ICOT Technical Memorandum: TM-0301

TM-0301

基礎ソフトウェア・システム
(昭和61年度　成果の概要)

May, 1987

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

Ｓ１．　引き続きまして，基礎ソフトウェア・システムについてお話します．

Ｓ２．　ここでは問題解決推論モジュールについて，そのつぎに知的プログラミングシステムについてお話します．

Ｓ３．　先程，内田から核言語の第１版の基礎がＧＨＣであることについてご説明致しましたが，ここで，これからすすめる話の角度からもう一度眺めて少しだけおさらいを致します．
ＧＨＣはプロセスとデータフローのパラダイムを両立していまして，データフロー的な並列制御を可能としております．また，コミットオペレータを用いることにより論理プログラムにガードの考え方を導入しています．
ですから，ガードのコミッティッドチョイスの際は複数の正しい道から１つが選ばれますと他の道が消されてしまうdon't care nonderminsticな処理を行います．このことによりＧＨＣには全解探索機能が欠落していると見えるわけです，が，あとで申しあげますように，全解検索機能はコンパイル技法によって実現することができます．
また，データフローとプロセスの関係はおおまかに言えばメッセージとオブジェクトの関係でありますので，ＧＨＣは，オブジェクト指向プログラミングであると見ることができるのも大きな特徴でございます．

このＧＨＣはICOTの上田によって提案されたものですが，類似の先駆者である並列プログラミング言語，たとえばClark のParlogとか，Shapiro のConcurrent Prolog などの中で最も単純な言語であるといえます．

Ｓ４．　このＧＨＣを基にして，核言語の第１版が階層的に定められました．これらの各階層間で上位から下位へのコンパイル技法，また，一番下の層を実現する処理系についての研究が進み，並列推論システムの実現に向かって開発を行っていることは前の話にありました．
ここでは，知識プログラミング言語と問題解決推論モジュールのお話をいたします．知識プログラミング言語はメタプログラミングとか制約プログラミ

ングとか高度な知識表現技法を一般的に扱えることを目的にします．この知識プログラミング言語を先ほど話のあったＫＬ－Ｕのさらに上位に位置づけることにしています．
ですから，このスライドで示したように，知識プログラミグ言語とそれ以下の言語へのプログラム変換がさらに必要となってきます．このために，問題解決推論モジュールを存在させるわけでございます．

Ｓ５．　特に私共は，現在，問題解決推論モジュールの研究に当たりましては，プログラム変換，メタプログラミング，部分計算を要素技術としてとらえているところであります．ここに，部分計算とは部分評価とも言いますが，その時点で計算可能な部分をその条件の範囲で計算してしまうことであります．この概念と，単一化を基本演算とするロジックプログラムとの親和性が良いことから，この後でお話いたしますように，部分計算を用いた種々の研究結果が得られています．
その他，問題解決推論モジュールの実現に向け高度な知識表現のコンパイル技法とか並列制御，実時間制御のプラグマティックな手法による研究も進めているところでございます．

Ｓ６．　つぎに，プログラム変換の最近の成果といたしましては，ここに挙げましたように，

・プロセスを融合する技法
・コルーチン制御を逐次制御へ変換する技法
・全解収集プログラムを決定的プログラムへ変換する技法

などがあります．
以下に少し詳しく御紹介いたします．

Ｓ７．　ＧＨＣにおけるプロセスの融合は，プロセスのモジュラリティと処理効率のトレードオフにかかわる問題であります．述語定義の展開および畳み込みによるプログラム変換技法によって，処理系の特性に合ったプロセス粒度を最適にすることができることを示しました．

S８．　つぎに二つ目のプログラム変換のお話です．generate and test 型で宣言的に書かれたPrologプログラムはプログラマにとって理解しやすいのですが，そのプログラムをそのまま逐次処理系で実行させると極めて効率が悪くなります．従って，これまではコルーチン処理系を用いて，その上でgenerate and test 型プログラムを実行させていました．しかし，このようなコルーチン処理系は実行時のオーバーヘッドが大きくなる欠点があり未だ十分効率的ではありませんでした．

そこで，ICOTの世木は展開及び畳み込みに基づくプログラム変換手法を与えて，generate and test 型からそれと等価な逐次型プログラムを導出できることを示しました．

このことにより，変換されたプログラムを逐次処理系の上で実行させることができ，このときの効率はコルーチン処理系以上の実行効率を達成することができました．

S９．　つぎは三つ目ですが，上田は入出力モード解析により全解収集論理プログラムはＧＨＣ，または決定性のprologプログラムに自動コンパイルできることを示しましたが，これは，ＧＨＣの上でも探索問題の並列実行ができることを示したことになります．すなわち，見掛け上欠落していた全解収集機能はコンパイル技法により回復することができることを示したわけであります．

全解収集プログラムのコンパイル技法としては，いまお話したコンティニュエィション方式の他，ストリーム方式が（筑波大の玉木により）提案されています．コンティニュエィション方式とはand 並列プロセスの一方の残りの計算を，もう一方のプロセスに付加して全解を求めて行く方式であります．ストリーム方式とはand 並列プロセスの一方の全解を求めます．それをもう一方のand 並列プロセスのストリーム入力とする方式であります．

最近さらに，ICOTの奥村らによって，レイヤードストリーム方式が考られました．レイヤードストリーム方式とは計算過程で発生する途中結果をand 並列プロセス間で相互に通信することができます．このストリームは計算の並

列度を上げるために，レイヤード構造を持っています．また，レイヤード構造を持ったフィルターでストリーム中の無駄な途中結果を削除できることが大きな特徴であります．

これらの方式の詳細と，グッドパス問題，８クイーン，キューブ，整合ラベリングとか，並列構文解析などの問題に実際に適用したときの並列度，処理効率などの評価については明日発表することにしています．

S 10. つぎに，問題解決推論モジュールにおける中核技術の一つとなる部分計算によるプログラム変換をメタとオブジェクトプログラムの概念に応用したお話に移ります．

メタ・プログラミングの特徴付けは，

（１） プログラムそのものをデータとして扱えること，

（２） 逆にデータをプログラムとして扱えて，評価できること

（３） プログラムの実行結果（true,fail)をプログラム又はデータとして扱えることであります．

要するにおおまかに言えばプログラムもデータも同じに扱えることであります．このことは，Bowen とKowalskiがdemo述語によって最初に提唱しました．メタの概念を導入すれば，メタ・レベルとオブジェクト・レベルを分離することができます．これにより，プログラムの意味が明瞭となり，かつ読み易く，書き易い利点が生まれます．しかも，メタ，オブジェクトのプログラムが同一論理体系の枠内で扱えます．その反面，メタ・レベルとオブジェクト・レベルを分離したまま実行すると効率が著しく悪くなるという欠点があります．そこで，インタプリタをメタとして，プログラムまたはルールをオブジェクトとしてとらえて，メタインタプリタを部分計算することにより，プログラム対応の特殊化されたインタプリタを生成することができます．こうすることにより，実行効率は充分実用的なものに改善されます．

S 11. つぎにもう一つの例をこのスライドに示します．Partial Evaluator をメタとして，Interpreter をオブジェクトとして部分計算すればコンパイラが生成できることは既に示されていますが，このコンパイラをメタとして，知識，

ルール，又はプログラムをオブジェクトとして，段階的に追加して部分計算すればコンパイラを自動生成できます．このことはICOTの研究者であった竹内によって提案されました．これをインクリメンタルなコンパイリングと呼んでいます．これは段階的にコンパイラを高度なものにしてゆくことができることと，しかも，前段に入れたオブジェクトプログラムはそれ以降は不要であるという利点があります．このことはprologの重要な特徴の一つである段階的プログラム開発の容易化とうまく調和しています．

また，このスライドに示したPartial Evaluator Peval'はコンパイラ・コンパイラに位置着けられます．ここで，ICOTの藤田はPartial Evaluator Peval ＝Partial Evaluator Peval'である自己適用可能なPartial Evaluator が pure prolog については存在していることと，実際のプログラミングによりその形を示しました．

現在，さらに，コンパイラの自動化と応用に向けた基礎研究のために，Partial Evaluator の実験プログラムを作成しております．

また，ＧＨＣへの拡張についても検討を開始したところであります．

S12. つぎにこのように，プログラムや知識が段階的に加わってゆく場合に，ある段階で，どうしても，ある推論を行って見たいとか，プログラムを走らせてみたいというようなことが起ります．この場合，その時点で情報が充分でないことが有り得るわけですが，足りない部分は常識的に解釈して，なんらかの，妥当な推論を行なえれば便利であります．このような高次の推論を行なうための概念のモデルとしてascription（帰着）がICOTの有馬によって新しく提案されました．

このモデルはMcCarthyによる非単調推論の形式であるcircumscription の自然な拡張であります．

このスライドに示す，Ｋとは実体を要素とする集合であり概念とか性質とか知識であります．Ｋmin とはすべての要素が‘Ｋである’と分かっている領域であり，Ｋmax とは‘Ｋでない’と分かっているものをすべて除いた領域であります．もう少し違った言い方をすれば，Ｋmin とは正しいと証明されていることしか信じない解釈であります．また，ｋmax とは，否定されてい

ないものはすべて正しいとする解釈であります．ある性質Ｋの解釈において，Ｋmin とＫmax との間のどこかに在る，常識的解釈，すなわちたとえばこのスライドで示した破線に帰着させようとするモデルであります．
このAscription，すなわち，帰着　のモデルを，高次推論を用いた知識獲得モデルとか学習モデルに応用する研究も着手したところでございます．

以上が問題解決推論モジュールと知識プログラミング言語におけるこれまでの成果の主なものであります．

Ｓ１．　最後になりますが，つぎに知的プログラミングシステムのお話に入ります．

Ｓ２．　定理の自動証明系，証明の自動検証については，今から３０年頃前に命題論理の定理を証明するプログラムについて発表されたのがこの分野の出発であると言われております．定理の自動証明系とは“命題が与えられたとき，その命題が正しければ，いつかは必ずその証明を作り上げるシステム”のことであります．このシステムが存在していることを原理的に保証したのがＧｅｄｅｌの完全性定理であります．また，命題論理の範囲でも，計算量として多項式時間内では処理できない証明が存在していることも知られていますが，私共は，もちろん，システムの存在が原理的に保証されている範囲内で，定理の自動証明系を取り扱かおうとしています．
また，このシステムの開発を通じて，定理とプログラム仕様，証明作成とプログラム作成を対応づけながら証明からアルゴリズムまたはプログラムを生成する証明コンパイラの研究開発を進めているところでございます．

Ｓ３．　証明支援システムの研究としては当初から，
（１）　線形代数（Linear Algebra）
（２）　記号算術（Symbolic Airthmetic）
（３）　微分幾何

の３つの分野を選択してスタートしてまいりました．このうち，線形代数についてはＣＡＰ－ＬＡとして具体的システムの試作を進めて参りました．ですからここでは，ＣＡＰ－ＬＡを基本にしてお話を進めます．まず，数学的証明の２大要素と致しましては，計算式で使用する等号「＝」の処理と，論理式で使用する論理結合子「and,or,imply,not」の処理が出きることが必要になってくるわけですが，これらが扱えて，またこれを用いて記述された対象をコンピュータが理解できる必要があります．ですから，このための言語を定義する必要があります．で私共は，構文規則が明解で数式や論理式が表現できる形式的言語ＰＤＬ(Proof Description Language)を設計しました．ＰＤＬではall,someの他，対象のtype宣言が可能であることを，また，Gentzen のＮＫを採用して導入規則と，除去規則の記述表現を可能としております．このＰＤＬによって古屋茂著の「行列と行列式」の中に出てくる証明をほとんど記述できることを実証しております．

Ｓ４．　このＰＤＬを基本として等号と論理結合子の２大要素の処理系からなるＣＡＰ－ＬＡシステムを開発していますが，目標としている能力としては，大学初年程度ぐらいであります．要するに，数学する人々のよりよきノート代わり，さらにはパートナーとなるシステムを開発して行こうとするものであります．

具備している機能としては，

証明記述の支援

証明の論理展開のチェック

証明のデバック

などが挙げられます．

その他，ＰＤＬの範囲で日本語表示による入出力機能を充実させています．

Ｓ５．　システム構成図はこのスライドに示しましたように証明チェッカ，証明エディタ，論理知識ベースおよび入出力サブシステムから構成されます．証明チェッカは，Metis と呼んでいて，それ単独でも使用できる項書き替えサブシステムを内部にもっています．本年度から，高階の記述およびメタ記述を取

り入れて証明記述言語の拡大を図り，証明する対象範囲拡大のフィジィビリティの検討を行ってゆきます．また，会話的な検証および証明戦略のユーザ定義を可能とするためにエディタ，チェッカを高度化します．さらに，論理知識ベースの完備化を行いまして，構築した論理の登録，削除，検索を中心に理論知識ベースの管理，また，理論間の継承とか依存関係のチェックを行なえるようにします．

Ｓ６．　項書換え系については特に，等式の書き換えの規則化を行っておりまして，書き換えの停止性を保証しています．また，等価な項が必ず同一項に書き替えられるように曖昧性が除去できる Knuth and Bendix アルゴリズムが取り入れられています．
これらの実際の動きについては，デモンストレーションプログラムが用意されておりますので是非御覧になって頂きたいと思います．

Ｓ７．　つぎに，まだほんの卵の段階ですが，新しい２つの研究テーマについて少し紹介致します．

第１番目は，構成的論理すなわち，背理法や排中律を用いない証明のクラスを取り扱い，このクラスの証明から実現性すなわちreaslizabilityの解釈を通して，証明アルゴリズムをさらにはプログラムを自動生成させようとするシステムであります．このテーマの推行に当たっては，このスライドに示したように先ずこれまでの資産であるＣＡＰ－ＬＡシステムを実験ツールとして位置づけてゆくことにしております．

Ｓ８．　２つ目の新しい研究テーマとしては，制約ロジックプログラミングから証明系を作成しようとするものでございます．これを現在，ＣＡＬと名付けたところでありますが，制約として代数方程式を持つロジックプログラムであり，その構文規則はこのスライドに示したようにプログラム節のBody部　L_1・・・L_n　の各　L_i がリテラルか，または，制約であるかを許しているものであります．

S９．　ＣＡＬインタープリタについては，現在，実験システムを開発中であります．構成概念図は，このスライドのようになっております．大きく分けて，代数方程式処理系とロジック処理系から構成されます．代数方程式処理系ではBuchbergerアルゴリズムを用いて制約として与えられた代数方程式のGrobnerベースを求めております．このＣＡＬが持つ論理的潜在能力についての評価及びその適用分野についても調査中であります．

S10.　最後に，前期から開発を行ってきていますので既にご存じであると思いますが，Prologプログラムの検証システムArgus を紹介致します．

Argus はプログラマが意図している論理的性質を持つかどうかを自動的に証明しようというものでありまして，その具体的な機能については，このスライドに示しましたように，検証，解析，作成，修正の機能から成り立っております．

Argus は，与えられたPrologプログラムの性質を示す論理式を証明するのに，そのプログラムの拡張実行と帰納法的証明に必要なヒューリスティック制御の自動化を行っております．このようなプログラムの検証のためにＬＩＳＰを対象としたBoyer Moore のTheorem Proverがありますが，Prologプログラムにこれに対応する新しい方式を三菱の金森らが導入しました．この方式は，おおまかに言えばBoyer Moore の方式よりもヒューリスティック制御が単純にできることなどが特徴であります．

さらに，この手法を並列ロジックプログラムにまで拡張するための検討に着手したところであります．

このシステムについてもデモンストレーションプログラムが準備されておりますので，是非ご覧になって頂きたいと思います．

以上が基礎ソフトウェアシステムのお話でございます．
時間の都合で随分急いだために一部は表面的なお話になってしまったたことと，成果の全部を紹介出来なかったことをお詫びします．また，

最後に，本日お話しましたこれらの研究開発に関連したワーキンググループの方々の日頃の熱心なご討論に謝意を表して，私の話を終わらせていただきます．

S1

基礎ソフトウェア・システム

— 概　要 —

1987. 6. 9

ICOT　研究所　第1研究室

S2

基礎ソフトウェア・システム

(1)　該言語・問題解決推論モジュール

(2)　知的プログラミング・システム

S3

Guarded Horn Clauses (GHC)

・汎用並列プログラム言語

・GHC＝論理プログラミング

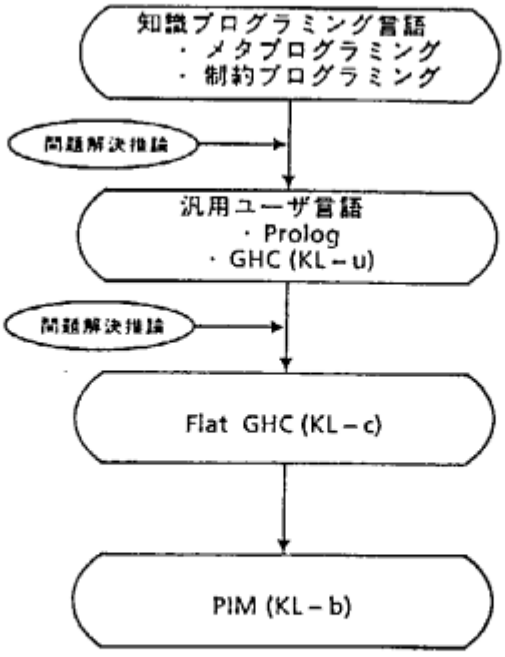
+ データフロー的並列制御
− 全解探索機能

} ガードにより実現

・データフロー、プロセス記述の両パラダイム

・通常の論理型言語の下位言語

S4　問題解決推論モジュールと核言語



S5　(1) 問題解決推論モジュール
プログラム変換

・メタプログラミング

・部分計算

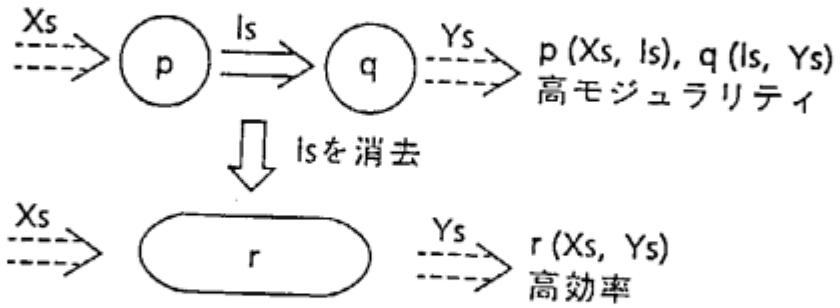
S6　プログラム変換の最近の成果

・プロセスの融合

・コルーチン (疑似並列) 制御の逐次制御への変換

・全解収集プログラムの決定的プログラムへの変換

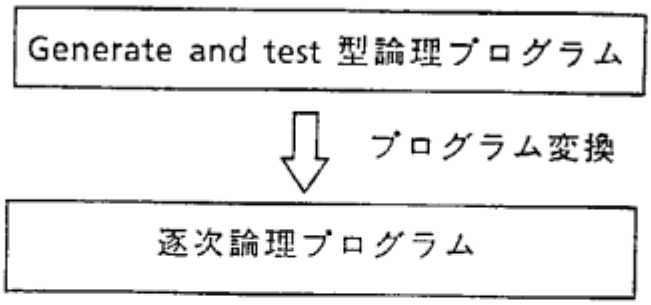
S7　GHCプロセスの融合



・部品化プログラミングと効率の両立
・処理系の特性に合ったプロセス粒度の選択

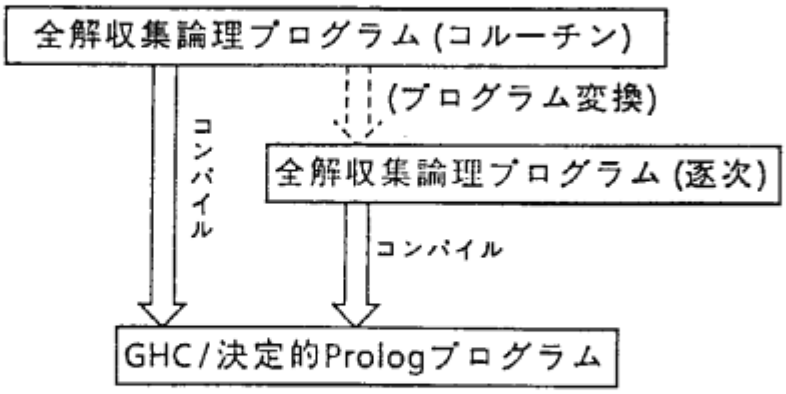
S8

コルーチン制御 ⟶ 逐次制御



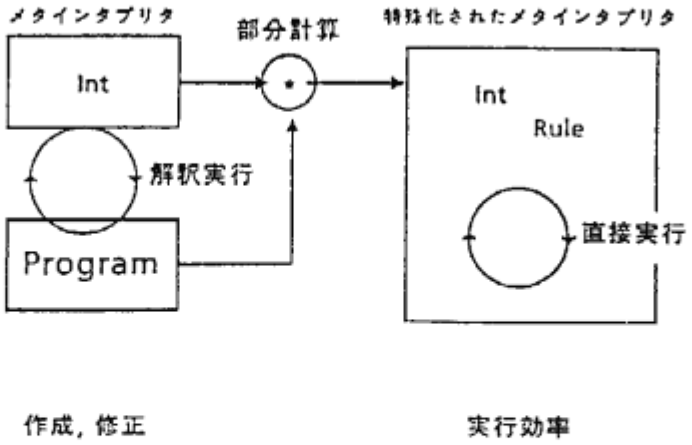
・従来の論理プログラムの変換手法の拡張

S 9 全解収集プログラム ⟶ 決定的プログラム



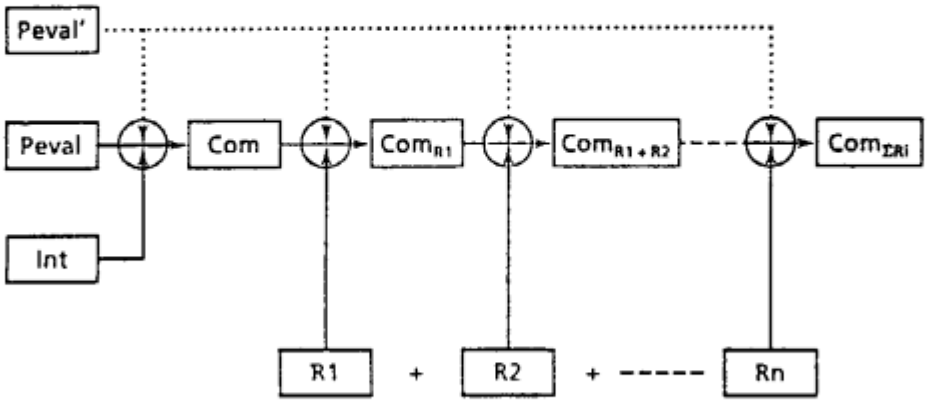
・モード解析による自動変換
・GHCで、検索問題の並列実行

S 10 メタプログラミングと部分計算



S11

段階的コンパイリング

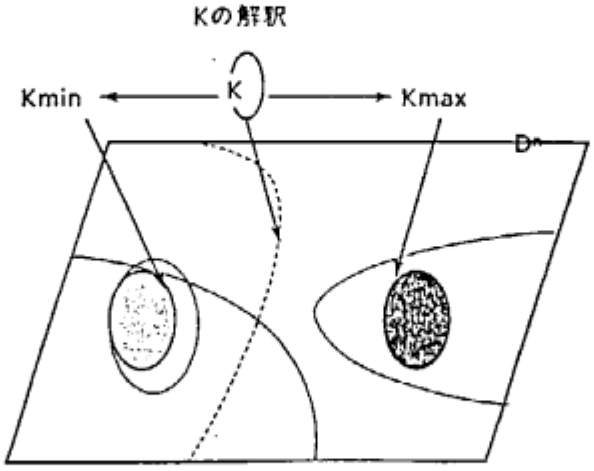


S12

Ascription (帰着)

K：概念、性質　　($K<D^n$)

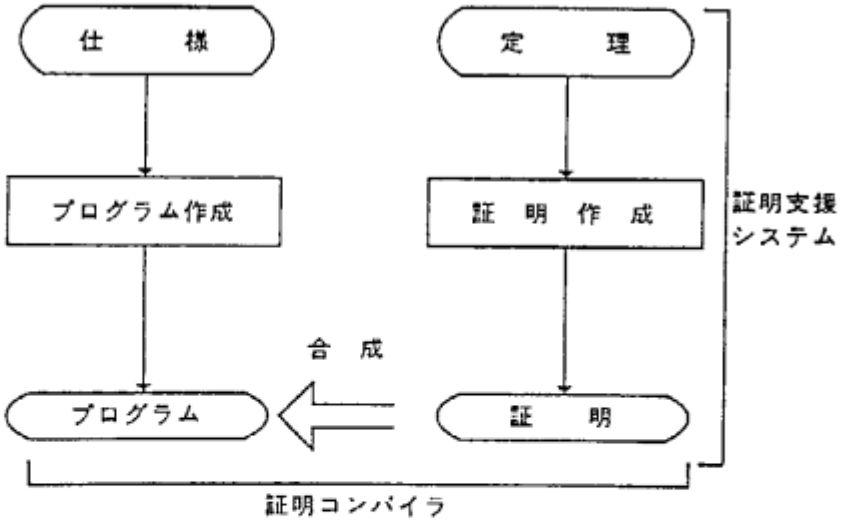
D：領域（実体の空間）



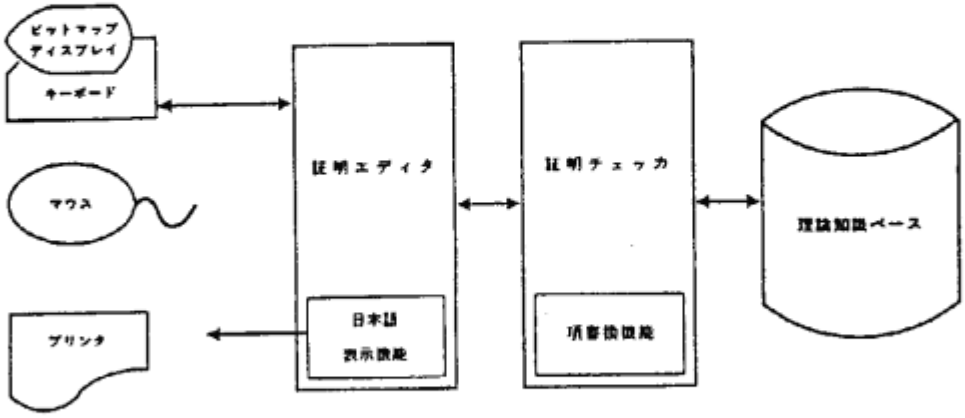
S1　　基礎ソフトウェア・システム

(2) 知的プログラミング・システム

S2　　プログラミングと定理証明



S5　　証明支援システム構成図



S6　　項書換えシステム

- 曖昧性の検定および除去手続きの改善
  （Knuth & Bendix の完備化手続き）
- 数学的帰納法への応用
- 証明支援システムへの応用

S3　数学的証明の2大要素と言葉

○計算 … 等号「＝」の処理
　項書換えシステム

○論理 … 論理結合子「∨, ∧, ⇒, ¬」の処理
　証明支援システム

○PDL　(proof description language)

S4　CAP－LAシステム

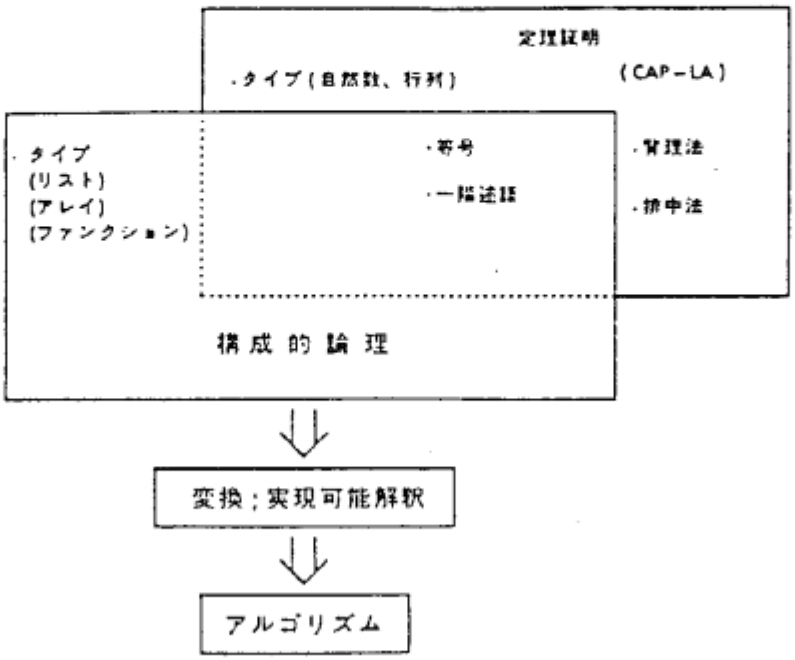
目的：　大学初年級程度の線形代数の理論構築の支援

機能：　証明記述の支援 (エディタ)

　　証明の理論展開のチェック
　　(チェッカ、理論知識ベース)

　　証明の誤り訂正の支援 (証明デバッガ)

S7　　アルゴリズム生成システム



S8　　制約プログラミング(代数式 + ロジック)

CALの構文

・プログラム　　プログラム節の集合

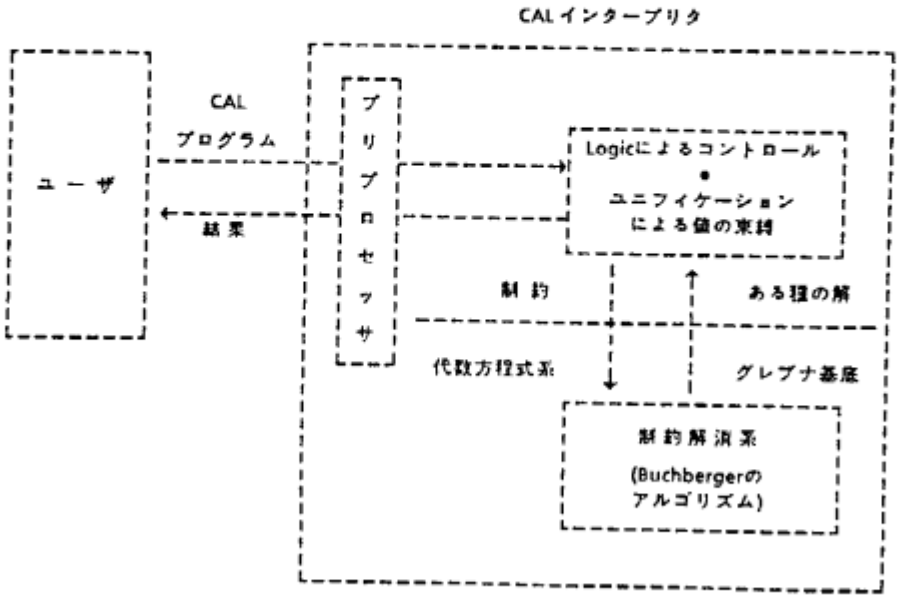
・プログラム節
　H - - > L1, L2, ..., Ln.
　　Li: リテラルまたは制約

・制約
　LHS = RHS
　　LHS, RHS: 代数式

・単位節
　H - - >.

・ゴール節
　- - > L1, L2, ..., Lm.

S9　　CALインタープリタ構成



S10　　知的プログラミング実験システムArgusの構成