS)周辺の構成を示す．

推論部は，証明全体のチェックを行い，証明中に等号で展開された記述を見つけると，式中の変項を自由変項化した後TRSに渡す．

ルール作成部は，推論部によって正しさがチェックされた定理類をルールの形に変換して，知識ベースへ保存する．

理論知識ベース部は，(1) 公理，(2) 証明済定理の他に，(3) 証明の仮定，(4) 証明中で一時的に導かれた事実等，TRSが書換え規則として使用できるルールと，その成分表現等を保存する．但しこれらは，方向を持たない等式として保存されている．

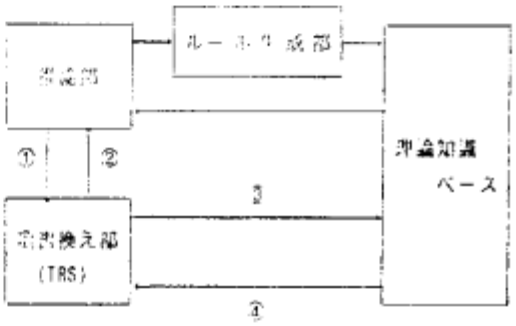


図-1.CAPシステム構成

4．TRSの機能

4.1 TRSの戦略

完備なTRSに於いては，基となる等号論理の上で等価な項は必ず等しい正規型に簡約され，等価問題は決定可能となるが[3],[5]，一般に数学で扱う定理の集合（ルール）が完備であることは少ない．

停止性を持つTRSに対して，合流性を組い完備なシステムを生成するアルゴリズムとして，Knuth & Bendixの完備化手続き(completion procedure)[4]等もあるが，このアルゴリズムは，(1) 停止しない，(2) 等式に対する方向付ができない等の理由で失敗することがあり，特に結合，交換律を満すルールをうまく扱えないため，応用は困難である．

そこでTRS1.0版では，完備でないシステムを前提として，以下の点を考慮する．

(1) 停止性の保証

TRSの停止性を保証するために，次の定義を行なう．

・具象項(ground term)間で全順序となる単純化順序を強単純化順序(strong simplification ordering; 以降

>>で示す）というと，この順序は整礎(well-founded)順序となる[2]

・具象項t[s]と等式 l=R との間にs=θl，θl >>θR が成立するとき，t[s]はルール l=R によってt[θR]に簡約されたという．

この簡約を TRSへ応用することにより，TRSが決定性を持つことは明らかである．

(2) 可約部の選択

合流性を持たない TRSでは，任意の項から導かれる既約項は一意に定まらず，可約部の選択に依存して異る既約項に到達する可能性がある．しかし停止性が保証されていれば，すべての既約項をバックトラック等の機能によって求めることは容易である．

4.2 チェックアルゴリズム

以上の戦略に基き，TRSは次の手順で等式のチェックを行なう．

・まず推論部から等式を受取ると（図-1①），スコーレム化を行ない具象項を得る．

・理論知識ベースから適用可能なルールを検索し（②④），等式両辺のすべての可約部に，ルールによる簡約を適用して既約項を求める．

・すべての既約項が求まり，両辺の既約項の中に等しい項が現れたとき，等式成立として等式の成立条件等を推論部へ返す（③）．

尚，簡約を行なう際の項単純化順序には，オペレータ間に適当な順序を規定した辞書式部分項順序(lexicographic subterm ordering)[5] を採用した．

5. 実行例

次の様な線形代数の問題を考える．

> (m,n)行列 Aの行と列を入れ換えてできる (n,m)行列を Aの転置行列といい，t(A)で表わす．A,Bがともに (m,n)行列であれば，t(A+B) t(A)+t(B)となることを証明せよ．

前提を前置型に変形し，変項を自由変項化すると次式(Th)が得られる．（ matは行列を示す．またアルファベット大文字は変項，小文字は定項を示す．）

Th: mat(t(A+B),I,J) = mat(t(A)+t(B),I,J)

ここでルールとしては，(I1) 転置行列の定義，(I2) 行列の加算の定義のみが与えられているものと仮定する．

I1: mat(t(X),M,N) = mat(X,N,M)

I2: mat(X+Y,M,N) = mat(X,M,N)+mat(Y,M,N)

TRSでは，等式Thを以下の手順でチェックする．

(1) Thをスコーレム化してTh' を得る．

Th':mat(t(a+b),c,d) = mat(t(a)+t(b),c,d)

(2) オペレータ間の順序を規定する．

mat > + > t > a > b > c > d

(3) Th' の両辺を簡約しすべての既約項を求める．

Th' の左辺にE1を適用すると，

θ={ a+b/X, c/M, d/N } によって

s=mat(t(a+b),c,d)+θl,

θl=mat(t(a+b),c,d)>> mat(a+b,d,c)= θR

となり，ルール[1]によって，mat(t(a+b),c,d) → mat(a+b,d,c) なる簡約が行なわれる．同様にE2によってmat(a+b,d,c)→ mat(a,d,c)+mat(b,d,c)」となり既約項が得られる．

同様な操作によって，すべての既約項を求めると，

左辺 →* mat(a,d,c)+mat(b,d,c)」

右辺 →* mat(a,d,c)+mat(b,d,c)」

(4) 両辺の既約項を比較する．

この場合両辺の既約項は一致しThは証明される．

6. おわりに

TRS1.0版については現在評価を続けている．第2版以降の課題としては以下のものが挙げられるであろう．

・完備化手続きの導入

・失敗しない完備化手続きの導入[2]

・結合，交換律ルールの扱い[6]

・TRS が持つべき自然数，有限集合（2），有限組（f1）等に関する知識の検討

・タイプの導入，等式の同所化等による効率の向上

・数式処理，証明記述支援等への応用検討

参考文献

[1] N.Dershowitz, "Termination", lecture Notes in Comput.Sci.202,Springer-Verlag, PP.180-224,1984

[2] J.Hsiang, "On Word Problems in Equational Theories -fxtended Abstract-",1985 (private communication)

[3] G.Huet and D.C.Oppen,"Equations and rewrite rules : A survey",in R.Book(Ed.), Formal Languages : Perspective and Open Problems, Academic Press,pp.349-393,1980.

[4] D.Knuth and P.Bendix,"Simple Word Problems in Universal Algebras", in J.Leech(Ed.), Computational Problems in Abstract Algebra, pp263-297,1970

[5] K.Sakai, "An Ordering method for term rewriting systems", ICOT TR-062,1984

[6] 高山,"ACユニフィケーション・アルゴリズムとその実装"，理研シンポジウム －関数プログラミング－ ,Jan.30,1986