ICOT Technical Memorandum: TM-0167

TM-0167

構成的証明に基づくプログラム検証・変換・合成システム

高山幸秀、坂井　公、横井俊夫

April, 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

構成的証明に基づくプログラム検証・変換・合成システム

高山　幸秀，坂井　公，横井　俊夫

April ,1986.

目次

(1) はじめに

ソフトウェア開発支援システム構築のための基幹技術として，三菱中研の金森グループを中心に論理型プログラムの等価変換システムの研究開発が既に進められてきた．論理型プログラムの検証・変換・合成技術としての，等価変換技術の重要性は言うまでも無い．しかしながら，プログラムを一階述語論理で書かれた仕様に対する構成的証明とみなす観点の技術研究(構成的証明に基づくプログラム検証・変換・合成技術)も一つの重要な流れであり，論理型プログラミングの立場からみてこれを見逃す訳にはいかないと思われる．そこで，従来の研究開発の流れを踏まえて，ICOT知的プログラミング・グループのプログラミングに関する研究は，次のような方針で行うことを計画している．

(1) 基幹技術のもうひとつの流れとして，構成的証明に基づくプログラム検証・変換・合成技術の研究グループを新たに立ち上げ，基礎的検討，システム化検討，部分的設計・試作を行う．
(2) 三菱中研グループと密接な技術的交流を図ることによって，等価変換技術と構成的証明に基づくプログラミング技術との関連を検討する．
(3) さらに，この検討を通じて等価変換技術をベースにしたソフトウェア開発支援システムのイメージを具体化する．

(1) はICOT側を中心に，(2)はICOTと三菱中研が中心になって行う．また(3)は三菱中研を中心に行う．なお,61年度はシステム作りの基本的検討および一部モジュールの試作・実験を中心に行い本格的な設計・実装は62年度以降に行う．最終的には，2つの技術がバランス良く融合あるいは役割分担されたトータル・システムの技術の開発を目標とする．

本論文は，我々が研究を始めている構成的証明に基づくプログラム検証・変換・合成システム位置付け，技術内容，システム・イメージなどに就いて概説し，この方面の研究内容の全体像を紹介することを目的としている．なお，本論文の内容は，ICOT知的プログラミング61年度計画作成のための議論を通じてまとめ上げられたものであり，ICOT第4研究室横田一正主任研究員および佐藤雅彦東北大学教授(前東京大学助教授)には幾つかの有益な助言を頂いた．

(2) 構成的証明に基づくプログラミング技術とその周辺

(1) "構成的証明 = プログラム"のパラダイム

1) 思想

・プログラミングと数学の定理証明の類似性は古くから指摘されており，ICOT知的プログラミング・グループで行われているCAPプロジェクトも基本的にはこの観点に基づくものである．
・定理の証明とプログラムの大きな相違点の一つとしては，プログラムの方がより手続き的なことを多く記述するのに対して，証明の方は手続きの記述が無い場合があることである．例えば，ある集合に対してその最大値が存在することを証明する場合，最大値の構成方法を示さないで証明することが多い．
・しかしながら，"証明は構成的でなければならない"という条件を付けると，定理の証明はきわめてプログラムに類似したものとなる．上の最大値の例で言えば，図-1のような関連が付けられる．

⟨⟨定理⟩⟩ "集合Aの最大値が存在する"	⟨⟨仕様⟩⟩ "手続きPは集合Aの最大値を計算する"
⟨⟨構成的証明⟩⟩ 最大値の構成方法を示すことによる証明	⟨⟨プログラム⟩⟩ 最大値を計算する（構成する）プログラム

図-1:: 構成的証明とプログラムの関連

ここで⟨⟨仕様⟩⟩には，手続きの内容(アルゴリズム)は示されていないと考える．

・プログラミングを定理の証明として行うことの利点は，
1) 数学的素養を持った人ならば，特殊なプログラミングの知識，とくにプログラミング言語の繁雑な文法や書法をそれほど勉強しなくてもプログラミングに入っていける．すなわち，人間の論理的思考により近い形で記述できる．
2) 通常の高級言語よりも高いレベルの記述ができるので見通しが良い．例えば，Prologよりもよりロジックに近いあるいはロジックそのもので記述できるのでプログラムの論理的構造が分かりやすい．
3) プログラミングと，プログラムの正当性の確認が一体になっている．すなわち，人間がプログラムの正当性を確認しながら記述することができる．
4) 証明検証系や自動証明系の技術と直接的に関連するので，プログラムの検証・(半)自動合成機能を持ったプログラミング環境構築の手掛かりが掴みやすい．

図-2 に証明としてのプログラミングと通常のPrologプログラミングとの対応表を示す．

	証明としてのプログラミング	通常のPrologプログラミング
記述レベル	・一階述語論理あるいはそれにほぼ一致するもの	・Horn論理(一階述語論理の部分クラス) + 制御機構
検証情報	・プログラム(証明)そのものが検証情報になっている	・別途付加する必要あり
検証方法	・証明検証系によりほぼ自動的に行われる(静的検証)	・デバッガを使って具体的なデータについて正しく動作することを確認する (厳密には検証ではない)
プログラマに要求される知識・素養	・数学的素養 ・データ構造に関する初歩的な知識 ・正しい論理の筋道を把握できるだけの注意力	・バックトラック，ユニフィケーション，カットなどのProlog特有の制御構造 ・論理の筋道以外の細部に対す注意力(デバッグ時など)

図-2:: 通常のPrologプログラミングとの比較

但し，ここでいう検証には仕様そのものがプログラマの意図に沿うものがどうかをチェックする操作は含まれない．たとえば，形式的仕様のレベルである程度の実行が可能な記述系を考え，例題を実行してその結果を調べることによって仕様自体がプログラマの意図に沿ったものかどうかチェックする機能が考えられるが，それは構成的証明によるプログラミングのパラダイムとは一応独立なものと考えられる．

また構成的証明とかそれを記述する形式的体系の基礎である直観主義論理の場合，時制を考慮した体系は今のところまだ提案されていない．従って，それをもとにしたプログラミング・システムでも時制論理は考えない．従って通信システムやプロセス概念の入ったプログラムの開発支援は現在のところこのパラダイムの範囲外である．

2) realizability interpretation と realizer

・realizability interpretationとは，構成的証明の証明図を解析してプログラムを導出する手続きであり，今世紀半ばにKleeneによる形式的演繹を部分帰納的関数で表現する手法の研究に端を発する．realizerとはrelaizability interpretationによって導出された実行可能コード，つまりプログラムのことを指す．すなわち，証明の記述を高級言語によるプログラミング，realizability interpretationはそのコンパイラ，realizerはターゲット・コードだと思ってよい．以下，構成的証明に基づくプログラミング技術とは構成的証明とそのrealizability interpretationを中心としたものを指すものとする．

・証明を構成的に行うという考え方は，実は直観主義論理と深くかかわっている．実際構成的証明を記述するための論理体系としては直観主義に基づくものを用いる．論理の筋道だけを公理化した古典論理の形式的体系と比べて，構成的証明を記述するために作られた形式的体系は，if-then-else，場合分け，λ-式，対化(pairing)，射影

(projection) など，プログラミングの言葉で言えば，手続きを具体的に記述するための概念が付け加えられており，それらに対する推論規則を与えるところに特徴があると言える．

・上にのべた，if-then-elseなどの概念に対応するものは，プログラミグ言語またはそのサブセットで与え，それをrealizerとして使用する．realizability interpretationとは，証明の筋道すなわち古典論理で扱われているand，or，implication，exist，allの構造に注意しながらrealizerを一つ一つ組み合わせていく操作だと言える．

・例として，EON (Elementary theory of Operations and Numbers)という構成的数学記述のための形式的体系を考えよう．ここでは，EONの公理系を詳しく列挙することは省略するが，Herbert & Ackermanの一階述語論理の直観主義論理版に部分組み合わせ代数(Partial Combinatory Algebra)の公理系と数学的帰納法の公理を付け加えたものと思ってもよい．ここで"部分"組み合わせ代数とは，例えば1/XのX=0の時に意味が決らないように，常に意味が決るとは限らない項(Partial Term)までも扱うための体系である．EONのrealizerは次の様に定義される．

1) 論理式Aに対するrealizerを e として，"e が Aを実現する"ことを表わす新たな論理式を "e q A" と書き表す．すなわち，形式的体系を"realizing variable" Rと記号 q を導入して "R q A"の形の論理式を含んだものに拡張する．ここでRに入るものが論理式Aのrealizerである．

2) 各論理式A に対して"e q A"は次のように定義される．ここでp0，p1は射影を与える定数であり，対化を与える定数pとともに

pxy & p0(pxy)=x & p1(pxy)=y

なる公理が成り立っているものとする．また項tに対してt↓とは，"項tが定義できる"ことを表わす論理式，N(x)とはxが自然数であることを表わす命題とする．

e q A = A : A がatomicの場合

atomicな論理式はそれ自身で自分を実現しているとしてrealizerを特に付与しない．

e q (A -> B) = $\forall$ a(A & a q A ->ea↓ & ea q B)

この場合のrealizer e は，Aのrealizer aに作用させるとBのrealizerを返すような関数として特徴付けられる．

e q $\exists$ x A = A(p1*e) & p0*e q A(p1*e)

この場合のrealizer e は，実際に変数xに入る値(p1*e)と，その値をxに代入した論理式A(p1*e/x)のrealizerの組として特徴付けられる．

e q $\forall$ x A = x(ex↓& ex q A)

この場合のrealizer e は，任意のxに作用させるとAのrealizerを返すような関数として特徴付けられている．

e q A or B = N(p0*e) & [(p0*e = 0 -> A & p1*e q A)
&
(p0*e =/=0 -> B & p1*e q B)]

この場合のrealizer e は，A，Bのどちらが実際成り立つのかを表わす自然数(p0*e)と，成り立つ方の論理式のrealizerの組として特徴付けられる．

e q A & B = p0*e q A & p1*e q B

この場合のrealizer e は，AのrealizerとBのrealizerの組として特徴付けられる．

3) EONに限らず，構成的証明記述のための形式的体系におけるrealizerの定義の仕方は本質的に同じである．ここで構成的数学や直観主義論理における論理式の意味の捉え方を下に示す．2)のrealizerの定義のところで，"realizer"を"証明"と読み替えると下に示した論理式の意味とほぼ一致する．

A or B の証明 = A の証明 または B の証明
A & B の証明 = A の証明 と B の証明
A -> B の証明 = 任意のAの証明からBの証明を構成する操作手続き
$\exists$ xA(x)の証明 = あるcに対するA(c)の証明
$\forall$ xA(x)の証明 = 任意のcに対してA(c)の証明を構成する操作手続き
not A の証明 = A -> $\perp$ の証明
(= 任意のAの証明から矛盾を導く操作手続き)

4) 論理演算を行うことによってrealizerも変形してゆく．自然演繹法(NK)風の推論規則を考えた場合，例えば

$$\frac{A \qquad B}{A \ \& \ B}\ (\&\text{-導入})$$

に対して規則

$$\frac{c1\ \mathbf{q}\ A \qquad c2\ \mathbf{q}\ B}{p(e1,e2)\ \mathbf{q}\ A\ \&\ B}$$

を対応させる．この規則を仮に"R-規則"と呼ぶことにしよう．ここでp(e1,e2)が&-導入規則の適用によって新たに生成されたrealizerである．実際2)の定義にあてはめてみると，

$$p(e1,e2)\ \mathbf{q}\ A\ \&\ B = p0(p(e1,e2))\ \mathbf{q}\ A\ \ \&\ \ p1(p(e1,e2))\ \mathbf{q}\ B$$
$$= e1\ \mathbf{q}\ A\ \ \&\ \ e2\ \mathbf{q}\ B$$

となってR-規則の前提部分に矛盾しない．(これをSoundness of realizability interpretationという)

このように各推論規則に対してR-規則が対応付けられる．

構成的証明の証明木が与えられたとき，それぞれの節(推論規則を適用している部分)にR-規則を対応させて新しい節(仮に"R-規則節"と呼ぶ)を作ってゆく．そのとき証明木の根に対応するR-規則節の結論部分に現れるているrealizerが，その証明から導出されたプログラムになっている．realizability interpretationとはこのような操作を言う．

・realizerの考え方は(直観主義)論理学と帰納的関数論との接点から生れたものなので，関数型プログラミングとも親和性が良く，従来の研究例を見ても，realizerとして関数型プログラムを採用したものが多い．

(2) 構成的型理論

・直観主義論理や構成的数学の流れの一つに構成的型理論がある．これはとくに1970年代からMartin-Löfやde Bruijinらの手によって発展してきた論理学であり，論理定数や形式的演繹の意味を直観主義の考え方で捉えなおし，構成的数学を十分記述できるような形式的体系の構築を狙ったところに特徴がある．

・構成的型理論では，束縛記号でくくられる変数に注目し，その変数がとりうる値の集合の構造を詳しく論じる．とくにMartin-Löfの理論ではこの方向が強く打ち出されており，変数の集合とその上の論理式や推論規則が同一の視点で扱えることを示している．このような観点からの考察は古典論理の世界では十分には行われていなかったと言える．

例としてMartin-Löfの体系について考えると，集合は

1) canonical element の形式とその等号条件
2) non-canonical elementの評価(evaluation)規則

を与えることによって決定されるものと定義され，さらに集合から新しい集合を構成する規則が与えられている．例えば，$\prod$-規則(Cartesian積)の場合，

(構成規則)

$$\frac{A\ \mathrm{set} \qquad [\,x \in A\,]\ B(x)\ \mathrm{set}}{(\prod x{\in}A)B(x)\ \ \mathrm{set}} \qquad \frac{A = C \qquad [\,x \in A\,]\ B(x) = D(x)}{(\prod x{\in}A)B(x) = (\prod x{\in}C)D(x)}$$

(導入規則)

$$\frac{[\,x{\in}A\,]\ b(x) \in B(x)}{(\lambda x)b(x) \in (\prod x{\in}A)B(x)} \qquad \frac{[\,x{\in}A\,]\ b(x) = d(x) \in B(x)}{(\lambda x)b(x) = (\lambda x)d(x) \in (\prod x{\in}A)B(x)}$$

(除去規則)

$$\frac{c{\in}(\prod x{\in}A)B(x) \qquad a \in A}{Ap(c,a)\ \ B(a)} \qquad \frac{c = d \quad (\prod x{\in}A)B(x) \quad a = b \in A}{Ap(c,a) = Ap(d,b)\ \ B(a)}$$

(等号規則)

$$\frac{a{\in}A \qquad [\,x{\in}A\,]\ b(x){\in}B(x)}{Ap((\lambda x)b(x),a) = b(a) \in B(a)}$$

の4種類の規則が与えられている．構成規則はCartesian積の構成規則で，集合の構成の仕方および2つのCartesian積が等しくなるための条件を示している．導入規則は集合のcanonical elementの構成条件および等号条件，除去規則は集合のelementに対する基本関数の構成規則，等号規則はその基本関数の実行(評価)規則をそれぞれ与えている．

$\prod$-規則のほかに集合の直和を与える$\sum$-規則が有り，やはり構成・導入・除去・等号

の4種類の規則から成り立っている．この2つがMartin-Löfの形式的体系の中心になっている．この2つの規則を"Proposition as type"という解釈で捉え直すことによって，通常の論理学の論理定数に関する公理系・推論規則が導かれる．また，この2つの規則から通常のプログラミング・システムで使われるif-then-elseや射影，対化，超限帰納法などのスキーマも自然に導かれる．

・"Proposition as type"の考え方は，直観主義論理や構成的型理論の特徴的なものである．一般に直観主義論理や構成的数学では"命題が真である"ことを"その命題に対する証明が存在する"こととして定義する．そして，なにをもって命題に対する証明が存在すると言うかを明確にする．一方，構成的型理論では，あるelementがある集合の要素であることを，そのelementが確かにその集合の要素になっているという証明の存在をもって定義し，なにをもってその証明が存在すると言うかを明確にする．すなわち，それが集合の構成・導入・除去・等号規則がそれである．構成的型理論では，このような命題の真/偽の定義，集合の要素のとらえかたの類似性に注目して集合と命題を同一の方法で扱おうとする．これが"Proposition as type"である．

・構成的型理論は，realizability interpretationの考え方と比べて計算機科学との関連付けに不明な点が残されている．すなわち，計算機科学者達の間でその豊かな応用可能性が指摘されてはいるものの，それを実証するシステム設計例や具体的な応用手法の研究例がまだ十分に出揃っている状態ではない．しかしながら構成的型理論が計算機科学とくにプログラミングの問題に重要な示唆を与えていることは確かである．実際，Martin-Löfの理論に基づいて設計された証明記述言語やプログラミング言語は1970年代から研究・試作が行われているし，抽象データ型やそれを拡張したプログラムの型付けへの応用，あるいは仕様とプログラムの関係を明確にしソフトウェア開発支援システムを構築する際の設計思想に反映しようという研究が最近活発に行われている．この意味で"構成的証明 + realizability interpretation"の方向での研究開発をおこなう際にも構成的型理論の計算機科学への応用研究の動向は無視できない．

・とくにプログラムの型付けについては，プログラムの静的検証，最適化，プログラムの再利用を狙ったプログラム・モジュールの一般化および検索を行う上で強力な道具になることが認識されはじめており，構成的型理論はその観点からも無視できない．

(3) Typed Logical Calculus (QJ)

・Typed Logical Calculusは東北大(前東大)の佐藤雅彦教授が提案している関数型プログラミングを包含する論理体系の理論で，その形式的体系はQJと呼ばれる．この理論は論理型プログラミングの立場から見ても十分魅力的な特徴を持っている．

・すなわち，形式的仕様，仕様に対する証明，それらから導出されるプログラムが同一の形式的体系の中で記述できるので，システムの統一性が良く，さらにrealizability interpretationの手法を使ってプログラムの導出を行う方法も示されている．またプログラムの型付けが行われており，型による正当性検証の推論規則の体系が与えられている．とくに帰納法に関しては通常よく使われる超限帰納法の他に，再帰的に定義された型の構造に関する帰納法の規則が与えられているところに特徴がある．これらの意味で，知的なプログラミング・システムの在り方の一つであると言える．

・Typed Logical Calculusの概念図を図-3 に示す．

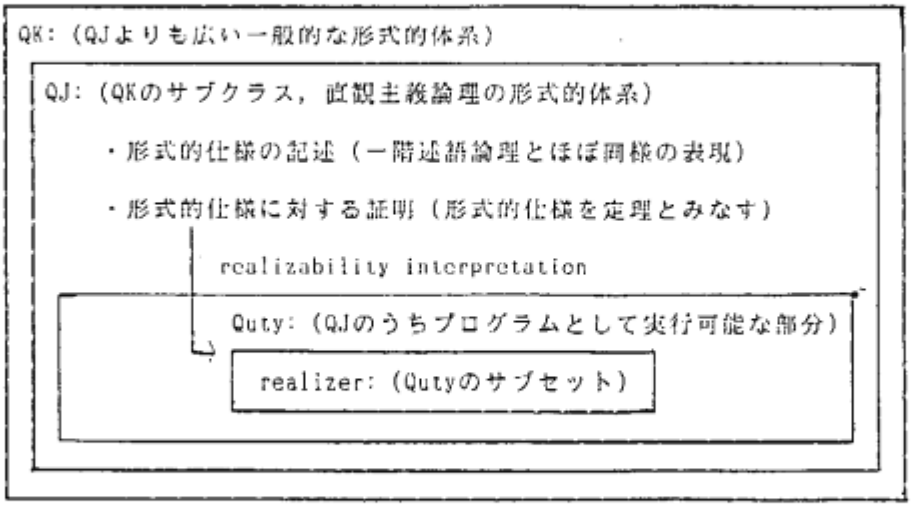

図-3:: QJ/Quty(Typed Logical Calculus)の概念図

Qutyそのものは"論理式もある程度使える関数型言語"とも言うべきもので，Prolog的な書き方もできるが，クローズをOR並列に並べるような書き方はできない．またrealizerの部分はかなり関数型言語に近いものになっている．

(4) 現状と展望

1) 構成的証明の経験

・CAP-LAで扱われている線形代数の場合，その証明の多くは構成的であると言える．しかし，例えばQJなどで言う形式的仕様に対する構成的証明がどのようなものであるかは，高級言語によるプログラミングで言う再帰呼出しに対応する数学的帰納法による証明が多く，内容的には通常のプログラミングの概念をそれほど逸脱するような側面は少ないであろうということ以外に明確な実感が十分に得られているわけではない．

・すなわち，"構成的証明 ≡ プログラム"のパラダイムでのプログラミングの経験は現状では十分とは言えない．その意味でも，構成的証明の記述実験に止まらず，実験的なプログラミング環境を早期に構築することにより，プログラミング経験を蓄積してゆくことが必要である．

2) 非決定的処理

・通常，構成的証明といえば手続きが明解にわかったものについての記述であるため，アルゴリズムとしては決定的なものに限られてしまう．しかしながら，構成的証明の書き方を拡張することにより，非決定的な動きをするプログラムを導出することの可能性を完全に否定してしまうような根拠は今のところ見当たらない．従って，構成的証明によるプログラミング・パラダイムの発展/拡張によって，非決定的処理の記述が可能になることが期待される．

・あるいは，証明が構成的でなければならないという制限を取り除くか弛めることによって，非決定的プログラムの導出にむすびつけることも考えられる．これは技術的には未知であるが，証明やプログラムの理論的枠組みを大きく拡げることを可能とするものなので，検討に値する．

・また非決定性の記述を一つの特徴とする論理型言語でのプログラミングを考えてみても，決定的処理のプログラミングの比重は大きく，その部分に絞って開発支援を行っても十分効果は大きい．

3) 論理型言語の体系での展開

・関数型プログラミングの世界で始まった等価変換技術は，現在論理型プログラミングの体系の中に取り込まれ，論理型プログラミングの等価変換として発展している段階にある．

・それに対して構成的証明に基づくプログラミング技術を論理型プログラミングの体系の中で展開する試みは今まで十分には行われてこなかった．等価変換技術に対するもうひとつの技術として，現在おもに関数型プログラミングの世界で展開されている構成的証明によるプログラミング技術，特にrealizability interpretationの技術を論理型言語の体系の上で展開することが期待される．

・ただし，現在の枠組みのままでは論理型言語の体系で展開してもそれほど利点があるとは言えないむしろ導出されるプログラムが質的に論理型言語に向いているようなものであることが必要である．

4) 大規模プログラミング

・大規模なプログラムを開発するためには
  a) モジュラー・プログラミング
  b) 抽象データ型の記述
  c) 既存モジュールの再利用
  などの手法を導入する必要がある．

・構成的証明に基づくプログラミング技術の領域では，このようなことが十分に配慮されているとは言い難い．しかしながらa) b)に関しては，計算機科学での蓄積はすでに十分あると判断できるし，c)に関しても構成的型理論をもとにプログラムの型付けを行うことによりかなりのことが可能になるという研究報告もある．

・これらの記述を構成的証明に基づくプログラミング技術にどのように取り込んで行くかは大きな課題の一つである．

・また構成的証明において，証明の細部を自動的に補う機能を実現することも大規模プログラミングを可能にするための必須条件である．

(5) 日本語仕様記述言語(JSL)との関連

1) プログラム部品作成ツール

・(4)4)で述べたように構成的証明に基づくプログラミング技術を大規模プログラミングに十分使えるものに発展させることは重要である．しかしながら，厳密に検証され

たプログラム部品を作るツールとしては，この技術が現在のレベルでも十分使える．
・この場合，作成済みのプログラム部品を格納し必要に応じて再利用できる機能を実現するために，モジュール知識ベースが不可欠になってくる．
・構成的証明に基づくプログラミング技術をこのように位置付けた場合，JSLはもっと大きなレベルのプログラミングを支援することが望まれる．すなわちプログラマが問題を前にしておおまかなモジュール分割やモジュールの機能仕様を考えたり，それを少しずつ詳細化してゆく作業を支援するものであることが望ましい．

2) JSLとのインタフェース
・JSLと構成的証明に基づくプログラミング・システムとの間のインタフェースは，一階述語論理あるいはそれに基づく形式的仕様記述言語である．これは等価変換技術の領域で考えられているものと同様である．
・しかしながら，構成的証明に基づくプログラミング・システムをプログラム部品作成ツールと見做した場合，どのレベルまでをプログラム部品と呼ぶか，言い換えれば人間がJSLを使ってどのレベルまでモジュールの詳細化を行わなければならないかは研究課題である．
・この問題については，技術的にJSLや構成的証明に基づくプログラミング・システムがどこまでの機能を持ちうるか，といったことのみならず，人間にとってどこまでの作業を自然言語で行うのが快適かといったことも考慮する必要がある．

(6) システム全体像
システム全体像を図-4 に示す．各モジュールの詳細は(4)(2)で触れる．

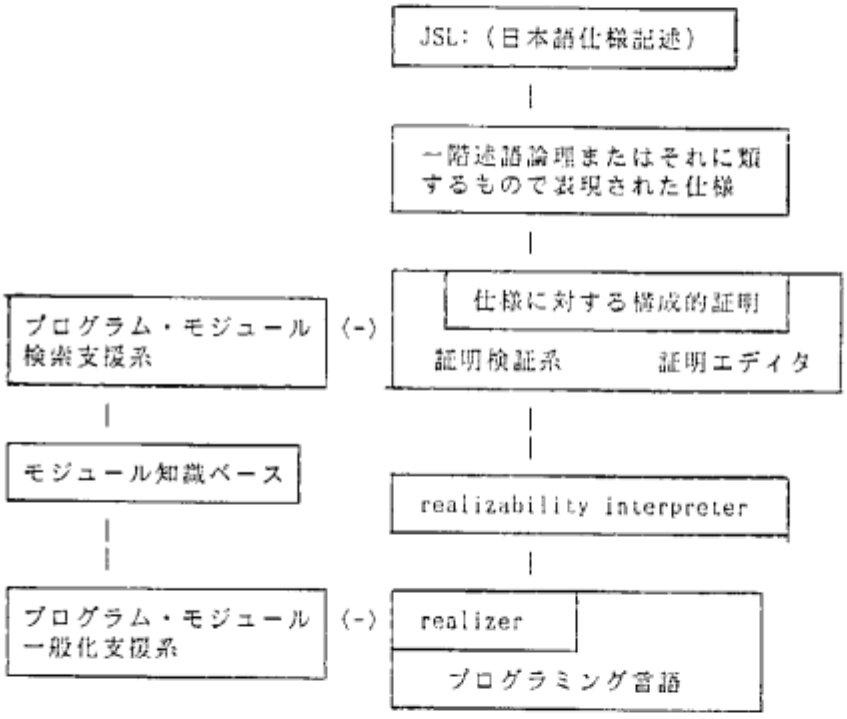

図-4::構成的証明の基づくプログラミング・システムの全体像

(3) 等価変換技術との比較

(1) プログラム導出の考え方
・等価変換技術においても一階述語論理あるいはそれに類似した形式的仕様からプログラムを導出する手法が研究されている．等価変換での手法は形式的仕様に変換規則を適用していって最終的にプログラムを導出しようという立場に立っている．
・等価変換規則を推論規則とみなすことも可能であるが，等価変換技術はあるモデル(例えばPrologの最小不動点モデル)を考えて，そのモデルが与えるプログラムの意味

を変えないように変換規則を定めることが本質的であることに注意しなければならない．すなわち，通常の証明論で言うところの推論規則とは内容的にかなり異質である．
・また変換規則は現在のものだけで十分であるかどうかは，理論的にも実際的にも明確には分かっていないし，変換規則の意味が人間にとって明解なものでないと使いやすい会話的なシステムが構築しにくいという条件が付く．
・さらに等価変換の考え方からすると，変換前の仕様もモデルに関する十分な情報を明示しているものでなければならず，完全な仕様を書くための支援方法を別途考える必要が出てくる．
・それに対して構成的証明に基づくプログラミング技術は形式的仕様を変換するという考え方ではなく，形式的仕様を定理とみなしてそれに対する証明を記述するという考え方をとる．

(2) 項書換えシステム，証明検証技術から見た特徴
・等価変換技術によるシステムは一種の簡約系(リダクション・システム)と考えることができる．推論がかなり高いレベルで行われる点を除けば，項書換えシステムと類似していると言える．等価変換は一般に項書換えではないので，等価変換への項書換えシステムの直接的応用は限られてしまう．しかしながら，システム全体のユーザ・インタフェースの設計に項書換えシステム開発の経験は十分生かせる．
・構成的証明に基づくプログラミング技術では，構成的証明の検証系や証明記述支援系の部分にCAPの技術が直接的に生かせる．この場合，項書換えシステムはCAPシステムと同様に等式部分の検証に使われることになる．

(3) 最適化の問題
・等価変換技術では形式的仕様からのプログラム導出のみならず，導出されたプログラムの最適化も同様の方法で自然に扱えるという特徴がある．
・それに対して構成的証明に基づくプログラミング技術では，最適化技術についての明確な方法論が確立されているとは言い難い．

(4) 等価変換技術と構成的証明によるプログラミングの関連
・等価変換技術と構成的証明に基づくプログラミング技術は(1)(2)(3)でみたように互補的な特徴を持っている．
・とくに，構成的証明に基づくプログラミング技術は扱う対象として決定的処理が中心となる．それに対して等価変換はモデルなどの意味論が明確に与えられている言語ならば原理的には成立しうる技術であり，Prologなどの持つ非決定性なども自然に扱える長所を持っている．したがってこれらを融合することが望ましい．
・しかしながら，両者が技術的に融合しうるのかあるいはシステム的に融合すべきなのかは現段階では不明な点が幾つか残されている．

(4) 研究開発方針と研究項目

(1) 研究開発方針
1) QJ/Qutyの捉え方
・構成的証明に基づくプログラミング技術の理論として現在最も体系的にまとまったものとしてQJ/Qutyを出発点とする．
・QJ/Qutyは佐藤教授のライフ・ワークとして年々発展し続ける理論なので，ICOTとしてはQJ/Qutyの適当なバージョンを切り出して固定し，それを当面の題材として広い立場で研究する．
・将来的にはICOT版QJ/Qutyとして，ICOTオリジナルなシステムにまでもって行くことを目標とする．その場合にも，佐藤教授側での理論的発展をできうる限り取り入れる方向で進める．
2) 研究開発の進め方についての方針
・当面はCAPプロジェクトの範囲の活動として，定理証明技術のプログラミング技術への応用という観点で取り組む．
・特に構成的証明とそのrealizability interpretationの周辺に絞って，
　　1. 構成的証明としてのプログラミングの特性の把握
　　2. realizability interpretation手法の有効性確認と実装技術の検討
　に最重点を置いて研究開発を進める．
・その他の部分については，上の1,2.の作業を通して基礎的検討を行う．

(2) 研究項目

以下の項目において，〔当面の目標〕とは現段階において実現可能性がある程度見えているもの，〔将来の目標〕とは何らかの研究成果をあげることによって可能性が開かれるであろうと期待されるものを指す．

1) 調査

・構成的数学一般およびそのプログラミングへの応用についての技術動向調査

〔備考〕

・具体的調査項目の候補としては
Martin-Lofの構成的型理論とそのプログラミング上の応用，萩谷氏のプログラム・モジュール一般化の研究，林晋氏のLISPプログラム生成系の研究，Fefermanの型理論，QJの最新の状況，Waldingerのプログラム検証/生成システムなど

2) 証明検証系

〔当面の目標〕

・証明記述言語によって記述された構成的証明の正当性をチェックするもの．
・証明は省略のない完全なものを対象とする．
・補題(lemma)の参照ができる機能をもたせる．

〔将来の目標〕

・証明の行間を適当に補ってくれる機能を持たせる．

〔備考〕

・CAP-LAの研究開発の成果を直接的に反映して行く．
・証明検証系の実装はESPで行う．
・CAP-LAにrealizability interpreterをつける方向での検討も行う．
・記述実験により，証明検証系の機能詳細を洗い出す．

3) realizability interpreterの検討

〔当面の目標〕

・検証済みの構成的証明の証明図を解析してrealizerを生成する
・決定的アルゴリズムを記述したプログラムの生成

〔将来の目標〕

・realizerとしてPrologに近いプログラムが生成できるようにする．具体的には，非決定的アルゴリズムを表現したプログラムが生成できるようにする．

4) 対象言語および言語処理系の検討

〔当面の目標〕

・Qutyあるいはそれに基づくもの
・処理系はESP上のインタプリタ

〔将来の目標〕

・Prologをサブセットとして完全に含み，さらに項表現や抽象データ型の表現が強化された言語．

5) 仕様記述言語および証明記述言語

〔当面の目標〕

・QJまたはそれに基づくもの
・項の表現能力が多少強化された一階述語論理

〔将来の目標〕

・抽象データ型の表現能力の強化されたもの

〔備考〕

・CAP-LAで開発された証明記述言語PDLをベースにしたものと想定している．

6) 仕様/証明記述用エディタ

・CAP-LAシステムと同じく，システム・コントローラと直結して，エディター内部から検証系/realizability interpreter，モジュール検索・一般化支援系などが呼び出せるもの．
・強力な項書換え機能をもち，簡単の論理演算や機械的な変換はエディタからのコマンド操作によりその場で実行できるようなもの．

〔備考〕

・CAP-LAで開発中のものが使えることが望ましいが，いずれにせよ設計には手間がかかる．

7) 最適化系
〔当面の目標〕
・realizerとして導出されたプログラムを等価変換によって最適化する．但し，変換後のプログラムは，realizerとして導出可能なものの範囲に限定することにより，構成的証明とプログラムの対応関係を保つ．
〔将来の目標〕
・等価変換によって最適化されたプログラムがrealizer集合をはみ出すようなものも扱う．この場合，明解な意味付けをされた等価変換で最適化する．
・証明図の正規化による証明レベルの最適化．
（lemmaを参照している部分をlemmaの証明で置き換えた上で証明全体を正規化する）
〔備考〕
・最適化は実用化システムとして磨き上げる段階での課題とする．

8) モジュール知識ベースとモジュール再利用支援系
〔当面の目標〕
・型付けされたプログラム・モジュール，仕様，変換・証明情報を格納する．
・モジュール名，型による検索
・世代管理，一貫性管理，クロス・リファレンス，変換過程のログの管理
〔将来の目標〕
・型付けに基づいて一般化（パラメータ化）されたプログラム・モジュールの格納
・型の階層構造による類似検索

〔5〕文献案内

〈〈構成的数学〉〉
・〔Beeson 85〕は内容的に一番まとまっている．内容は，構成的数学の例，歴史的経緯，直観主義的集合論，直観主義的型理論(Martin-Löf理論)，Fefermanの理論などで，構成的数学を記述するための形式的体系の幾つかを紹介し基本的な考え方を詳説している．予備知識としてモデル理論の初歩，古典論理学は必須である．
モデル理論の文献としては，〔Chang & Keisler 73〕など，また古典論理学については〔Kleene 52〕〔Shoenfield 67〕〔Prawitz 65〕あたりが良い．
・その他参考になるものとしては，〔Bishop 67〕〔McCarty 84〕〔Troelstra 73〕〔Troelstra 77〕〔Martin-Löf 68〕がある．〔Bishop 67〕は構成的数学の古典とされている．〔McCarty 84〕はrealizerbility interpretation について詳しい．〔Martin-Löf 68〕は帰納的関数，構成的解析学，順序数とボレル集合，測度論について直観主義論理の立場から論じている．〔Troelstra 73〕〔Troelstra 77〕は構成的数学記述のための形式的体系として70年代にとくに注目されていたN-HAを中心に扱っている．
また〔Hayashi 83〕は構成的証明からLISPのプログラムを導出する研究である．
・〔Coquand & Huet 85〕はこの方面の詳細な文献リストである．

〈〈Typed Logical Calculus〉〉
・〔Sato 85-2〕が現在のところ公表されている唯一の論文である．しかしながら〔Sato 83〕〔Sato 85-1〕も合せ読むと佐藤教授の一貫した思想が読み取れよう．
・〔Beeson 85〕の前半部分の知識とScott理論の知識があると，〔Sato 85-2〕の理解の助けになろう．Scott理論については〔中島 82〕が分かり易い入門書である．
・〔Sato 85-3〕は東京大学情報科学科での講義録である．Typed Logical Calculus の内容は〔Sato 85-2〕と若干異なっている．"プログラム"の定義が若干変更され，また〔Sato 85-2〕には示されていなかったQJの操作的意味論が与えられている．
・Qutyの前身であるQuteについては〔Sakurai & Fujita 85〕〔Sato & Sakurai 84〕をみよ．

〈〈等価変換技術〉〉
・等価変換技術の古典としては〔Burstall & Darlington 77〕．
・論理型言語の等価変換技術の文献としては〔Tamaki & Sato 83〕〔Kanamori & Maeji 86〕〔Kanamori & Horiuchi 86〕〔Kanamori & Fujita 86-1〕などをみよ．三菱中研の金森グループの研究は多岐にわたっており，等価変換以外にも論理型プログラムの検証方法についても幾つかの文献がある．それについては，〔Kanamori & Seki 86〕〔Kanamori & Fujita 86-2〕〔Kanamori & Horiuchi 85〕〔Kanamori 86〕〔Kanamori, et al 86〕などをみよ．
・論理型言語の等価変換技術の基礎になっているPrologの最小不動点モデルについては〔Lloyd 84〕がよくまとまっている．

〈〈Martin-Löf理論〉〉
・Martin-Löf理論については[Martin-Löf 82][Martin-Löf 84]．特に[Martin-Löf 84]が内容的に一番まとまっている．また[Beeson 85]にもMartin-Löf理論についての詳しい解説がある．また[高山 86]もある．
・[Martin-Löf 85]はMartin-Löf自身の直観主義論理に対する哲学を論じている．難解とされるMartin-Löf理論の哲学的背景が紹介されている．
・Martin-Löf理論に関連した研究としては[Smith 82][Smith 84][Aczel 80]などがある．
・構成的型理論の計算機科学への応用としては[萩谷 85][萩谷 86]がある．

〈〈定理証明/証明記述言語〉〉
・Edinburgh大学で開発されたLCF/MLについては[Gordon et al 84]
LCFは証明システムでMLはシステム記述言語である．また証明記述言語はPLAMBDAと呼ばれる．
・Martin-Löf理論に基づいた証明記述言語として有名なAUTOMATHについては[de Bruijn 70]
・その他構成的数学をもとにした証明記述言語の文献としては[Constable et al 85][Constable 83]
・ICOTで開発された定理証明システム(CAP-LA)については[廣瀬　他 86][横田　他 86][筧　他 86][永田　他 86][坂井　他 86][大須賀　他 86]をみよ．
・[Nagata et al 80][Nakanishi et al 79][Nishimura et al 75]は定理自動証明をもとにしたプログラミング支援技術の研究として参考になろう．
・定理証明技術の入門書としては[Loveland 78][Chang & Lee 73]がある．

〈〈その他〉〉
・[Burstall & Lampson 84]は大規模プログラミング言語Pebbleについての文献である．
・[Futatsugi et al 84]は項書換えに基づく形式的仕様記述言語OBJ2についての文献である．
・[Partsch & Steinbruggen 83]は種々のプログラム変換システムを紹介しており，この方面の動向を知るのに適している．
・[Knuth 85]は数学とプログラミングの思考の違いや類似について書かれた格調高い論文であるが，比較的気軽に読める．
・[Takasu & Sato 83]は証明とプログラムの関係付けを理論的に扱ったものとして参考になろう．
・項書換えシステム(TRS)に関する文献は数多いが，ここでは[大須賀 他 86]を挙げるにとどめる．

[Aczel 80]Aczel,P.,"Frege Structure and the Notions of Proposition,Truth and Set", The Kleene Symposium (ed:Barwise, Keisler and Kunen), pp31-59,North-Holland,1980.

[Barendregt 83]Barendregt,H.,"Semantics for Calssical AUTOMATH and Related Systems", Information and Control 59, pp127-147, 1983.

[Beeson 85] Beeson,M.,"Foundations of Constructive Mathematics", Springer 1985.

[Bishop 67]Bishop,E.,"Foundations of Constructive Analysis", McGraw-Hill, 1967.

[Burstall & Darlington 77]Burstall,R.M.and Darlington,J.,"A Transformation System for Developing Recursive Programs", J.of ACM, Vol.24, pp44-67, 1977.

[Burstall & Lampson 84]Burstall,R.M.and Lampson,B.,"A kernel language for abstract data types and modules", Semantics of Data Types, Lecture Notes in Comp.Science 173, Springer, 1984.

[Chang & Keisler 73]Chang,C.C.,and Keisler,H.J.,"Model Theory", North-Holland, 1973.

[Chang & Lee 73]Chang,C.and Lee,R,C.,"Symbolic Logic and Mechanical Theorem Proving", Academic Press, 1973.

[Constable,et al 85]Constable,R,L.,Knoblock,T,B.,and Bates,J,L."Writing Programs that Construct Proofs", J.of Automated Reasoning 1, pp285-326, 1985.

[Constable 83]Constable,R.,"Partial Functions in Constructive Formal Theories", Lecture Notes in Comp.Science 145, pp1-19, Springer, 1983.

[Coquand & Huet 85]Coquand,T.,and Huet,G.,"A Selected Bibliography on Constructive Mathematics, Intuitionistic Type Theory and Higher Order Deduction", J.Symbolic Computation 1, pp323-328, 1985.

[de Bruijn 70]de Bruijn,N,G.,"The mathematical language AUTOMATH, its usage, and some of its extentions", Lecture Notes in Math. 125, Springer, 1970.

[Futatsugi,et al 84]Futatsugi,K.,Goguen,J,A.,Jouannaud,J,P. and Meseguer,J., "Principles of OBJ2", ICOT Technical Report 97, ICOT, 1984.

[Gordon,M.et al 78]Gordon,M.,Milner,R.,and Wadsworth,C.,"Edinburgh LCF", Lecture notes in Comp. Science 78, Springer, 1978.

[萩谷 85]萩谷,"構成的型理論における一般化について",日本ソフトウェア科学会第2回大会論文集,pp189-192, 1985.

[萩谷 86]萩谷,"構成的型理論における一般化について",理化学研究所シンポジウム - 関数型プログラミング - 論文集, pp61-67, 1986.

[Hayashi 83]Hayashi,S.,"Extracting LISP programs from constructive proofs: A formal theory of constructive mathematics based on LISP", Publ.RIMS, Kyoto Univ., 19, pp169-191, 1983.

[廣瀬 他 86]廣瀬, 横井, 横田, 坂井, 玉井,"CAP プロジェクト(1) ねらいと構想", 情報処理学会第32回全国大会予稿集,1986.

[筧 他 86]筧, 坂井, 西山,"CAP プロジェクト(3) 証明記述言語",情報処理学会第32回全国大会予稿集, 1986.

[Kanamori & Horiuchi 86]Kanamori,T.and Horiuchi,K., "Construction of Logic Programs Based on Generalized Unfold/Fold Rules", ICOT Technical Report, to appear, 1986.

[Kanamori & Fujita 86-1]Kanamori,T.and Fujita,H.,"A Regimement of Unfold/Fold Transformation of Logic Programs", ICOT Technical Report, to appear , 1986.

[Kanamori & Maeji 86]Kanamori,T.and Maeji,M.,"Derivation of Logic Programs from Implicit Definition", ICOT Technical Report, to appear, 1986.

[Kanamori 86]Kanamori,T.,"Soundness and Completeness of Extended Execution for Proving Properties of Prolog Programs", ICOT Technical Report, to appear, 1986.

[Kanamori,et al 86]Kanamori,T.,Fujita,H.,Seki,H.,Horiuchi,K.and Maeji,M., "Argos/V : A System for Verification of Prolog Programs", ICOT Technical Report, to appear, 1986.

[Kanamori & Seki 86]Kanamori,T.and Seki,H.,"Verification of Prolog Programs Using an Extention of Execution", Proc. of 3rd International Conference on Logic Programming, 1986.

[Kanamori & Horiuchi 85]Kanamori,T.and Horiuchi,K.,"Type Inference in Prolog and Its Applications", Proc. of 9th International Joint Conference on Artificial Intelligence, 1985.

[Kanamori & Fujita 86-2]Kanamori,T.and Fujita,H.,"Formulation of Induction Formulas in Verification of Prolog Programs", Proc. of 8th Conference on Automated Deduction, 1986.

[Kleene 52]Kleene,S.C.,"Introduction to Metamathematics",North-Holland, 1952.

[Knuth 85]Knuth,D.E.,"Algorithmic Thinking and Mathematical Thinking", American Mathematical Monthly, pp170-181, March 1985.
(邦訳:一松 信訳,"算法的思考と数学的思考",bit Vol.18,No 4, 1986)

[Loveland 78]Loveland,D,W.,"Automated Theorem Proving: a logical basis", North-Holland, 1978.

[Martin-Löf 68]Martin-Löf,P.,"Note on Constructive Mathematics",Almoqvist& Wiksell, Stockholm,1968.

[Martin-Löf 82] Martin-Löf,P.,"Constuctive mathematics and computer programming", in 'Logic, Methodology, and Philosophy of Science VI(ed: Chohen,L.J.et al)" pp153-179, North-Holland,Amstermam, 1982.

[Martin-Löf 84]Martin-Löf,P.,"Intuitionistic Type Theory", Biblioplis, Napoli, 1984.

[Martin-Löf 85]Martin-Löf,P.,"On the meaning of the logical constants and the justifications of the logical laws", Proceedings of the 3rd. Japanese-Swedish Workshop,pp34-112, 1985.

[McCarty 84]McCarty,D,C,."Realizability and Recursive Mathematics", Carnegie-Mellon University, CMU-CS-84-131, 1984.

[中島 82]中島玲二,"数理情報学入門", 朝倉数理科学ライブラリ3, 朝倉書店, 1982.

[Nagata et al 80]Nagata,M.,Akiyama,T.and Fujikake,Y.,"An interactive supporting system for functional recursive programming", Proc. of IFIP Congress 80, pp263-268, 1980.

[Nakanishi et al 79]Nakanishi,M.,Nagata,M.and Ueda,K.,"An automatic theorem prover generating a proof in natural lenguage", Proc.of 6th IJCAI, pp636-638, 1979.

[永田 他 86]永田, 高山, 西山,"CAP プロジェクト(4) 証明の検証",情報処理学会第32回全国大会予稿集, 1986.

[Nishimura et al 75]Nishimura,T.,Nakanishi,M.,Nagata,M.and Iwamaru,Y., "Gentzen-type formal system representing properties of functions and its implementation", Proc.of 4th IJCAI, pp57-64, 1975.

[大須賀 他 86]大須賀, 藤瀬,"CAP プロジェクト(6) 等式の検証",情報処理学会第32回全国大会予稿集, 1986.

[Partsch & Steinbruggen 83]Partsch,H.and Steinbruggen,K.,"Program

Transformation Systesms", ACM Computing Surveys,Vol.15,No.3, pp199-236, 1983.

[Prawitz 65] Prawitz,D.,"Natural Deduction", Almqvist and Wiksell, 1965.

[坂井　他 86]坂井，小久保,"CAP プロジェクト(5) 証明支援ユーティリティ", 情報処理学会第32回全国大会予稿集，1986.

[Sakurai & Fujita 85]櫻井，藤田,"QUTE処理系操作説明書", ICOT Technical Report 106, ICOT, 1985.

[Sato 83]Sato,M.,"Theory of Symbolic Expressions,I", Theoretical Computer Science 22, pp19-55,1983.

[Sato 85-1]Sato,M.,"Theory of Symbolic Expressions,II", to appear in Publ. RIMS,Kyoto Univ.

[Sato 85-2]Sato,M.,"Typed Logical Calculus (Provisional version)",Proceedings of the 10th.IBM Symposium of Mathematical Information Science ,1985.

[Sato 85-3]Sato,M.,Lecture given at the Tokyo University on Metamathematics, Tokyo University, 1985. (河内谷清久二氏による講義ノート)

[Sato & Sakurai 84]Sato,M. and Sakurai,T.,"Qute:A Functional Language based on Unification", Proc. of the International Conference on Fifth Generation Computer Systems", pp157-165, 1984.

[Shoenfield 67]Shoenfield,J.R.,"Mathematical Logic", Addison-Wesley, 1967.

[Smith 82]Smith,J.,"The identification of propositions and types in Martin-Lof's type theory: a programming example", Lecture Notes in Comp. Science 158, Springer, 1982.

[Smith 84]Smith,J.,"An interpretation of Martin-Lof's type theory in a type-free theory of propositions", J.of Symbolic Logic Vol.49,No.3, 1984.

[Tamaki & Sato 83]Tamaki,H.and Sato,T.,"A Transformation System for Logic Programs which Preservers Equivalence", ICOT Technical Report 18, ICOT , 1983.

[Takasu & Sato 83]Takasu,S.and Sato,M.,"Studies of Program Synthesis:Proofs (-) Programs", JARECT Vol.7, Computer Science & Technologies(ed: T.Kitagawa),OHMUSHA and North-Holland, 1983.

[高山 86]高山,"Martin-Löf理論に関する調査報告",ICOT知的プログラミング・グループ内部資料,1986.

[Troelstra 73]Troelstra,A.(ed),"Metamathematical Investigation of Intuitionistic Arithmetic and Analysis",Lecture Notes in Math.344, Springer, 1973.

[Troelstra 77]Troelstra,A.,"Aspects of constructive mathematics" , Handbook of Mathematical Logic (ed: Barwise,J.), North-Holland, 1977.

[横田　他 86]横田，藤田,"CAP プロジェクト(2) 証明記述支援環境",情報処理学会第32回全国大会予稿集,1986.